

THE UNIVERSITY OF CALGARY

Computational Complexity Applications in Machine Learning

by

Christino Theodore Tamon

A DISSERTATION

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

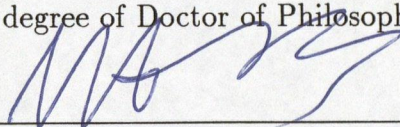
CALGARY, ALBERTA

AUGUST, 1996

© Christino Theodore Tamon 1996

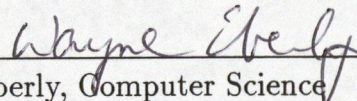
THE UNIVERSITY OF CALGARY
FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a dissertation entitled "Computational Complexity Applications in Machine Learning" submitted by Christino Theodore Tamon in partial fulfillment of the requirements for the degree of Doctor of Philosophy.



N. H. Bshouty, Computer Science

University of Calgary



W. Eberly, Computer Science

University of Calgary



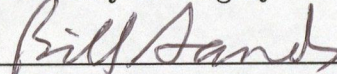
B. A. MacDonald, Computer Science

University of Calgary



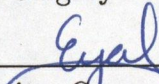
J. R. B. Cockett, Computer Science

University of Calgary



G. W. Sands, Mathematics and Statistics

University of Calgary



Eyal Kushilevitz, Computer Science

Technion Institute of Technology

Date August 15, 1996

Abstract

This thesis presents some theoretical investigations on the learnability of Boolean functions using techniques from computational complexity theory. Some of the main findings are as follows.

- Monotone Boolean functions are learnable in the probably approximately correct (PAC) model in subexponential time under product distributions.
- Boolean functions computable by some classes of branching programs of *width less than five* are efficiently learnable in the exact identification model using equivalence and membership queries.
- Boolean functions computable by polynomial size circuits and by polynomial size formulas in normal forms are efficiently learnable in the exact identification model using equivalence queries by a probabilistic oracle Turing machine that has access to an \mathcal{NP} oracle.

The first result is obtained through a new analysis of the harmonic spectra of monotone Boolean functions. The second result is in contrast to the cryptographically impossible task of learning width *five* branching programs and is obtained through a combination of techniques from harmonic analysis and automata theory. An implication of the third result is that if Boolean formulas in normal forms are not efficiently exactly learnable from equivalence queries then $\mathcal{P} \neq \mathcal{NP}$.

Dedication

*This thesis is dedicated
to memory of my aunt
Ria Halim-Dihardja*

Acknowledgments

First and foremost my deepest thanks to my supervisor Nader H. Bshouty. Nader gave me a chance, believed in me, taught me many wonderful things about research, and was extremely generous. This thesis would not have been possible without his guidance and patience. Thank you, Nader!

My research has been strongly influenced by many people. Nader has had a hand in all of my research, my way of thinking, and my understanding of learning theory. He got me started on our first paper on learning with the \mathcal{NP} oracle. I was overwhelmed with the ideas that he poured on me: the Monotone Theory (where we discovered together lattice theory lurking behind it), trading equivalence for the NP oracle (where I noticed self-reducibility), trading unlimited time for the NP oracle plus random bits (where my knowledge of probabilistic arguments is really being stretched), and lower bounds (where Nader's instincts are unparalleled). In our second paper, once I saw Cauchy-Schwarz, Nader kept me on the right tracks for the rest of the trip. The analysis that we worked out together in Section 4.3. was quite a nail-biting experience for me. Nader was also very supportive when I got carried away with cross correlation. Finally, in our third paper, my ramblings that the number of sinks has something to do with *rank* triggered Nader's enthusiastic response that resulted in our joint work on learning branching programs. I very much enjoyed our ensuing collaboration where we pulled resources from the kitchen sink, e.g., Fourier, multiplicity automata, lists and nested differences.

My thanks to Jeff Jackson, who graciously shared his broad knowledge on Fourier transform. Jeff showed me another proof of Fact 3.7 that prompted me to look for other neat Fourier things. The fact that I stumbled on Cauchy-Schwarz is minor compared to the original enthusiasm that Jeff inflicted upon me. Jeff also helped me when I was grappling with applying lower bound on influences to weak learning monotone functions.

I would like to thank my coauthors from whom I have learned a lot. My thanks to Richard Cleve, Ricard Gavaldà, and Sampath Kannan, for collaborating with me on the oracle paper. Their broad knowledge in complexity theory, especially things related to random generation, amplifiers, and oracles in general, was an inspiration to me. My thanks to Francesco Bergadano and Stefano Varricchio for collaborating with me on the branching program paper. I thank them, especially Stefano, for being very patient when I was struggling to understand multiplicity automata and for correcting my original mistakes.

My thanks also to numerous people who have kindly helped with suggestions, comments, improvements, and extensions on research appearing in this thesis. Thanks to David A. Mix Barrington for graciously sharing his knowledge on permutation branching programs, to Dan Boneh for suggesting *influence norm*, to Eyal Kushilevitz and Yishay Mansour for pointing out *average sensitivity*, to Sleiman Matar for asking many good questions, to Osamu Watanabe for kindly us allowing to include in [BCG⁺] his beautiful observation, to the anonymous referees of [BCG⁺, BT95] for comments that greatly improved the presentations of the papers, and to Atsuyoshi Nakamura for discussions on learning branching programs.

My thanks to Professors Wayne Eberly, Bruce MacDonald, Bill Sands, Richard Cleve, Robin Cockett, and Eyal Kushilevitz, for serving in of my supervisory and/or examination committee. Their keen and sharp comments have greatly improve the presentation of this thesis. My thanks especially to Wayne who tirelessly read and corrected a manuscript of this thesis.

My thanks to some very nice people from computer science office: John Aycock, Bev Frangos, Elsie Mason, Beverley Shevchenko, Lorraine Storey. Without them I am sure some of my troubles will be quite nightmarish.

Many kind people have made life felt more than just academic work. Thanks to Nadera and Vivian Bshouty for their hospitality, to Sleiman Matar who is always a good friend. Thanks to my Botany connection, Roger and Val Mandel, Jin-Hao Liu and family, Jacqui Purvis-Smith, and Simon Chuong, for their friendship and for being there. Thanks to people from computer science, my ex-lunchmate Pete Vesely, the theory crowd: Lynn Burroughs, Dave Wilson, Zhang Wei, Jola Warpechowska-Gruca, and Nathaly Verwaal; my squash enemies, Robin Cockett, Camille Sinanan, and Roberto Flores-Mendes; and also to Todd Reed, and the Gomes family.

My thanks once again to my mentor and friend, Hisao Tamaki, whose friendship, encouragement, help, and advice were crucial in so many subtle ways. It was Hisao who was responsible to point me back to Calgary ... where apparently many wonderful things await me. Thanks to my Gaelic muse, Catherine (a.k.a. Baby Leo) Convery, whose letters kept me close with an Ireland that I will someday visit.

My thanks goes also to my family in Calgary who made my long journey away from home possible and for their patience with this difficult relative; my uncle, Indraman, my cousins, André, Manda, and Yuswan, and my adorable nieces, Elise and Celine.

I would like to thank my Mom Thea Lierungan, for her love and support, despite the long distance and time away from home, and my lazyness in writing letters. We made it, Mom! Finally to my partner in life, Siew Hwee, whose love, encouragement, support, and entourage (BearBear, Teebo, Mooky, Odie), were essential in keeping me insanely happy (mostly, not in that order), my big thanks. It requires more than randomness and an \mathcal{NP} oracle to find a life partner, but perhaps it was more than an oracle that made it happen.

Publication Notes

An extended abstract describing the results in Chapters 3 and 4 was published in *Proceedings of the 27th Annual ACM Symposium on Theory of Computing 1995* [BT95], a journal version of which will appear in *Journal of the ACM*, vol. 43, no. 4, July 1996. This research is joint work with Nader H. Bshouty. A small part of Chapter 3 also appeared in [T95].

The results of Chapters 5 were described in [BBTV96]. This research is joint work with Francesco Bergadano, Nader H. Bshouty, and Stefano Varricchio, and is being submitted for publication.

The research in Chapter 6 is joint work with Nader H. Bshouty, Richard Cleve, Ricard Gavaldà, and Sampath Kannan. An extended abstract describing this research appeared in *Proceedings of the Seventh Annual ACM Workshop on Computational Learning Theory 1994* [BCKT94], a journal version of which appeared in *Journal of Computer and System Sciences*, vol. 52, no. 3, June 1996 [BCG⁺].

Contents

Approval Sheet	ii
Abstract	iii
Dedication	iv
Acknowledgments	v
Publication Notes	viii
Contents	ix
List of Figures	xi
Glossary of Notation	xii
Chapter 1. Introduction	1
Chapter 2. Preliminaries	7
2.1. Complexity Theory	7
2.2. Computational Learning Theory	10
2.2.1. Representation Classes	14
2.2.2. Probably Approximately Correct Learning Model	15
2.2.3. Exact Identification Learning Model.....	17
2.3. Specific Representation Classes for Boolean Functions	19
2.3.1. Boolean Circuits and Formulae	20
2.3.2. Decision Programs	21
2.4. Inequalities and Probabilities	24
Chapter 3. Harmonic Analysis of Boolean Functions	26
3.1. Basic Theory of Fourier Transform	28
3.2. Influence and Average Sensitivity	31
3.3. Relating Fourier Spectrum and Average Sensitivity.....	32
Chapter 4. Learning Monotone Functions in the PAC Model	39
4.1. The Linial-Mansour-Nisan Learning Algorithm.....	40
4.2. Subexponential Learning for All Monotone Boolean Functions	43

4.3. Analysis of Learning under Any Product Distributions	44
4.4. Proving Near Optimal Performance	48
4.4.1. Error Rates	48
4.4.2. Bounds for the Low-degree Fourier Algorithm	50
4.4.3. Sample Complexity	51
4.5. Polynomial-time PAC Learning	52
Chapter 5. Learning Bounded Width Branching Programs	57
5.1. Characterizations of Width Two Branching Programs	58
5.2. Learning <i>Monotone</i> Width Two Branching Programs in the PAC Model ...	59
5.2.1. The Harmonic Sieve Learning Algorithm of Jackson	60
5.2.2. A Fourier Correlation Lemma.....	62
5.3. Exact Learning Width Two Branching Programs with $O(1)$ Sinks	65
5.4. Exact Learning <i>Permutation</i> Branching Programs	69
5.4.1. Multiplicity Automata	70
5.4.2. Transformation to Small-depth Circuits	73
Chapter 6. Learning Boolean Functions with the NP Oracle	78
6.1. Uniform Generation of Polynomial-time Structures	79
6.2. The Halving Algorithm Revisited	81
6.3. Exact Learning with the Equivalence and NP Oracles	87
6.4. Exact Learning with the Membership and NP Oracles	88
6.4.1. Trading Unlimited Time for the NP Oracle.....	89
6.4.2. Trading the Equivalence Oracle for the NP Oracle	91
Chapter 7. Conclusions and Open Problems	98
7.1. Summary	98
7.2. Minor Extensions	100
7.3. Open Questions	101
Bibliography	103
Closing Credits	110

List of Figures

1.1	Example of a DNF formula	2
1.2	Some classes of Boolean functions considered in the thesis	6
2.1	Probabilistic Oracle Turing Machine.....	8
2.2	Two-person Learning Game.....	11
2.3	The PAC Learning Model	16
2.4	Example of a Boolean Circuit	20
2.5	Example of a Decision List	22
2.6	Example of a Branching Program	23
4.1	The Linial-Mansour-Nisan algorithm.....	41
5.1	Example of a strict width 2 branching program	58
5.2	Example of a monotone width two branching program for DNF $x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2$	59
5.3	Example of S_3 -PBP computing the identity permutation	69
5.4	Example of \mathbb{F}_2 -automata for DNF $x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2$	71
5.5	Examples of $mod_q\text{-}\widehat{LT}_1$ and $mod_q\text{-}mod_{\mathbb{Z}}$ circuits	75
5.6	Barrington's circuit characterization of S_3, A_4 -PBPs	76
6.1	Examples of amplifiers	85
6.2	The δ -Halving algorithm using amplifiers.....	86
6.3	Monotone of $x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2$	92

Glossary of Notation

Early lower case letters, such as a, b, c, \dots , are normally used to denote absolute constants. The end lower case letters, such as \dots, x, y, z , are used to denote variables, strings, bit vectors, etc. Middle lower case letters, such as f, g, h, \dots are used to denote functions. Other lower case letters are used in a first-come first-need fashion for almost anything. Capital letters are reserved for sets, events in some probability space, etc. Greek letters are used sparingly for special functions, but mainly in use for asymptotic notations.

$ \alpha $	set cardinality, string length, Hamming weight, or absolute value
$[\Upsilon]$	1 if statement Υ is true, 0 otherwise
\doteq	definition symbol
\mathbb{R}, \mathbb{Z}	field of real numbers, ring of integers (resp.)
\mathbb{F}_q	finite field of q elements
$[a, b]$	integer (or real) interval between a and b (inclusive)
$[n]$	$\{1, 2, \dots, n\}$
$(a, b]$	the real interval between a (excluded) and b (included)
$\{0, 1\}^n$	set of Boolean n -bit vectors
a_i	i th bit of $a \in \{0, 1\}^n$ or i th element in a list
$e_i \in \{0, 1\}^n$	n -bit vector that is zero everywhere except for the i th bit which is 1
$0_n, 1_n$	the all-zero, all-one n -bit vector in $\{0, 1\}^n$
$a \oplus b$	bitwise exclusive OR between $a, b \in \{0, 1\}^n$

$a \cdot b$	inner product between $a, b \in \{0, 1\}^n$, i.e., $\sum_{i=1}^n a_i b_i \pmod{2}$
$a \circ b$	concatenation of two bit strings
$a \leq b$	standard ordering on $\{0, 1\}^n$, i.e., $a_i \leq b_i$, for all $i \in [n]$ ($0 < 1$)
$f \leq g$	standard ordering on Boolean functions, i.e., $f(x) \leq g(x)$, for all $x \in \{0, 1\}^n$
$f \equiv g$	$f(x) = g(x)$, for all x
$\Pr_D[A]$	probability of event A under distribution D
$\mathbf{E}_D[X]$	expectation of random variable X with respect to distribution D
U_n	uniform distribution on $\{0, 1\}^n$
$\hat{f}(a), \tilde{f}(a)$	Fourier coefficient of function f at a
χ_a, ϕ_a	basis function at a according to uniform or product distribution (resp.)
$\ln x$	logarithm function to base $e = 2.7182818\dots$
$\log x$	logarithm function to some base (usually 2)
$e^x, \exp(x)$	exponential function in x
$H(x)$	binary entropy function, i.e., $x \log_2(1/x) + (1 - x) \log_2(1/(1 - x))$

Some notation to describe asymptotics is described next.

$f(n) \in \mathcal{O}(g(n))$	$\exists c > 0, n_0 \forall n > n_0 \ f(n) \leq cg(n)$
$f(n) \in \Omega(g(n))$	$\exists c > 0, n_0 \forall n > n_0 \ f(n) \geq cg(n)$
$f(n) \in \Theta(g(n))$	$\exists c_1 > 0, c_2 > 0, n_0 \forall n > n_0 \ c_1 g(n) \leq f(n) \leq c_2 g(n)$
$f(n) \in o(g(n))$	$\lim_{n \rightarrow \infty} f(n)/g(n) = 0$
$f(n) \in \omega(g(n))$	$\lim_{n \rightarrow \infty} f(n)/g(n) = \infty$
$f(n) \in \tilde{\mathcal{O}}(g(n))$	$f(n) \in \mathcal{O}(g(n) \log g(n))$
$f(n) \sim g(n)$	$\lim_{n \rightarrow \infty} f(n)/g(n) = c$, for some constant c
$f \ll g$	$f(n) \in o(g(n))$
$f \gg g$	$f(n) \in \omega(g(n))$

CHAPTER 1

Introduction

Consider a world containing robots and elephants. Suppose that one of the robots has discovered a recognition algorithm for elephants that can be meaningfully expressed in k -conjunctive normal form. Our Theorem A implies that this robot can communicate its algorithm to the rest of the robot population by simply exclaiming “elephant” whenever one appears.

– L.G. Valiant, Comm. ACM, 1984.

The complexity of learning Boolean functions has been studied for more than a decade. During this period the area of *computational learning theory* has matured into a distinct field of research, has developed its own theories, and has provided a clean theoretical model for studying learning problems. The area has borrowed results from statistics, complexity theory, cryptography, automata theory, and others. New techniques were developed to answer basic questions about learning Boolean functions and fruitful interactions with practical problems have emerged slowly.

Despite these efforts and successes, we are still unable to answer one of the earliest and basic learning questions posed by L. G. Valiant. Recall that a Boolean formula in *Disjunctive Normal Form* (or DNF for short) is a disjunction (logical OR) of conjunctions (logical ANDs) of literals (a variable or its negation). It is a fact that any Boolean function can be represented as a DNF formula, i.e., this representation is universal. In his seminal paper that founded computational learning theory, Valiant [V84b] asked whether Boolean formulas in Disjunctive Normal Form can be *efficiently* learned from random examples. The main emphasis here is on efficiency since learning

is always possible with an exponential amount of resources. This question is very interesting since DNF formulas are a very simple and yet universal representation for Boolean functions.

Valiant also noted that the same question for polynomial size Boolean circuits is a hopeless one if we make a certain cryptographic assumption. The existence of a family of cryptographic functions called *pseudorandom functions* eliminates any possibility of learning polynomial size Boolean circuits.

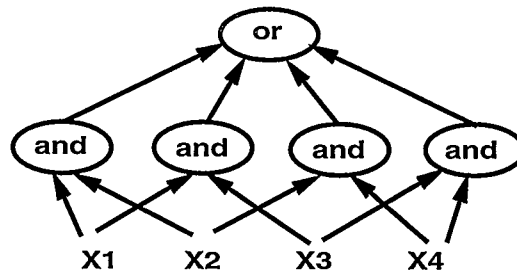


FIGURE 1.1. Example of a DNF formula

Half a decade passed before Kearns and Valiant [KV89] (and subsequently Angluin and Kharitonov [AK95]) strengthened the grip of Valiant's negative observation. They showed that, under reasonable and popular cryptographic assumptions, even a simpler class of Boolean circuits, called NC^1 circuits, is not learnable under a very generous learning model. NC^1 circuits are restricted to have logarithmic depth and polynomial size in with respect to their input size.

On the brighter side, a decade passed until Jackson [J94] proved the surprising and beautiful result that, under some additional relaxations of the learning model, the class of DNF formulas is learnable. This result constitutes one of the recent strongest positive answers to Valiant's question.

In this thesis Valiant's question and observation will be addressed from several different angles. First it is shown that monotone Boolean circuits (even of exponential size) are learnable from random uniform examples with a subexponential amount

of resources. This result is derived by proving that the *average sensitivity* of any monotone Boolean function is low. This implies that they have a simple harmonic spectrum that can be then exploited for learning purposes. Although the result on sensitivity is new, the connection between learning and harmonic spectrum is known from the seminal work of Linial, Mansour, and Nisan [LMN93].

Secondly, Jackson’s DNF learning result [J94] is strengthened in a natural way. It is shown that the class of *monotone width two branching programs* is learnable under the same assumptions made by Jackson. It is known [BDFP86] that the latter class includes the class of DNF formulas as a strict subset. The proof of this result relies on a non-trivial extension of a Fourier correlation lemma due to Jackson and is based on a recent characterization of width two branching programs as parity decision lists developed in [BTW96].

By switching the viewpoint from the Boolean circuit model (that includes DNF formulas) to the branching program model, the boundaries between learnability and non-learnability can be seen. Barrington [Bar89], in his seminal work, showed that the class of Boolean functions computable by width *five* (permutation) branching programs is equivalent to the class of Boolean functions computable by polynomial size and logarithmic depth Boolean circuits. Thus the negative cryptographic results of [KV89, AK95] eliminate any possibility for learning width five branching programs.

The investigation on learning branching programs is pursued to the case of widths *three* and *four*. It is shown that certain *permutation* branching programs of widths three and four are exactly learnable with equivalence and membership queries. The learning results rely on a recent breakthrough on the learnability and applications of *multiplicity automata* due to Bergadano, Catalano, and Varricchio [BCV96, BV94].

Thirdly in the thesis, it is shown that the class of polynomial size Boolean circuits and the class of polynomial size DNF formulas are exactly learnable from equivalence queries if the learning algorithm is equipped with an oracle that can solve any \mathcal{NP} problem. Allowing access to a computationally powerful oracle is a well-studied tech-

nique in structural complexity theory called *relativization*. Relativization is a means of studying how much help can an oracle provide in resolving typical hard questions such as $\mathcal{P} \neq \mathcal{NP}$. The result can be read backwards: if Boolean circuits or DNF formulas are not learnable then $\mathcal{P} \neq \mathcal{NP}$. The proof technique that is used relies on a preliminary observation of Kannan [K93] combined with a classic algorithm due to Jerrum, Valiant, and Vazirani [JVV86] for randomly generating combinatorial structures. Recently Watanabe observed that the learning result implies that if each language in \mathcal{NP} has a polynomial size circuit, i.e., $\mathcal{NP} \subseteq P/poly$, then the polynomial-time hierarchy (see [P94], Chapter 17) collapses to $\mathcal{ZPP}^{\mathcal{NP}}$. This improves a well-known theorem of Karp and Lipton [KL80] who proved a collapsing level of $\mathcal{NP}^{\mathcal{NP}}$ ([P94], page 431).

Those are three main lines from which Valiant’s initial questions are approached in this thesis. Elsewhere in the thesis more specific related results are pursued as well.

This thesis is organized as follows:

In Chapter 2, an overview of the relevant definitions, models of computation, and models of learning from complexity theory and learning theory are described. After defining some standard models of Turing machines, the two main learning models that are considered in this thesis — namely the *Probably Approximately Correct (PAC)* learning model of Valiant and the *Exact Identification* learning model of Angluin, are given in detail. Finally several specific representation classes for Boolean functions used in this thesis, such as Boolean circuits and formulas, decision trees and lists, and branching programs, are described.

In Chapter 3, the main results on the Fourier transform of Boolean functions are developed. First, some relevant definitions and standard facts in this area are given and an overview of known results is provided. Then the following results are proved: an upper bound on the *average sensitivity* for monotone Boolean functions and its impact on their Fourier spectrums. These results can be regarded as providing alternative (and perhaps simpler) proofs to a well-known result of Kahn, Kalai, and

Linial [KKL88] as well as providing a useful connection between the older notion of *average sensitivity* and PAC learning. The easy case of uniform distribution is stated first and then subsequently it will be extended to the more general case of product distributions.

In Chapter 4, main results from Chapter 3 are applied to derive results in learning. First a subexponential time learning algorithm for monotone Boolean functions is derived. This almost follows directly from the previous work by Linial, Mansour, and Nisan except for one detail. Some careful analysis is required to handle the case of *product* distributions which is slightly more problematic than the uniform distribution case. In a separate section, a careful analysis of the learning algorithm for monotone Boolean functions under product distributions is given. Next some evidence that the subexponential learning result is nearly optimal with respect to some parameters is shown. In the second part of this chapter, the attention is focused on efficient learning of monotone Boolean functions. Some improvements on the learning results of Kearns and Valiant [KV89] and of Sakai and Maruoka [SM94] are described.

In Chapter 5, ideas from Jackson's work on learning DNF formulas are applied to address the problem of learning Boolean functions computable by monotone width two branching programs (or MW_2 functions for short). To prove that MW_2 is learnable under the same learning assumptions a non-trivial extension of a key lemma due to Jackson is proved and a new characterization of width branching programs as parity decision lists given in [BTW96] is used. Subsequently in the chapter efforts are focused on showing that the general model of width two branching program (not necessarily monotone) is exactly learnable using equivalence queries assuming that there is only a constant number of sinks. The proof relies on combinatorial arguments due to Blum, Helmbold, Sloan, and Warmuth [Bl92, HSW90] on decision lists. Further in the chapter the learnabilities of bounded width *permutation* branching programs and of small depth Boolean circuits with modular and threshold gates are studied. The investigation is motivated by Barrington's work [Bar89] on the alternative character-

ization of NC^1 circuits as bounded width permutation branching programs and by a recent application of multiplicity automata in learning [BBBKV96].

In Chapter 6, a well-known algorithm of Jerrum, Valiant, and Vazirani [JVV86] for randomly generating combinatorial structures is used to obtain an exact learning algorithm for Boolean circuits. The learning algorithm that is obtained uses equivalence queries and requires access to an \mathcal{NP} oracle. This result implies that: if $\mathcal{P} = \mathcal{NP}$ then Boolean circuits are exactly learnable.

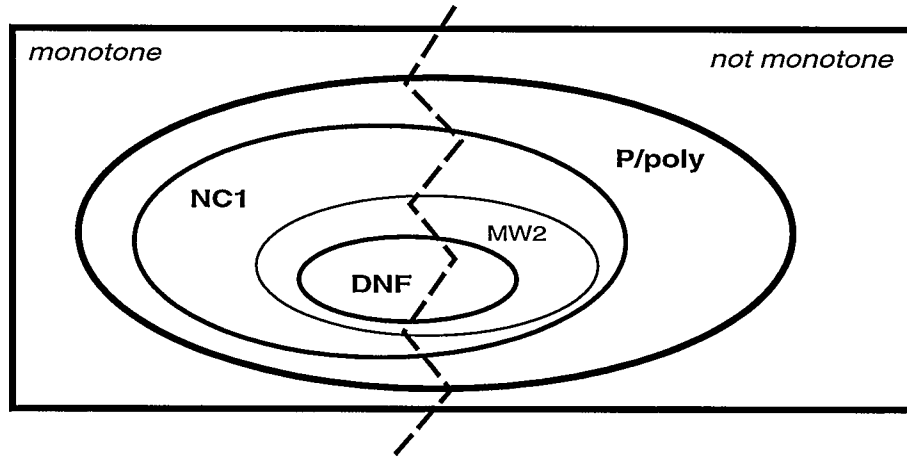


FIGURE 1.2. Some classes of Boolean functions considered in the thesis

Finally the thesis is summarized in Chapter 7, some minor extensions are mentioned, and some open questions raised by this work are stated.

CHAPTER 2

Preliminaries

In this chapter we describe relevant definitions and notions from computational complexity theory and computational learning theory. We focus our attention on problems related to the complexity and learnability of Boolean functions. To discuss the learnability of Boolean functions we describe some specific representation classes that we consider in this thesis, such as Boolean circuits and formulas, decision trees and lists, and branching programs.

2.1. Complexity Theory

We use the *Turing machine* as our model of computation. We assume the reader's familiarity with the basic models of *deterministic* and *nondeterministic* Turing machines (see, for example, [P94]). Recall that \mathcal{P} and \mathcal{NP} are the classes of languages accepted by polynomially time-bounded deterministic and nondeterministic Turing machines, respectively. Throughout this thesis we shall assume a *reasonable* encoding scheme, in the sense of Garey and Johnson [GJ79], when dealing with inputs or problem instances to Turing machines.

In this thesis we will also consider probabilistic and oracle Turing machines. A *probabilistic* Turing machine (PTM) is a standard deterministic Turing machine equipped with the ability to make decisions based on the outcome of a random coin flip. More formally, a PTM is a Turing machine which has a *coin-tossing* state and a special tape called the *random* tape. When the computation enters this coin-tossing state,

the machine receives a bit on the random tape that depends on the outcome of an unbiased coin flip.

An *oracle* Turing machine (OTM) is a deterministic Turing machine equipped with a special tape, called the *query* tape, and three special states, called the *query*, *YES*, and *NO* states, respectively. The computation of an oracle Turing machine requires that a set, called the *oracle set*, be fixed prior to the computation. If computation enters the query state and leaves some string w on the query tape, then computation switches to the *YES* state if w belongs to the oracle set or switches to the *NO* state if w is not in the oracle set. All other computations proceed as in the deterministic Turing machine case. If M is an oracle Turing machine and A is some oracle set then we denote M^A as the oracle Turing machine that has A fixed as its oracle set prior to computation. A *probabilistic oracle* Turing machine (POTM) is a Turing machine that is equipped with both the oracle and the random tapes.

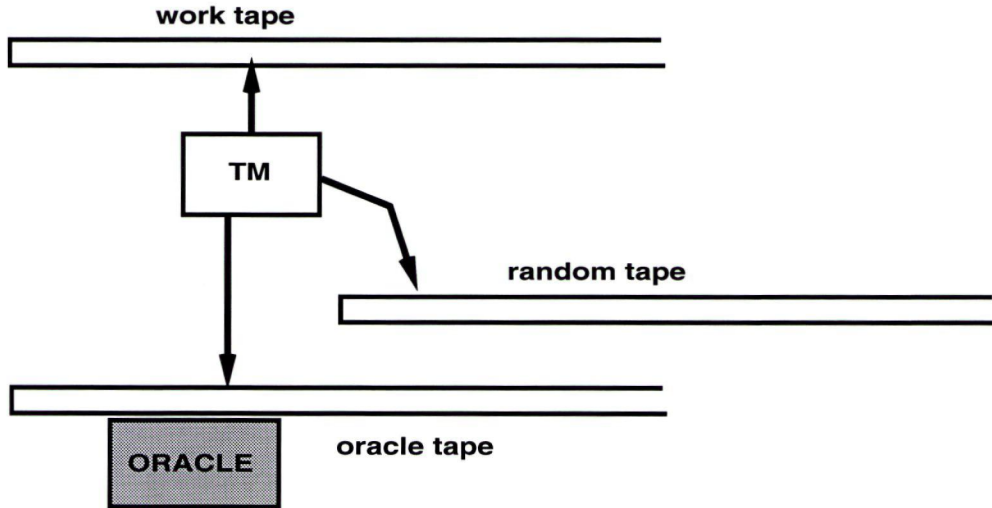


FIGURE 2.1. Probabilistic Oracle Turing Machine

A probabilistic Turing machine is said to run in *bounded polynomial-time* if there is a fixed polynomial function p such that for all inputs of length n the Turing machine

on input w always enters some final state after at most $p(n)$ computation steps. On the other hand we say that a probabilistic Turing machine runs in *worst case expected polynomial-time* if there is a fixed polynomial p such that the expected running time of the machine is at most $p(n)$, for every input of length n . The expected running time is taken with respect to coin tosses performed by the machine.

There are several well-known complexity classes that are associated with probabilistic polynomial-time Turing machines, but we will only mention one of them, namely \mathcal{ZPP} .

DEFINITION 2.1. A language L is said to be in \mathcal{ZPP} if there is a probabilistic Turing machine M running in bounded polynomial-time that satisfies the following. For every input x :

- If $x \in L$ then $\Pr[M(x) = \text{ accepts}] \geq 3/4$ or halts without output with probability at most $1/4$.
- If $x \notin L$ then $\Pr[M(x) = \text{ rejects}] \geq 3/4$ or halts without output with probability at most $1/4$.

We now define complexity classes related to oracle machines. The next informal description is taken from Johnson's survey paper [J].

If \mathcal{C} is a complexity class defined in some way, and A is a fixed oracle set, then \mathcal{C}^A is the analogous class defined using the same resource bounds but augmenting the machine model with a (perhaps additional) oracle tape for asking questions about membership in A .

For instance, we say that a language L is in $\mathcal{ZPP}^{\mathcal{NP}}$ if there is a probabilistic polynomial-time oracle Turing machine M , equipped with an oracle $A \in \mathcal{NP}$, such that, for every input x :

- If $x \in L$ then $\Pr[M^A(x) \text{ accepts}] \geq 3/4$ or halts without output with probability at most $1/4$.

- If $x \notin L$ then $\Pr[M^A(x) \text{ rejects}] \geq 3/4$ or halts without output with probability at most $1/4$.

When the meaning is clear from context we will say *an \mathcal{NP} oracle* rather than *an oracle $A \in \mathcal{NP}$* .

Our discussion above is slightly incomplete since we only describe *decision* problems whereas our learning algorithms solve *search* problems, i.e., algorithms that return outputs that are different from just *yes* or *no*. The notion of polynomial-time computation, whether it is deterministic, probabilistic, or relativized (oracle), can be extended in a natural way to search problems.

When describing algorithms in this thesis, we will in general use a very high-level description and omit the technical details of implementing these algorithms in the models described above. In some cases it might be helpful to consider *random access machines* (RAM) instead of Turing machines. Thus, we will informally use the words *algorithm* and/or *program* to mean a Turing machine or a RAM program.

2.2. Computational Learning Theory

The area that has become known as *Computational Learning Theory* was started in 1984 with the seminal paper of Valiant [V84b]. The chief contribution made by this paper is a clean and simple theoretical and computational model for which questions about learning can be posed and analyzed rigorously. Prior to Valiant's paper there was already ongoing theoretical machine learning research but most of these efforts tended to concentrate solely on computability issues rather than on computational efficiency.

In the following we give an overview sketch, using Valiant's DNF question, that is intended to motivate the learning models that we consider in this thesis. A more formal treatment of these models is given towards the end of this chapter.

One of the earliest but still the hardest open questions in learning theory was posed by Valiant. Recall that a *DNF (disjunctive normal form) formula* is a disjunction

(logical OR) of conjunctions (ANDs) of literals. Valiant asked whether DNF formulas are learnable in his proposed model. In one of its alternative formulations, this question can be stated as a two-person game between a teacher and a learner. The teacher holds a DNF formula on n variables, say ϕ , of size s (i.e., the number of ANDs is s). The only prior knowledge that the learner has is that the mysterious function ϕ is represented as a DNF formula, that it depends on n inputs, and that it has size s .

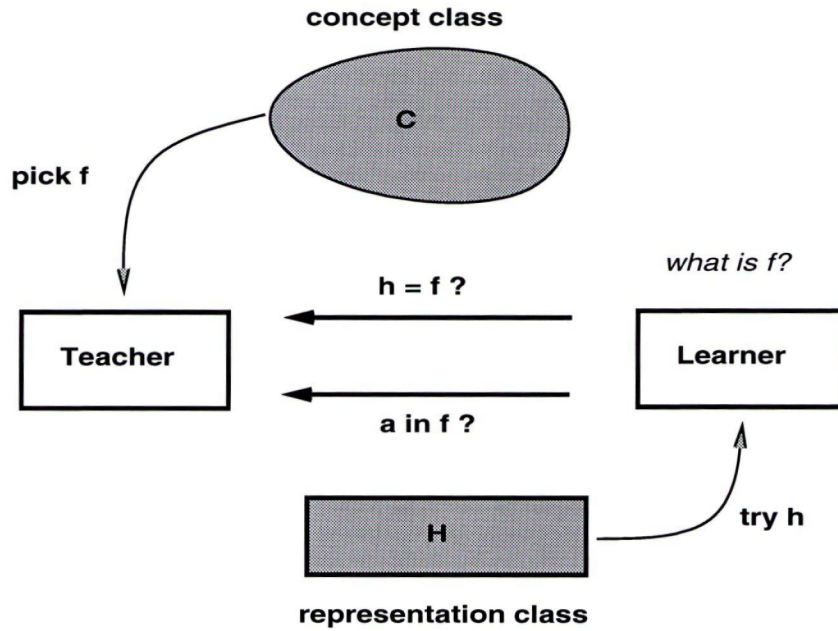


FIGURE 2.2. Two-person Learning Game

During the play of the game, the learner can ask two kinds of questions: it can either ask the teacher to return the value of ϕ on a specific assignment a of its choice or ask if another formula ψ is *equivalent* to the mystery function ϕ . In the first kind of question, the teacher replies simply with $\phi(a)$ while in the second type of question the teacher either responds with *yes*, signifying that ψ is equivalent to ϕ (which implies that the function has been learned), or with a *no* along with a *counterexample* b (such

that $\phi(b) \neq \psi(b)$). A question of the first type is called a *membership query* and a question of the second type is called an *equivalence query*. The learner wins if it can find a formula equivalent to ϕ in time proportional to a polynomial function of n and the *size* of ϕ (a binary encoding of ϕ). This model is called the *exact identification* model.

Notice that the learner can trivially succeed if allowed an exponential amount of resources. For instance, asking 2^n membership queries is sufficient to uncover the truth table of the DNF formula to be learned. Alternatively, asking as many equivalence queries as there are DNF formulas of size s , of which there are at most 3^{ns} , will surely discover the DNF formula to be learned. So the main thrust of Valiant's question is *efficiency*, i.e., whether the learner succeed using a polynomial number of queries and polynomial computation time.

Using combinatorial arguments, Angluin [Ang90] has proved that only using one type of question is not enough, i.e., neither membership nor equivalence queries alone are sufficient to learn DNF formulas. It is not known to date, whether both types of queries are sufficient. In another interesting paper, Angluin and Kharitonov [AK95] proved that under cryptographic assumptions the existence of membership queries in learning DNF formulas is irrelevant. That is, they proved that either DNF formulas are learnable without membership queries or DNF formulas are not learnable with membership queries. These two results, one with a combinatorial flavour and another with a cryptographic flavour, constitute the current knowledge about learning DNF formulas in the exact identification model.

Judging from these results, we were tempted to ask what is the minimum computational requirements sufficient to learn exactly DNF formulas. In Chapter 6, we show that randomization and a helpful \mathcal{NP} oracle are sufficient to guarantee the exact learning of polynomial size DNF formulas exactly. In fact our techniques imply the learnability of polynomial size Boolean circuits as well. A consequence of the statement of our result is that if DNF formulas are not exactly learnable then

$\mathcal{P} \neq \mathcal{NP}$.

We note that the learning formulation given above requires that the learner finds an *equivalent* function. There is another model that only requires the learner to find an *approximation* to the target DNF formula ϕ . More formally, the learner is asked to find, with probability at least $1 - \delta$, a formula ψ that satisfies

$$\Pr_D[\phi(x) \neq \psi(x)] \leq \epsilon$$

where D is some probability distribution on the domains of ϕ and ψ . In this model, the equivalence query is replaced with a *statistical* teacher who randomly draws points x according to D and supplies the learner with $(x, \phi(x))$. This model is called the *probably approximately correct* model or PAC model for short.

Intuitively, learning should be easier in the PAC model than in the exact model. This intuition is supported by the two results of Angluin [Ang88] and Blum [Bl94]. Angluin proved that an exact learning algorithm can be used to learn in the PAC model. Equivalence queries are replaced by a high probability sampling step; this can be achieved by appealing to standard large deviation bounds such as Chernoff bounds. Blum showed that, under a reasonable cryptographic assumption, there is a class of functions that can be learned in the PAC model but that cannot be learned in the exact model.

Kearns *et al.* [KLPV87] proved that if monotone DNF formulas are learnable in the PAC model then DNF formulas are learnable in the PAC model. Thus the monotonicity restriction does not make learning any easier. On the other hand, Angluin [Ang88] showed that monotone DNF formulas are exactly learnable with both equivalence and *membership* queries. In Chapter 4 we exhibit a subexponential time PAC learning algorithm for the class of all monotone Boolean circuits and any monotone DNF formula (including the ones that require exponential size). The algorithm does not require the use of membership queries but it requires that the examples are generated *uniformly*. In contrast, Chapter 5 provides a learning algorithm for

a superclass of DNF formulas in the PAC model with membership queries under a uniform probability distribution *in polynomial time*.

For more information about computational learning theory, we refer the reader to the survey paper of Angluin [Ang92].

2.2.1. Representation Classes. In the preceding section we have sketched an outline of the learning game played between the learning algorithm and the teacher or oracle. In those discussions we did not distinguish between a *concept*, i.e., a Boolean function, and a *representation* of a concept, e.g., a DNF formula that represents a Boolean function. In some cases this distinction need not be made but in most cases it is important to distinguish them. For instance, it is known that Boolean functions computed by DNF formulas with at most k terms (or ANDs) are not efficiently learned from random examples using DNF formulas with at most k terms [PV88] but are efficiently learnable using other representations, e.g., DNF formulas with at most n^k terms. Thus learning turns out to be a representation-sensitive phenomenon, and hence the need to differentiate between concepts and representations.

In this section we give a standard and formal treatment of representation classes for concepts or Boolean functions (see [Ang90] for a more complete treatment).

Let Σ be an alphabet, usually $\{0, 1\}$. The set Σ^* is the set of all strings over Σ and the set Σ^n is the set of all strings over Σ of length n . The length of a string $x \in \Sigma^*$ is denoted $|x|$.

A *representation class* $\mathcal{C}(\Sigma, \Delta, R, \mu)$ is a 4-tuple where Σ and Δ are finite alphabets, $R \subseteq \Delta^*$ and μ is a mapping from R to subsets of Σ^* . A *concept* is any subset of Σ^* . The set Σ^* is called the *example space* or *instance space*. An *example* is a pair (x, b) where $x \in \Sigma^*$ and $b \in \{0, 1\}$. The alphabet Δ is called the alphabet of representations. The strings of R are called the valid representations in the class \mathcal{C} . The function μ is a mapping from representations to concepts. The *size* of a representation $r \in R$ is $|r|$. The size of a concept c is $\min\{|r| : \mu(r) = c\}$. If a concept does not have a representation in R then its size is infinity.

The *concept class* C defined by a representation class \mathcal{C} is the set of concepts that have representations in R . That is, $C = \{\mu(r) : r \in R\}$. For any natural number n , we define the parametrized concept class C_n as $\{\mu(r) : r \in R, |r| \leq n\}$. Thus, $C = \bigcup_{n \geq 0} C_n$.

Example: Consider the class of Boolean functions computable by conjunctions of literals, also called the class of monomials. As an example, $f(x_1, x_2, x_3) = x_1 \wedge \bar{x}_2$ is a monomial. We now describe a representation class $\mathcal{R}(\Sigma, \Delta, R, \mu)$ for monomials. Taking $\Sigma = \{0, 1\}$, the concept f defines the subset $\{100, 101\}$ of $\{0, 1\}^3$. We can use some natural encoding of monomials as follows. The monomial $x_1 \wedge \bar{x}_2$ is represented as the string $10\star$ over the alphabet $\Delta = \{0, 1, \star\}$, where 1 means the variable appears unnegated, 0 means it appears negated, and \star means it does not appear. So the mapping μ maps the string $\langle 10\star \rangle$ to the concept $f = \{100, 101\}$. It also maps the string $\langle \star 1\star \rangle$ to the concept $g = \{010, 011, 110, 111\}$, i.e., $g = x_2$.

In most of the representation classes we consider there is a natural but implicit choice for each of Σ , Δ , R and μ . In most case considered in the thesis, $\Sigma = \{0, 1\}$, and the example or instance space is $\{0, 1\}^n$, for some fixed n . A standard encoding scheme (such as the ones in [GJ79]) can be assumed for the representations in $R \subseteq \Delta^*$. We will use the word *size of a concept* to mean size of the concept with respect to some natural representation class.

When dealing with a representation class $\mathcal{C}(\Sigma, \Delta, R, \mu)$ we will occasionally abuse the distinction between the concept class, i.e., $C = \{\mu(r) : r \in R\}$, defined by a representation class \mathcal{C} with the representations R from \mathcal{C} . That is, when the context is clear, we will use $r \in R$ (a representation) in place of the more precise $\mu(r)$ (a concept) when talking about concepts from C .

2.2.2. Probably Approximately Correct Learning Model. The Probably Approximately Correct (PAC) model was introduced by Valiant [V84b]. In this model, the learner receives from its environment a sequence of “random” classified or labeled

examples. The goal of the learner is to find a sufficiently good approximation or hypothesis that can explain the sequence of random examples that it has seen. The goodness measure of the hypothesis is the likelihood of a misclassification under the same random process that generated the examples.

In the PAC model the environment is modeled by an *example oracle* for the target concept, say f . Underlying the example space X (normally, $\{0, 1\}^n$) is a probability distribution D from which the example oracle draws its examples. The example oracle $EX(f, D)$ works as follows: upon request from the learner, it draws a random input $x \in X$ according to D and returns the classified example $(x, f(x))$. Unless otherwise stated we will assume that our instance or example space is $\{0, 1\}^n$, for some fixed n .

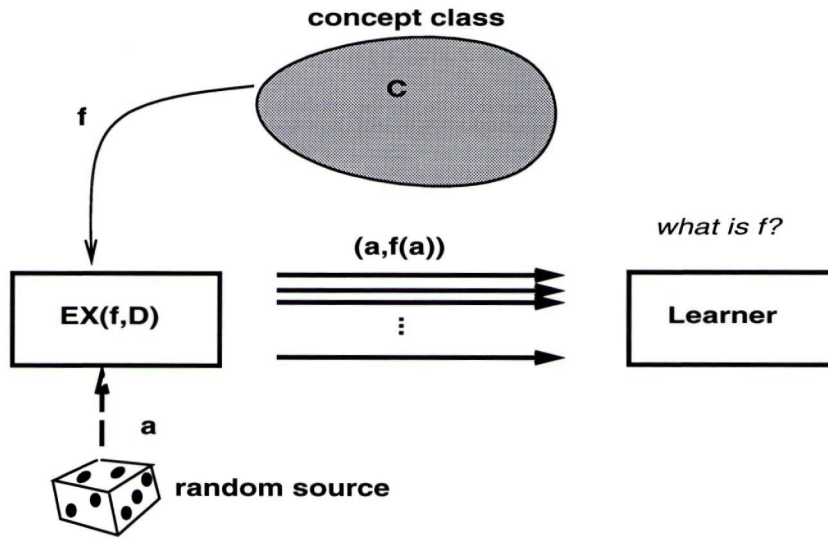


FIGURE 2.3. The PAC Learning Model

DEFINITION 2.2. (*PAC Learnable*)

Let C be a concept class and let H be a representation class. Then C is *PAC learnable using H* if there exists an algorithm A so that: for any concept $f \in C$, for any distribution D over X , for any $0 < \epsilon, \delta < 1$, if A is given access to $EX(f, D)$ and inputs ϵ, δ , then with probability at least $1 - \delta$, A outputs a hypothesis $h \in H$

satisfying $D(f \triangle h) \leq \epsilon$, where $f \triangle h = \{x \in X \mid f(x) \neq h(x)\}$. The last probability is taken over the possible internal randomization of A along with the randomization in the calls to $EX(f, D)$.

The last definition only stipulates that a “good” hypothesis can be found by the learning algorithm. It did not require that this hypothesis be found in a reasonable amount of time. The next definition of *efficiently PAC learnable* imposes this additional requirement.

DEFINITION 2.3. (*Efficiently PAC Learnable*)

Let C be a concept class and let H be a representation class. Then C is *efficiently PAC learnable using H* if C is PAC learnable using H and the learning algorithm runs in time polynomial in n , $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, and the size of the target function f .

Next we define the weak variation of the PAC learning model of Kearns and Valiant [KV89]. In this variation the learner is only expected to be able to approximate the target concept with an error that is slightly better than guessing.

DEFINITION 2.4. (*Weakly PAC Learnable*)

Let C be a concept class and let H be a representation class. Then C is *weakly PAC learnable using H* if C is PAC learnable using H with $\epsilon = \frac{1}{2} - \frac{1}{p(n,s)}$, where s is the size of the target concept f and p is a fixed polynomial function. It is *efficiently weakly PAC learnable* if the running time of the weak learning algorithm is polynomial in n , s , and $\frac{1}{\delta}$.

2.2.3. Exact Identification Learning Model. The *exact identification* learning model, or also known simply as the *exact* learning model, was introduced by Angluin in [Ang88]. This model differs from the PAC model in several ways. The exact learning model, as the name suggested, requires the learner to find a hypothesis that is logically equivalent or identical to the target concept. In this model, the learning algorithm interacts with a *teacher* who answers questions asked by the learner.

Let C be a concept class and H be a representation class over the same instance or example space X (usually $\{0,1\}^n$). We will define two different kinds of teachers or oracles that can answer questions from the learner. Before the learning process begins we fix a target concept f from C . The goal of the learner is to find $h \in H$ that satisfies $h \equiv f$.

We define the two most common oracles used in the exact model, namely the equivalence and membership oracles. An *equivalence oracle* $EQ_f(h)$ can answer equivalence queries, i.e., questions of the following form. The learner supplies the oracle with an input function $h \in H$ and the equivalence oracle answers with either *yes*, signifying that $h \equiv f$, or a counterexample $b \in X$ that is an assignment that satisfies $h(b) \neq f(b)$. A *membership oracle* $MQ_f(b)$ answers black box queries about the target concept f . That is, the input to this oracle is an assignment $b \in X$ and the answer is $f(b)$.

DEFINITION 2.5. (*Exactly Learnable*)

Let C be a concept class and let H be a representation class. Then C is *exactly learnable using H* if there exists an algorithm A so that: for any concept $f \in C$, if A is given access to some of the above oracles (EQ_f, MQ_f), A outputs a hypothesis $h \in H$ satisfying $f \equiv h$. Learning must succeed against any valid choice of counterexamples by the oracles.

The next definition specifies the conditions that must be satisfied for computationally efficient learning.

DEFINITION 2.6. (*Efficiently Exactly Learnable*)

Let C be a concept class and let H be a representation class. Then C is *efficiently exactly learnable using H* if it is exactly learnable using H and the learner halts in time polynomial in n and the *size* of f .

There is a nice connection between the exact identification learning model and the probably approximately correct learning model. This is the fact that any exact

learning algorithm using equivalence queries can be transformed into a PAC learning algorithm for the same learning problem. The idea is to replace the equivalence oracle with a sampling procedure that with high probability will behave like the equivalence oracle. Since the PAC learning model allows for a certain failure probability, we can appeal to standard large deviation bounds to allocate the failure probability to each sampling simulation of the equivalence oracle. This gives us the following theorem due to Angluin [Ang88].

THEOREM 2.1. [Ang88] *Let C be a concept class and H be a representation class of Boolean functions. If C is efficiently exactly learnable using H and H contains only polynomial-time computable functions then C is efficiently PAC learnable using H .*

2.3. Specific Representation Classes for Boolean Functions

A Boolean function over n inputs is a function from $\{0,1\}^n$ to $\{0,1\}$. By default the variables will always be $\{x_1, x_2, \dots, x_n\}$ unless stated otherwise. One can also consider a *family* of Boolean functions \mathcal{F} which is a countable set of Boolean functions $\{f_n \mid n \geq 0\}$, where each f_n is a Boolean function on n inputs.

We now describe some Boolean functions that we consider in this thesis. A Boolean function is called a *monomial* or *term* if it is expressible as a conjunction of literals. A Boolean function is called *monotone* if for all $x, y \in \{0,1\}^n$, $x \leq y$ implies $f(x) \leq f(y)$. A Boolean function is called *symmetric* if its output is uniquely determined by $\sum_{i=1}^n x_i$. Some examples of symmetric functions are the parity function, the majority function, and the threshold functions. The *parity* function over inputs x_1, x_2, \dots, x_n is the Boolean function that returns $x_1 + x_2 + \dots + x_n \pmod{2}$. The *threshold* function TH_k^n on inputs x_1, x_2, \dots, x_n is defined as $TH_k^n(x) = [\sum_{i=1}^n x_i \geq k]$. The *majority* function is defined as $TH_{\lceil n/2 \rceil}^n(x)$.

In the following we will discuss some of the more specific representation classes for Boolean functions that we will need. These include Boolean circuits and formulas,

decision trees and lists, branching programs, and others.

2.3.1. Boolean Circuits and Formulae. A Boolean circuit on n input variables x_1, x_2, \dots, x_n is an acyclic digraph whose nodes are labeled with either variables, logical operators (AND, OR, and NOT), or constants (TRUE = 1 and FALSE = 0). There is a single output (sink) node whose output value is the output of the circuit. The source nodes are labeled with variables or constants. A Boolean circuit is called *monotone* if it does not contain any negation (NOT) gates. It is a standard fact that a Boolean function is monotone if and only if it is computable by a monotone Boolean circuit.

The *size* of a circuit is the number of gates or nodes in that circuit. The *depth* of a circuit is the length of the longest root-to-leaf path in that circuit.

A Boolean circuit has *bounded fan-in* if the arity of the AND and OR gates are bounded by a constant; otherwise we say that the circuit has *unbounded fan-in*. These two different types of circuits define two well-studied circuit complexity classes. The class AC^k is defined to be the class of all Boolean functions computable by families of unbounded fan-in Boolean circuits of depth $\mathcal{O}(\log^k n)$ and size $n^{\mathcal{O}(1)}$. The class NC^k is defined to be the class of all Boolean functions computable by families of bounded fan-in Boolean circuits of depth $\mathcal{O}(\log^k n)$ and size $n^{\mathcal{O}(1)}$.

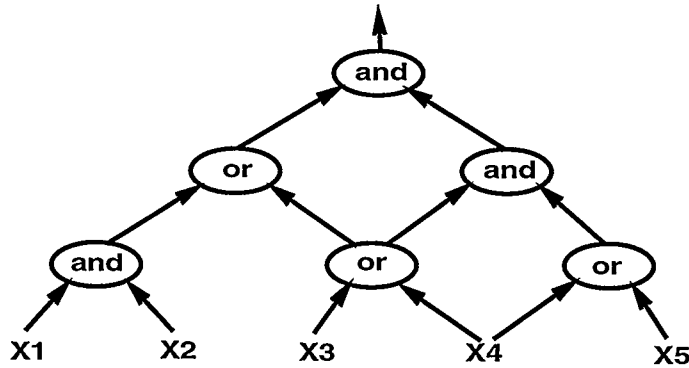


FIGURE 2.4. Example of a Boolean Circuit

We will be mostly interested in the classes AC^0 and NC^1 . It is known that $AC^0 \subseteq NC^1$ and that the inclusion is strict. Let us define AC_d^0 to be the subclass of AC^0 that contains Boolean functions computable by families of unbounded fan-in polynomial size Boolean circuits with depth exactly d , where d is some constant. The class AC_1^0 contains Boolean functions computable by unbounded fan-in AND and OR gates. An AND of literals is also called a *monomial* or a *term*. An OR of literals is also called a *clause* or a disjunction.

The class of depth two Boolean circuits with an OR gate at the top and AND gates at the bottom level, is of a particular interest to us. This class is also known as the class of *Disjunctive Normal Form* formulas. The *size* of a DNF formula is usually taken to be the number of AND gates or terms. The “dual” of DNF, where the top gate is an AND and the bottom gates are all OR, is known as the class of *Conjunctive Normal Form* (CNF) formulas. The size of a CNF formulas is taken to be the number of OR gates or clauses. In most cases we will focus our attention on DNF formulas knowing that we can, in most cases, use duality to obtain a similar result for CNF formulas. A DNF is called a k -term l -DNF if it has at most k terms (or ANDs) and each term has at most l literals. It is called *monotone* if no literals are negated.

2.3.2. Decision Programs. In this section we review the definitions of decision trees, decision lists, and branching programs. We provide a slightly more general definition of decision trees and also define the related representation class of Boolean branching programs.

Let \mathcal{F} and \mathcal{G} be two classes of Boolean functions over $\{0, 1\}^n$. An $(\mathcal{F}, \mathcal{G})$ -*decision tree* is a rooted binary tree whose internal nodes are labeled with functions from \mathcal{F} and whose leaves are labeled with functions from \mathcal{G} . Each internal node has precisely two outgoing edges, one labeled with 0 and the other labeled with 1.

A $(\mathcal{F}, \mathcal{G})$ -decision tree T computes a Boolean function from $\{0, 1\}^n$ to $\{0, 1\}$ in the following natural way. Suppose that the root node of T is labeled with $f \in \mathcal{F}$. Given an assignment $a \in \{0, 1\}^n$, the computation starts at the root node, evaluating the

function f on a , and taking the path out using the edge that is labeled with $f(a)$. The computation proceeds in this manner until a leaf node is reached, say labeled with $g \in \mathcal{G}$, whereby the computation ends with output $g(a)$.

The *size* of a $(\mathcal{F}, \mathcal{G})$ -decision tree is the sum of the sizes of functions labeling the internal nodes and the leaves. The *depth* of such a tree is the length of the longest root-to-leaf path in the tree. In the simple case of Boolean decision tree, where \mathcal{F} is the set of variables, i.e., $\mathcal{F} = \{x_1, x_2, \dots, x_n\}$ and \mathcal{G} is the set of constant functions, all-zero and all-one, then the size of the decision tree is defined instead to be the number of leaves in the tree. In the latter case, we write \mathcal{F} -decision tree where \mathcal{G} is understood to consist of the trivial constant functions.

An $(\mathcal{F}, \mathcal{G})$ -*decision list* is an $(\mathcal{F}, \mathcal{G})$ -decision tree whose internal nodes form a path. We will write an $(\mathcal{F}, \mathcal{G})$ -decision list as

$$[(f_1, g_1), (f_2, g_2), \dots, (f_m, g_m)]$$

where $f_1, f_2, \dots, f_m \in \mathcal{F}$ and $g_1, g_2, \dots, g_m \in \mathcal{G}$. We implicitly assume that the last function f_m is always the constant one (true everywhere) function.

A Boolean decision tree (respectively, a Boolean decision list) is an $(\mathcal{F}, \mathcal{G})$ -decision tree (respectively, an $(\mathcal{F}, \mathcal{G})$ -decision list) where \mathcal{F} is the set of literals and \mathcal{G} is the set of constant functions.

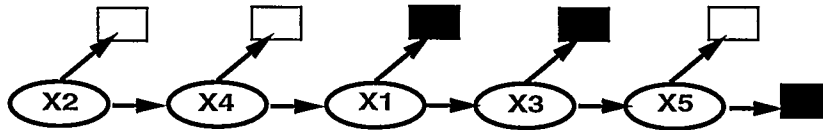


FIGURE 2.5. Example of a Decision List

A *branching program* M over $X_n = \{x_1, \dots, x_n\}$ is a directed acyclic graph whose nodes are labeled with variables from X_n and whose edges are labeled with the constants $\{0, 1\}$. It has a unique source (a node with no incoming edges) and at least

two sinks (nodes with no outgoing edges). The sinks are labeled with 0 (rejecting) and 1 (accepting), and both labels must be present. An assignment $a \in \{0,1\}^n$ to the variables induces a selection on the edges of M ; it keeps alive all edges that are consistent with the assignment a . Then the branching program is said to accept a if there is a directed path from the source to an accepting sink.

The *size* of a branching program is the number of nodes in the branching program. A branching program is called *leveled* if there is an ordered partition $\Pi = (L_1, L_2, \dots)$ of the nodes of the branching program such that all of the edges connect nodes of one level to the next one in the partition. The *width* of a leveled branching program is the maximum number of nodes in any level in the ordered partition.

In this thesis we will be interested in branching programs with bounded or $\mathcal{O}(1)$ width. More specifically, we will focus our attention on branching programs with widths 2, 3 or 4. Note that width one branching programs can be identified with decision lists where the accepting and rejecting nodes are collapsed into two sinks. We will not consider width 5 branching programs since the Boolean functions they compute are known to be equivalent to the class NC^1 and are known to be not learnable under some cryptographic assumptions.

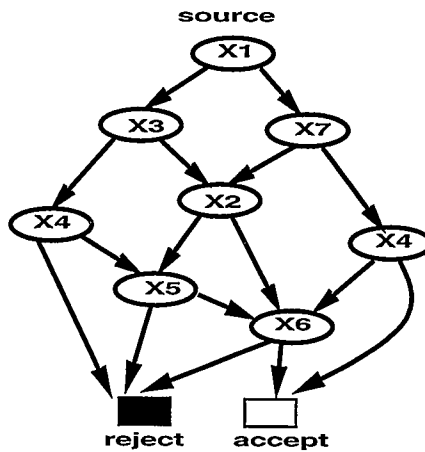


FIGURE 2.6. Example of a Branching Program

2.4. Inequalities and Probabilities

The purpose of this section is to describe some useful inequalities and facts from probability theory. We start with two bounds on the sum of binomial coefficients.

FACT 2.2. *For any integers n, k , $\sum_{i=0}^k \binom{n}{i} \leq n^{k+1}$.*

Proof Since $\binom{n}{i} \leq n^i$, for all $k \leq n$, we get that the sum is bounded by $(n^{k+1} - 1)/(n - 1)$. \square

FACT 2.3. ([R92], Theorem 1.2.8, page 25)

For any integers n, k and $\alpha \in (0, 1/2)$, we have

$$\sum_{i=0}^{\alpha n} \binom{n}{i} \leq 2^{nH(\alpha)}$$

where $H(x) = -x \log x - (1 - x) \log(1 - x)$, for $x \in [0, 1]$.

The following inequality is well-known from mathematical analysis.

FACT 2.4. (Cauchy-Schwarz Inequality)

Let a_1, \dots, a_n and b_1, \dots, b_n be two sequences of non-negative numbers. Then

$$\left(\sum_{i=1}^n a_i b_i \right)^2 \leq \left(\sum_{i=1}^n a_i^2 \right) \left(\sum_{i=1}^n b_i^2 \right).$$

Moreover, equality is attained precisely if, for some constant λ , $a_i = \lambda b_i$, for all $i \in [n]$.

We now describe some inequalities from probability theory that we need, namely Markov's inequality, Chernoff bounds, and also Hoeffding bounds on the sum of independent random variables.

FACT 2.5. (Markov's Inequality)

For any non-negative random variable X and for any positive real number t ,

$$\Pr[X \geq t] \leq \mathbf{E}[X]/t.$$

We state some standard large deviation bounds from probability theory, known as Chernoff and Hoeffding bounds. We give several forms of these bounds.

FACT 2.6. (Chernoff bounds [R90])

Let X_1, X_2, \dots, X_n be independent Bernoulli trials with $\Pr[X_i = 1] = p_i, p_i \in (0, 1)$.

Let $X = \sum_{i=1}^n X_i$ and $\mu = \sum_{i=1}^n p_i$. Then for $\delta > 0$

$$\Pr[X > (1 + \delta)\mu] < \left[\frac{\exp(\delta)}{(1 + \delta)^{(1+\delta)}} \right]^\mu \doteq F^+(\mu, \delta).$$

Under the same hypothesis as above, for $\delta \in (0, 1]$,

$$\Pr[X < (1 - \delta)\mu] < \exp(-\mu\delta^2/2) \doteq F^-(\mu, \delta).$$

The second version of the large deviation bounds is known as *Hoeffding* bounds.

FACT 2.7. (Hoeffding bounds [M94])

Let X_1, \dots, X_m be independent identically distributed random variables with $\mathbf{E}[X_i] = p$, $|X_i| \leq B$, and let $S_m = X_1 + \dots + X_m$. If $m = m(\epsilon, \delta, B) \geq \frac{2B^2}{\epsilon^2} \ln \frac{2}{\delta}$, then

$$\Pr \left[\left| \frac{S_m}{m} - p \right| > \epsilon \right] \leq \delta.$$

Most of the probabilistic arguments or machinery that we use can be found in Alon and Spencer's book [AS92], e.g., averaging arguments, linearity of expectations, and also large deviation bounds, etc. As for notation, we use $\Pr_{x \in D}[P(x)]$ or $\Pr_D[P(x)]$ to indicate the probability that event $P(x)$ happens when x is randomly chosen according to distribution D . We usually omit specifying the underlying distribution whenever it is clear from context or if it is the uniform distribution U . The same convention applies for expectations and variances.

Harmonic Analysis of Boolean Functions

*All analysts spend half their time hunting through the literature
for inequalities which they want to use but cannot prove.*

– Harald Bohr

The goal of a PAC learning algorithm is to infer a target function from a rather small set of randomly chosen examples. The idea behind Fourier or harmonic analysis is to recover a function from its frequency patterns or *Fourier coefficients*. Fortunately, even a rather small randomly chosen set of examples can give a good approximation to most of the frequency patterns of the function. One can then recover the function approximately from these approximate frequency patterns.

The use of Fourier analysis in Boolean complexity theory was introduced by Chor and Geréb-Graus [CG88] and by Kahn, Kalai, and Linial [KKL88] in their work that studies the influence of variables on Boolean functions. The important connection to learning theory was discovered only later by Linial, Mansour and Nisan [LMN93]. In the latter paper the authors proved that the class of AC^0 functions, i.e., Boolean functions computable by families of unbounded fan-in constant depth and polynomial size Boolean circuits, is learnable in the PAC model under the uniform distribution in time $n^{\text{poly}(\log n)}$. Let us call their algorithm the *LMN algorithm*.

Further improvements to the LMN algorithm were given by Furst, Jackson, and Smith [FJS91] and by Aiello and Mihail [AM91]. Furst *et al.* proved that the class of AC^0 functions is learnable in the PAC model under constant-bounded product distri-

butions in time $n^{\text{poly}(\log n)}$. They obtained this result through a nontrivial extension of the work of Linial *et al.* [LMN93]. Aiello and Mihail [AM91] proved that probabilistic decision lists are learnable in the PAC model under the uniform distribution in polynomial time.

The main reason for the successes of these results is the ability to isolate a *small region* in the Boolean n -cube where a class of functions has most of its significant Fourier coefficients. In other words this isolated region stores a lot of information about the function in the form of frequency patterns. For example, in [LMN93] it was shown that most of the power spectrum of AC^0 functions is located on the vectors of Hamming weight $\text{poly}(\log n)$. The knowledge of *where* these “heavy” frequencies lie in $\{0, 1\}^n$ is crucial to the LMN learning algorithm. Using this information, Linial, Mansour, and Nisan devised their simple PAC learning algorithm for AC^0 functions that runs in time $n^{\text{poly}(\log n)}$.

In this chapter we will show that the class of monotone Boolean functions has a similar property to that of the class of AC^0 functions. We prove that most of the power spectrum of monotone Boolean functions is located on the vectors of Hamming weight $\mathcal{O}(\sqrt{n})$. Our proof relies on a simple connection between the Fourier spectrum and a well known measure called *average sensitivity*. The notion of average sensitivity was previously studied by Kahn, Kalai, and Linial [KKL88] and has been considered in several other works. In the proof we will actually show that the average sensitivity of monotone Boolean functions (regardless of their circuit complexities) is at most \sqrt{n} . Then we will use this to show that the power spectrum of monotone Boolean functions is concentrated in the vectors of Hamming weight at most $\mathcal{O}(\sqrt{n})$.

The results that we present in this chapter are slightly more general than the ones considered in [KKL88, LMN93]. We will consider the Fourier spectrum and average sensitivity under a more general class of probability distributions than the uniform distribution. We obtained this by a natural generalization of the analysis and proofs given in [KKL88].

For more information on Fourier analysis of Boolean functions, we refer the reader to the survey paper [M94].

3.1. Basic Theory of Fourier Transform

We start by associating with a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ a corresponding real-valued function $F : \{0, 1\}^n \rightarrow \{-1, +1\} \subset \mathbb{R}$ where f and F are related by the following simple equations:

$$F = 2f - 1, \quad f = \frac{1 + F}{2}.$$

In most cases we will call both f and F *Boolean* functions and sometimes abusively use f when we mean F when the context is clear.

The set F_n of all real-valued functions over $\{0, 1\}^n$ forms a vector space of dimension 2^n over \mathbb{R} . We will associate with this vector space an inner product that is induced by a probability distribution on $\{0, 1\}^n$. We will mainly focus on a specific class of probability distributions called *product* distributions.

DEFINITION 3.1. (*Product distribution*)

A probability distribution D over $\{0, 1\}^n$ is called a *product distribution* with parameters $\mu = (\mu_i)_{i=1}^n$, where each $\mu_i \in (0, 1)$, if for all $a \in \{0, 1\}^n$

$$D(a) = \prod_{i:a_i=1} \mu_i \prod_{i:a_i=0} (1 - \mu_i).$$

A product distribution D is called *c*-bounded if $\mu_i \in [c, 1 - c]$, for all $i \in [n]$. The distribution D is called *constant-bounded* if there is a constant $c \in (0, 1)$ so that D is *c*-bounded.

Given a product distribution $D = (\mu_1, \dots, \mu_n)$ over $\{0, 1\}^n$ and $i \in [n]$, we denote D_i to be the product distribution over $\{0, 1\}^{n-1}$ obtained by removing the *i*-th component of D , i.e.,

$$D_i = (\mu_1, \dots, \mu_{i-1}, \mu_{i+1}, \dots, \mu_n).$$

Note that the *uniform* distribution is a product distribution with $\mu_i = 0.5$, for all $i \in [n]$.

DEFINITION 3.2. (*Inner product*)

Given a probability distribution D on $\{0, 1\}^n$, the *inner product* $\langle \cdot, \cdot \rangle_D$ on F_n with respect to D is defined as

$$\langle f, g \rangle_D \doteq \sum_x D(x) f(x) g(x) = \mathbf{E}_D[fg].$$

A collection B of functions is called *orthonormal* if for any $f, g \in B$ we have $\langle f, g \rangle = [f = g]$. A collection of functions $B = \{g_1, \dots, g_m\}$ is a *basis* for F_n if it is a spanning set, i.e., for any $f \in F_n$ there exist constants $a_1, \dots, a_m \in \mathbb{R}$ so that $f(x) = \sum_{i=1}^m a_i g_i(x)$, and if it is a linearly independent set, i.e., whenever $\sum_{i=1}^m a_i g_i(x) = 0$, we have $a_i = 0$, for all $i \in [m]$.

FACT 3.1. Suppose that D is a product distribution over $\{0, 1\}^n$ with parameters $\mu = (\mu_i)_{i=1}^n$. Define $\sigma_i = \sqrt{\mu_i(1 - \mu_i)}$. The set of functions $\{\phi_a(x) : a \in \{0, 1\}^n\}$, where

$$\phi_a(x) = \prod_{a_i=1} \frac{\mu_i - x_i}{\sigma_i},$$

is an orthonormal basis.

These functions also satisfy an additional decomposability property, i.e., for all x, y, a, b with $|x| = |a|$ and $|y| = |b|$ we have

$$\phi_{a \circ b}(x \circ y) = \phi_a(x) \phi_b(y),$$

which is easy to verify.

The *Fourier coefficient* of f at $a \in \{0, 1\}^n$ under distribution D is defined as

$$\tilde{f}_D(a) = \langle f, \phi_a \rangle_D = \mathbf{E}_D[f \phi_a].$$

Note that $\tilde{f}(0_n) = \mathbf{E}_D[f]$. When D is clear from context, we will write $\tilde{f}(a)$ instead of $\tilde{f}_D(a)$. By orthonormality of the functions ϕ_a , each real-valued function f over

$\{0,1\}^n$ has a unique representation in terms of its Fourier coefficients. We state this in the following fact.

FACT 3.2. *For any function $f : \{0,1\}^n \rightarrow \mathbb{R}$ we have*

$$f(x) = \sum_a \tilde{f}(a) \phi_a(x).$$

Moreover this representation is unique.

We will need the following correlation lemma to derive some future facts.

LEMMA 3.3. (Cross-Correlation identity)

For any functions $f, g : \{0,1\}^n \rightarrow \mathbb{R}$, for any $y \in \{0,1\}^n$, and for any product distribution $D = (\mu_i)_{i=1}^n$ over $\{0,1\}^n$ we have

$$\mathbf{E}_D[f(x)g(x \oplus y)] = \sum_{a,b} \tilde{f}(a) \tilde{g}(b) \mathbf{E}_D[\phi_a(x) \phi_b(x \oplus y)].$$

Proof Note that $g(x \oplus y) = \sum_b \tilde{g}(b) \phi_b(x \oplus y)$. We simply expand the similar expression for f and then use linearity of expectation. \square

An important corollary to the lemma is Parseval's identity.

COROLLARY 3.4. (Parseval's identity)

For any real-valued function $f : \{0,1\}^n \rightarrow \mathbb{R}$ we have $\mathbf{E}_D[f^2] = \sum_a \tilde{f}^2(a)$. Moreover, if f is Boolean then this expression equals 1.

Proof Take $g = f$ and $y = 0_n$ in Lemma 3.3 and then apply orthonormality of ϕ_a 's to get the claim. \square

The expression $\sum_a \tilde{f}(a)^2$ is also called the *Fourier power spectrum* of f under product distribution D .

For the *uniform* distribution U there is an already established notation in the literature. For the uniform distribution U , the basis function at a is denoted $\chi_a(x) = (-1)^{a \cdot x}$ and the Fourier coefficient of f at a is denoted $\hat{f}(a)$, i.e., $\tilde{f}_U(a)$.

3.2. Influence and Average Sensitivity

The paper of Kahn, Kalai, and Linial [KKL88] gave a relationship between Fourier transform and two complexity measures called the *influence* and the *average sensitivity*. Our main results will be based on their original ideas, although we will provide some minor simplifications, extensions, and applications to machine learning. The notion of influence has been used earlier in connection with learning DNF formulas where each variable can only appear a constant number of times (see Hancock and Mansour [HM91]).

We now define the notions of influence and average sensitivity for Boolean functions.

DEFINITION 3.3. (*Influence*)

Let $f : \{0,1\}^n \rightarrow \{-1, +1\}$ be a Boolean function over the variables $\{x_1, \dots, x_n\}$ and let D be a product distribution over $\{0,1\}^n$. Then the *influence* of x_i on f over D is the probability that $f(x)$ differs from $f(x \oplus e_i)$ when x is chosen according to D , or,

$$I_{D,i}(f) = \Pr_{x \in_D \{0,1\}^n} [f(x) \neq f(x \oplus e_i)].$$

Remark. On occasion we use the restriction notation $f_0 = f|_{x_i \leftarrow 0}$ and $f_1 = f|_{x_i \leftarrow 1}$, when the variable x_i to be restricted is clear from the context.

FACT 3.5. For any Boolean function $f : \{0,1\}^n \rightarrow \{-1, +1\}$ we have $I_{D,i}(f) = \Pr_D[f_0(y) \neq f_1(y)] = \frac{1}{4} \mathbf{E}_D[(f_1 - f_0)^2]$. Moreover if f is monotone then $I_{D,i}(f) = \frac{1}{2} \mathbf{E}_D[f_1 - f_0]$.

DEFINITION 3.4. (*Influence Norm*)

Let D be a product distribution over $\{0,1\}^n$ with parameters $\mu = (\mu_1, \dots, \mu_n)$ and let $f : \{0,1\}^n \rightarrow \{-1, +1\}$ be a Boolean function. Then the *influence norm* of f with respect to D is defined as

$$I_D(f) = \sqrt{\sum_{i=1}^n (2\sigma_i I_{D,i}(f))^2}.$$

DEFINITION 3.5. (*Average Sensitivity*)

Let $f : \{0,1\}^n \rightarrow \{-1, +1\}$ be a Boolean function over the variables $\{x_1, \dots, x_n\}$ and let D be a product distribution over $\{0,1\}^n$. The average sensitivity $s_a(f)$ of f at a point $a \in \{0,1\}^n$ is the number of immediate neighbors of a whose values under f differ from a . More formally,

$$s_a(f) = |\{i \in [n] : f(a) \neq f(a \oplus e_i)\}| = \sum_{i=1}^n [f(a) \neq f(a \oplus e_i)].$$

The *average sensitivity* of f with respect to D is

$$s_D(f) = \mathbf{E}_{x \in D}[s_x(f)].$$

It is easy to show that average sensitivity is equal to the sum of influences.

FACT 3.6. $s_D(f) = \sum_{i=1}^n I_{D,i}(f)$.

Examples: Consider the *parity* function on n inputs, i.e., $\chi_{1_n}(x)$. Under the uniform distribution, the influence of each variable is one, and hence its average sensitivity equals n . The influence of any variable on a *constant* function is zero and hence the average sensitivity is also zero.

3.3. Relating Fourier Spectrum and Average Sensitivity

The authors of [KKL88] proved that the average sensitivity of a Boolean function is a “weighted” sum of the Fourier spectrum under the uniform distribution, where the sum is weighted according to the Hamming weight. We state for completeness their claim along with a simple proof in the following fact.

FACT 3.7. [KKL88] *For any Boolean function $f : \{0,1\}^n \rightarrow \{-1, +1\}$*

$$s_U(f) = \sum_a |a| \hat{f}(a)^2.$$

Proof We use Lemma 3.3 (cross correlation identity) for the special case of the uniform distribution with $g = f$ and $y = e_i$ (the unit vector at i):

$$\mathbf{E}_U[f(x)f(x \oplus e_i)] = \sum_{a,b} \hat{f}(a)\hat{f}(b)\mathbf{E}_U[\chi_a(x)\chi_b(x \oplus e_i)].$$

But $\chi_b(x \oplus e_i) = \chi_b(x)\chi_b(e_i)$ and $\chi_b(e_i) = (-1)^{b_i}$. Hence the expectation on the right-hand side simplifies to $\mathbf{E}[\chi_a(x)\chi_b(x)](-1)^{b_i} = [a = b](-1)^{b_i}$. Thus

$$\mathbf{E}_U[f(x)f(x \oplus e_i)] = \sum_b \hat{f}(b)^2(-1)^{b_i} = 1 - 2 \sum_{b:b_i=1} \hat{f}(b)^2.$$

Also we note that $I_{U,i}(f) = \frac{1}{2}(1 - \mathbf{E}_U[f(x)f(x \oplus e_i)])$ which yields $I_{U,i}(f) = \sum_{b:b_i=1} \hat{f}(b)^2$. We complete the proof by summing up this last expression for all $i \in [n]$ and then use Fact 3.6. \square

Our goal is to show that the above fact holds also under product distributions. We need the following identity in order to extend their result.

LEMMA 3.8. *For any Boolean function $f : \{0,1\}^n \rightarrow \{-1, +1\}$, any product distribution D over $\{0,1\}^n$, and any $i \in [n]$,*

$$\mathbf{E}_D[f(x)f(x \oplus e_i)] = 1 - \frac{1}{2\sigma_i^2} \sum_{a:a_i=1} \tilde{f}(a)^2.$$

Proof Let

$$\Delta = \mathbf{E}_D[f(x)f(x \oplus e_i)] = \sum_{a,b} \tilde{f}(a)\tilde{f}(b)\mathbf{E}_D[\phi_a(x)\phi_b(x \oplus e_i)],$$

using Lemma 3.3. Set $a = \bar{a} \circ a_i$ and $b = \bar{b} \circ b_i$, where $\bar{a}, \bar{b} \in \{0,1\}^{n-1}$ and $a_i, b_i \in \{0,1\}$ (assume without loss of generality that $i = n$). Also let $x = y \circ x_i$ and recall that D_i is the distribution obtained by removing the i -th component from distribution D .

We see that the expression $\mathbf{E}[\phi_a(x)\phi_b(x \oplus e_i)]$ equals to

$$(1 - \mu_i)\mathbf{E}_{D_i}[\phi_{\bar{a}}(y)\phi_{\bar{b}}(y)] \left(\frac{\mu_i}{\sigma_i}\right)^{a_i} \left(\frac{\mu_i - 1}{\sigma_i}\right)^{b_i} + \mu_i\mathbf{E}_{D_i}[\phi_{\bar{a}}(y)\phi_{\bar{b}}(y)] \left(\frac{\mu_i - 1}{\sigma_i}\right)^{a_i} \left(\frac{\mu_i}{\sigma_i}\right)^{b_i}.$$

Define $\Upsilon = \mathbf{E}_D[\phi_a(x)\phi_b(x \oplus e_i)]$. We consider the cases based on the values of $a_i, b_i \in$

$\{0, 1\}$.

$$\Upsilon = \begin{cases} \mathbf{E}_{D_i}[\phi_{\bar{a}}(y)\phi_{\bar{b}}(y)] & \text{if } a_i = b_i = 0 \\ \frac{\mu_i^2 - (1 - \mu_i)^2}{\sigma_i} \mathbf{E}_{D_i}[\phi_{\bar{a}}(y)\phi_{\bar{b}}(y)] & \text{if } a_i = 0, b_i = 1 \\ 0 & \text{if } a_i = 1, b_i = 0 \\ -\mathbf{E}_{D_i}[\phi_{\bar{a}}(y)\phi_{\bar{b}}(y)] & \text{if } a_i = b_i = 1 \end{cases}$$

Hence by orthonormality of the basis functions ϕ ,

$$\begin{aligned} \Delta &= \sum_{\bar{a}, \bar{b} \in \{0,1\}^{n-1}} \left[\tilde{f}(0\bar{a})\tilde{f}(0\bar{b}) - \tilde{f}(1\bar{a})\tilde{f}(1\bar{b}) + \frac{\mu_i^2 - (1 - \mu_i)^2}{\sigma_i} \tilde{f}(0\bar{a})\tilde{f}(1\bar{b}) \right] \\ &\quad \mathbf{E}_{D_i}[\phi_{\bar{a}}(y)\phi_{\bar{b}}(y)] \\ &= \sum_{\bar{a} \in \{0,1\}^{n-1}} (\tilde{f}(0\bar{a})^2 - \tilde{f}(1\bar{a})^2) + \sum_{\bar{a} \in \{0,1\}^{n-1}} \tilde{f}(0\bar{a})\tilde{f}(1\bar{a}) \frac{\mu_i^2 - (1 - \mu_i)^2}{\sigma_i}. \end{aligned}$$

Observing that

$$\tilde{f}(0\bar{a}) = (1 - \mu_i)\tilde{f}_0(\bar{a}) + \mu_i\tilde{f}_1(\bar{a}), \quad \tilde{f}(1\bar{a}) = \sigma_i(\tilde{f}_0(\bar{a}) - \tilde{f}_1(\bar{a})),$$

we obtain

$$\begin{aligned} &\sum_{\bar{a} \in \{0,1\}^{n-1}} \tilde{f}(0\bar{a})\tilde{f}(1\bar{a}) \frac{\mu_i^2 - (1 - \mu_i)^2}{\sigma_i} \\ &= \sum_{\bar{a} \in \{0,1\}^{n-1}} [(1 - \mu_i)\tilde{f}_0(\bar{a}) + \mu_i\tilde{f}_1(\bar{a})][\sigma_i(\tilde{f}_0(\bar{a}) - \tilde{f}_1(\bar{a}))] \frac{\mu_i^2 - (1 - \mu_i)^2}{\sigma_i} \\ &= [\mu_i^2 - (1 - \mu_i)^2] \sum_{\bar{a} \in \{0,1\}^{n-1}} [(1 - \mu_i)\tilde{f}_0(\bar{a})^2 - \mu_i\tilde{f}_1(\bar{a})^2 + (2\mu_i - 1)\tilde{f}_0(\bar{a})\tilde{f}_1(\bar{a})] \\ &= (2\mu_i - 1)[(1 - 2\mu_i) + (2\mu_i - 1) \sum_{\bar{a} \in \{0,1\}^{n-1}} \tilde{f}_0(\bar{a})\tilde{f}_1(\bar{a})]. \end{aligned}$$

But note that $\sum_{\bar{a} \in \{0,1\}^{n-1}} \tilde{f}_0(\bar{a})\tilde{f}_1(\bar{a}) = \mathbf{E}_{D_i}[f_0(y)f_1(y)] = \mathbf{E}_D[f(x)f(x \oplus e_i)]$. Thus

$$\Delta = 1 - 2 \sum_{\bar{a} \in \{0,1\}^{n-1}} \tilde{f}(1\bar{a})^2 - (2\mu_i - 1)^2 + (2\mu_i - 1)^2 \Delta,$$

which finishes the claim. \square

DEFINITION 3.6. (*Generalized Hamming weight*)

Let $D = (\mu_i)_{i=1}^n$ be a product distribution over $\{0,1\}^n$. The *weight* of $a \in \{0,1\}^n$

under D , denoted $\|a\|_D$, is defined as

$$\|a\|_D = \prod_{i:a_i=1} \log_2 \frac{1}{\sigma_i}.$$

FACT 3.9. *For any product distribution D , $\|a\|_D \geq |a|$, and equality holds if and only if D is the uniform distribution.*

We are now ready to prove the generalization of 3.7 for arbitrary product distributions.

THEOREM 3.10. *For any Boolean function $f : \{0,1\}^n \rightarrow \{-1,+1\}$ and for any product distribution D over $\{0,1\}^n$ we have*

$$s_D(f) = \sum_a \|a\|_D \tilde{f}(a)^2.$$

Proof First note that $I_{D,i}(f) = \frac{1}{2}(1 - \mathbf{E}_D[f(x)f(x \oplus e_i)])$. Thus, Lemma 3.8 gives $4\sigma_i^2 I_{D,i}(f) = \sum_{a:a_i=1} \tilde{f}(a)^2$. We combine this with $s_D(f) = \sum_{i=1}^n I_{D,i}(f)$ to complete the claim. \square

Using the above theorem we may derive a Fourier spectrum bound on any Boolean function over any product distributions. We will supply two bounds, one in terms of the average sensitivity and one in terms of the influence norm.

THEOREM 3.11. *Let $f : \{0,1\}^n \rightarrow \{-1,+1\}$ be a Boolean function and let D be a product distribution over $\{0,1\}^n$. Given an $\epsilon > 0$, let $A(\epsilon) \subseteq \{0,1\}^n$ be defined as $A(\epsilon) = \{a : \|a\| \geq \frac{6}{5}s_D(f)/\epsilon\}$. Then we have $\sum_{a \in A(\epsilon)} \tilde{f}(a)^2 \leq \epsilon$.*

Proof We start with the identity

$$\sum_a \|a\| \tilde{f}(a)^2 = \sum_i 4\sigma_i [\sigma_i \log \sigma_i^{-1}] I_{D,i}(f),$$

which can be derived from the identity $4\sigma_i^2 I_{D,i}(f) = \sum_{a:a_i=1} \tilde{f}(a)^2$ in the proof of Theorem 3.10. Now we use the fact $\sigma \leq 1/2$ and $x \log \frac{1}{x} \leq 3/5$ (not the best), for $x \in [0,1]$, to get $\sum_a \|a\| \tilde{f}(a)^2 \leq \frac{6}{5}s_D(f)$, since $s_D(f) = \sum_i I_{D,i}(f)$. Now we can partition the sum according to whether $\|a\| \geq \frac{6}{5}s_D(f)/\epsilon$ or not. \square

An alternative upper bound on the Fourier spectrum can be given using the influence norm. We consider the next theorem one of the main technical contributions in this chapter.

THEOREM 3.12. *If $f : \{0, 1\}^n \rightarrow \{-1, +1\}$ is a Boolean function and D is a product distribution over $\{0, 1\}^n$ then*

$$\sum_{\|a\| \geq k} \tilde{f}^2(a) \leq \frac{2}{k} I_D(f) \sqrt{\sum_{i=1}^n \left(\sigma_i \log \frac{1}{\sigma_i} \right)^2} \leq c I_D(f) \frac{\sqrt{n}}{k},$$

for some absolute constant c (for example, the inequality is correct if $c = 1.062$).

Proof Using the Cauchy-Schwarz inequality, Fact 2.4, with $a_i = 2\sigma_i I_{D,i}(f)$ and $b_i = 2\sigma_i \log \sigma_i^{-1}$, we get

$$\begin{aligned} I_D(f)^2 &= \sum_{i=1}^n 4\sigma_i^2 I_{D,i}(f)^2 \\ &\geq \frac{\left(\sum_{i=1}^n 4\sigma_i^2 I_{D,i}(f) \log \sigma_i^{-1} \right)^2}{4 \sum_{i=1}^n \left(\sigma_i \log \sigma_i^{-1} \right)^2}, \quad \text{by Cauchy-Schwarz} \\ &= \frac{1}{4 \sum_{i=1}^n \left(\sigma_i \log \sigma_i^{-1} \right)^2} \left(\sum_a \|a\| \tilde{f}^2(a) \right)^2 \\ &\geq \frac{1}{4 \sum_{i=1}^n \left(\sigma_i \log \sigma_i^{-1} \right)^2} \left(k \sum_{\|a\| \geq k} \tilde{f}^2(a) \right)^2. \end{aligned}$$

This proves the first inequality. The second inequality can be seen using simple calculus since $(x \log x^{-1})^2 \leq e^{-2} \log^2 e \leq 0.2817$ for all $x \in [0, 1/2]$. \square

The following lemma will supply us with the crucial link that connects the Fourier transform for monotone Boolean functions to influences – and hence to average sensitivity.

LEMMA 3.13. *If $f : \{0, 1\}^n \rightarrow \{-1, +1\}$ is a monotone Boolean function and D is a product distribution over $\{0, 1\}^n$, then for all $i \in [n]$, $\tilde{f}(e_i) = -2\sigma_i I_{D,i}(f)$.*

Proof Let D_i be the induced distribution over all the variables except the i -th one, i.e., x_i . Then

$$\tilde{f}(e_i) = \mathbb{E}_D[f\phi_{e_i}] = \mathbb{E}_{D_i} \left[(1 - \mu_i) \frac{\mu_i}{\sigma_i} f_0 + \mu_i \frac{\mu_i - 1}{\sigma_i} f_1 \right]$$

which simplifies to $\sigma_i \mathbb{E}_{D_i}[f_0 - f_1]$. Now recall that for monotone Boolean functions, $2I_{D,i}(f) = \mathbb{E}_{D_i}[f_1 - f_0]$. \square

Using Lemma 3.13 we can derive an upper bound on the average sensitivity of any monotone Boolean function with respect to product distributions.

THEOREM 3.14. *For any monotone Boolean function $f : \{0, 1\}^n \rightarrow \{-1, +1\}$ and any product distribution D over $\{0, 1\}^n$, we have*

$$s_D(f) \leq \sqrt{n} \sqrt{\sum_{i=1}^n \left(\frac{\tilde{f}(e_i)}{2\sigma_i} \right)^2}.$$

Moreover, for the uniform distribution, this bound becomes \sqrt{n} , because $\sigma_i = 1/2$, for all $i \in [n]$.

Proof Using Fact 3.6 and the Cauchy-Schwarz inequality we get that

$$s_D(f)^2 = \left(\sum_{i=1}^n I_{D,i}(f) \right)^2 \leq \left(\sum_{i=1}^n I_{D,i}(f)^2 \right) n = \left(\sum_{i=1}^n \frac{\tilde{f}(e_i)^2}{4\sigma_i^2} \right) n.$$

\square

Summary. Our main findings in this chapter are the following relationships, stated for the uniform distribution, that connect the Fourier spectrum, average sensitivity, and influences, for arbitrary Boolean functions:

$$\sum_{|a| \geq \frac{1}{\epsilon} s(f)} \hat{f}(a)^2 \leq \epsilon, \quad \sum_{|a| \geq k} \hat{f}(a)^2 \leq \frac{\sqrt{n}}{k} \sqrt{\sum_{i=1}^n I_i(f)^2}.$$

These equations have natural generalizations to the case of product distributions. For monotone Boolean functions, we can further use the fact that $I_i(f) = -\hat{f}(e_i)$, for all

$i \in [n]$, to obtain a simplification of the second equation:

$$\sum_{|a| \geq k} \hat{f}(a)^2 \leq \frac{\sqrt{n}}{k},$$

which can also be derived from the first equation by noting that $s(f) \leq \sqrt{n}$, for all monotone Boolean functions.

Learning Monotone Functions in the PAC Model

In this chapter we will describe some learning algorithms for classes of monotone Boolean functions. We review some well known facts about using the Fourier spectrum for PAC learning. In particular we describe the learning algorithm of Linial, Mansour, and Nisan [LMN93] and its randomized improvement due to Blum *et al.* [BFJ⁺94]. We will call this algorithm the LMN learning algorithm.

Then we will use facts that we developed from the previous chapter to prove that the class of monotone Boolean functions is PAC learnable under product distributions with a subexponential time and sample complexity with respect to the number of inputs and the inverse of the accuracy parameter. The dependency of the learning complexity on the confidence parameter is not as high, i.e., it is only logarithmic in $1/\delta$. Some other contributions of this result are as follows. First, the statement holds regardless of the circuit complexity of the target monotone Boolean function. That is, the learning complexity is independent of the circuit size measure of the target monotone Boolean function. Second, the result handles the general class of product distributions, whereas most other works require the product distribution to be constant-bounded (see [FJS91, HM91, J94]).

We present the basic application of the LMN learning algorithm combined with the Fourier results from the previous chapter. Next we provide a careful analysis of how to learn monotone Boolean functions under an arbitrary product distribution. This yields a subexponential time PAC learning algorithm for the class of all monotone

Boolean functions. Then we will consider some lower bounds questions in order to determine how tight these results are in terms of the tolerable error rates, time complexity, and sample complexity.

In the second part of this chapter we turn our attention to efficient (or polynomial-time) PAC learning of monotone Boolean functions. We improve a weak PAC learning algorithm for all monotone Boolean functions due to Kearns and Valiant [KV89] and a PAC learning algorithm for $\mathcal{O}(\log n)$ -term monotone DNF due to Sakai and Maruoka [SM94]. The first result relies on a lower bound given by Kahn, Kalai, and Linial [KKL88] while the second result uses a top-down decision tree learning algorithm.

4.1. The Linial-Mansour-Nisan Learning Algorithm

A key link between the learnability of a Boolean function and its Fourier power spectrum under the uniform distribution is given by the following fact due to Linial, Mansour, and Nisan [LMN93].

FACT 4.1. [LMN93] *For any Boolean function $f : \{0,1\}^n \rightarrow \{-1,+1\}$, for any real-valued function $g : \{0,1\}^n \rightarrow \mathbb{R}$, and for any product distribution D over $\{0,1\}^n$, we have*

$$\Pr_{x \in_D \{0,1\}^n} [f(x) \neq \text{sgn}(g(x))] \leq E_D[(f - g)^2] = \sum_a (\tilde{f}(a) - \tilde{g}(a))^2,$$

where $\text{sgn}(g)(x) = (-1)^{[g(x) < 0]}$ is the sign function of g .

Proof The first inequality is true because $[f(x) \neq \text{sgn}(g(x))] \leq |f(x) - g(x)|$. The second inequality is true by Parseval's identity. \square

In fact one can do slightly better with randomization as shown in [BFJ⁺94].

FACT 4.2. [BFJ⁺94] *For any Boolean function $f : \{0,1\}^n \rightarrow \{-1,+1\}$, for any real-valued function $g : \{0,1\}^n \rightarrow \mathbb{R}$, for any product distribution D over $\{0,1\}^n$, and*

for a hypothesis function $h : \{0, 1\}^n \rightarrow \{-1, +1\}$ defined as follows:

$$h(x) = \begin{cases} -1 & \text{with probability } p(x) \\ +1 & \text{with probability } 1 - p(x) \end{cases}$$

where $p(x) = \frac{(1-g(x))^2}{2(1+g^2(x))}$, we have

$$\Pr_{x,r}[f(x) \neq h(x)] \leq \frac{1}{2} E_{D,r}[(f - g)^2],$$

where r denotes the randomization required for $h(x)$.

If a Boolean function has most of its power spectrum concentrated on the coefficients of small Hamming weight, say at most k , then there is a simple statistical (sampling-based) algorithm that can approximate that Boolean function rather well under the uniform distribution. We will call this algorithm the LMN algorithm or the *low-degree* or k -degree Fourier algorithm when we need to specify explicitly the region $\{a \in \{0, 1\}^n : |a| \leq k\}$ from which the sampling is performed.

LMN algorithm

input: An integer k , a sample $(x_1, f(x_1)), \dots, (x_m, f(x_m))$.

(1) Set $A = \{a \in \{0, 1\}^n : |a| \leq k\}$.

(2) **For** each $a \in \{0, 1\}^n$ with $|a| \leq k$ **do**

$$c_a = \frac{1}{m} \sum_{i=1}^m f(x_i) \chi_a(x_i).$$

(3) Output $h(x) = \sum_{a \in A} c_a \chi_a(x)$.

FIGURE 4.1. The Linial-Mansour-Nisan algorithm.

FACT 4.3. [LMN93] *Let $f : \{0, 1\}^n \rightarrow \{-1, +1\}$ be a Boolean function on n inputs and suppose that $A = \{a \in \{0, 1\}^n : |a| \leq k\}$ satisfies $\sum_{a \notin A} \tilde{f}(a)^2 \leq \epsilon/2$. Then there*

is a PAC learning algorithm for f under the uniform distribution with error ϵ and confidence δ running in time $\mathcal{O}(\frac{n^{k+1}}{\epsilon} \ln \frac{n^{k+1}}{\delta})$.

Proof The algorithm simply approximates each $\tilde{f}(a)$, for $a \in A$, with c_a that is within $\sqrt{\epsilon/(2|A|)}$, i.e.,

$$|\tilde{f}(a) - c_a| \leq \sqrt{\epsilon/(2|A|)},$$

with probability at least $1 - \delta/|A|$ (using standard large deviation bounds). Then the real-valued hypothesis $h(x) = \sum_{a \in A} h_a \phi_a(x)$ satisfies

$$\begin{aligned} \Pr_D[f(x) \neq h(x)] &\leq \mathbb{E}_D[(f(x) - h(x))^2] \\ &= \sum_a (\tilde{f}(a) - \tilde{h}(a))^2 \\ &\leq \sum_{a \in A} (\tilde{f}(a) - \tilde{h}(a))^2 + \frac{\epsilon}{2} \\ &\leq |A| \frac{\epsilon}{2|A|} + \frac{\epsilon}{2} = \epsilon. \end{aligned}$$

The probability that there is a c_a that failed to be a $\sqrt{\frac{\epsilon}{2|A|}}$ -approximation of $\tilde{f}(a)$ is at most δ . Hence with probability at least $1 - \delta$, $h(x)$ satisfies $D(f \triangle h) \leq \epsilon$.

Let us calculate the number of sample points m required by this algorithm. Since $|f\chi_a| \leq 1 \doteq B$, by the Hoeffding bounds (Fact 2.7), we need to take at least

$$m(\sqrt{\epsilon/(2|A|)}, \delta/|A|, B \doteq 1) = \frac{4B^2|A|}{\epsilon} \ln \frac{|A|}{\delta}$$

sample points. In our case A is the set of all n -bit vectors with Hamming weight at most k , so we get

$$|A| = \sum_{i=0}^k \binom{n}{i} \leq n^{k+1}.$$

Thus $m \in \mathcal{O}(\frac{n^{k+1}}{\epsilon} \ln \frac{n^{k+1}}{\delta})$. It can be seen that the running time is dominated by this sample complexity. \square

We will need to distinguish two kinds of *approximations*: mean square and discrete.

DEFINITION 4.1. (*Notions of approximations*)

We say that h ϵ -approximates f under D in the *Mean Square Error* (or MSE) sense if

$\mathbb{E}_{x \in D}[(f(x) - h(x))^2] \leq \epsilon$. On the other hand, we say that h is an ϵ -approximation of f under D in the *discrete* sense if $\Pr_{x \in D}[f(x) \neq h(x)] \leq \epsilon$. Unless otherwise stated, when we say ϵ -approximation we mean approximation in the discrete sense.

Note that Facts 4.1 and 4.2 show that if h is a good real-valued approximation for a Boolean function in the mean square sense then it can be turned into a good approximation in the discrete sense (simply by taking the sign function).

4.2. Subexponential Learning for All Monotone Boolean Functions

The following theorem will show that monotone Boolean functions are PAC learnable under product distributions in subexponential time with respect to the number of inputs, n , and the inverse of the accuracy parameter, $1/\epsilon$. The dependency of the learning complexity on the confidence parameter δ is *only* logarithmic, i.e., $\log(1/\delta)$. For ease of analysis we will first assume that the learner knows the parameters of the underlying product distribution, i.e., the means μ_i are exactly known, for all $i \in [n]$. Later in the next section we will show why we can assume this without loss of generality. More specifically, we show that we will only incur a $\log n$ blow up in the exponent of the time complexity, i.e., the time complexity of the learning algorithm remains unchanged except for some additional logarithmic factors inside the $\tilde{O}()$ term.

THEOREM 4.4. *For any $\epsilon, \delta > 0$, any monotone Boolean function is PAC learnable under any product distribution with error $\epsilon + n^{-c}$, for some constant c , and confidence $1 - \delta$ in time*

$$\exp(\mathcal{O}(\epsilon^{-1} \sqrt{n} \log(\epsilon \sqrt{n}))) \ln \delta^{-1}.$$

Proof Fix a product distribution D . We will use the k -degree Fourier algorithm with

$$k = 1.062 I_D(f) \frac{\sqrt{n}}{(\epsilon/2)}$$

and with the hypothesis set to $h = \sum_{\|a\| \leq k} h_a \phi_a$, where h_a is an estimation of $\tilde{f}(a)$. By Theorem 3.12, h is an $(\epsilon/2)$ -approximation of f . Since $\|a\| \geq |a|$, we have that

$$\{a : \|a\| \leq k\} \subseteq \{a : |a| \leq k\}$$

and hence the k -degree Fourier algorithm only needs to collect (estimate) the Fourier coefficients of Hamming weight at most k . From the definitions of $\|a\|$ and ϕ_a , if $\|a\| \leq k$ we get that

$$|\phi_a(x)| = \left| \prod_{i:a_i=1} \frac{\mu_i - x_i}{\sigma_i} \right| \leq 2^{\|a\|} \left| \prod_{i:a_i=1} (\mu_i - x_i) \right| \leq 2^k,$$

since $|\mu_i - x_i| \leq 1$. Now by the previous section and the above equation, the algorithm outputs a hypothesis that is an approximation of f to within error ϵ with sample size and time complexity of $\exp(\mathcal{O}(\sqrt{n} I_D(f) \epsilon^{-1} \ln \frac{\sqrt{n} \epsilon}{I_D(f)})) \ln \delta^{-1}$. By Lemma 3.13, we note that $I_D(f) \leq 1$, for any monotone Boolean function f , because

$$I_D(f)^2 = \sum_i (2\sigma_i I_{D,i}(f))^2 = \sum_i \tilde{f}(e_i)^2 \leq 1$$

by Parseval's identity. Using simple calculus, we see that the function $x \log(1/x)$ is bounded from above by 1 in the interval $(0, 1)$. Thus we have $I_D(f) \log \frac{1}{I_D(f)} \leq 1$, and therefore

$$I_D(f) \log \frac{\epsilon \sqrt{n}}{I_D(f)} = I_D(f) \log \frac{1}{I_D(f)} + I_D(f) \log(\epsilon \sqrt{n}) = \mathcal{O}(\log(\epsilon \sqrt{n})).$$

This analysis proves the time complexity stated in the theorem. \square

We note that using the above algorithm with subexponential time, the best achievable error rate is $\epsilon = \frac{1}{\sqrt{n}}$. In a later subsection we show that this is the best possible error rate up to a $\mathcal{O}(\log n)$ factor.

4.3. Analysis of Learning under Any Product Distributions

In this section we address the issue of PAC learning monotone Boolean functions under a product distribution when the parameters of distribution are unknown, i.e.,

the learner is not told the precise values of μ_i 's. Schapire's thesis [S] also contains some work done on learning under general product distributions but for a different concept class.

We fix a product distribution $D = (\mu_1, \mu_2, \dots, \mu_n)$. First we argue that we may ignore all μ_i 's that are less than n^{-2} or greater than $1 - n^{-2}$ since this will contribute only an additive factor of n^{-1} to the final error. We show this in the following. We call an assignment $a \in \{0, 1\}^n$ *good* if for all $i \in [n]$ we have:

$$a_i = 0 \text{ whenever } \mu_i < n^{-2} \text{ and } a_i = 1 \text{ whenever } \mu_i > 1 - n^{-2}$$

The probability of obtaining a good assignment in sampling is at least $1 - n^{-1}$. Let $\mathcal{G} \subseteq \{0, 1\}^n$ be the set of all *good* assignments. Suppose that h is an ϵ -approximation to the target function f on the set \mathcal{G} . Then

$$\Pr_D[h(x) \neq f(x)] \leq \Pr_D[h(x) \neq f(x) \mid x \in \mathcal{G}] + \Pr_D[x \notin \mathcal{G}] \leq \epsilon + n^{-1}.$$

Thus we may assume that $\mu_i \in (n^{-2}, 1 - n^{-2})$, for all $i \in [n]$.

We define the following parameters:

$$k \doteq \mathcal{O}(\sqrt{n}/\epsilon), \quad M \doteq 2^k = 2^{\mathcal{O}(\sqrt{n}/\epsilon)}.$$

We will estimate each μ_i to within an error of $M^{-(\log n + 4)}$. Note that using Hoeffding bounds (Fact 2.7) this requires approximately $(M^{\log n})^{\mathcal{O}(1)}$ sample points. Suppose that $\hat{\mu}_i$, $i \in [n]$, are the estimated means and let \hat{D} denote the product distribution $(\hat{\mu}_1, \dots, \hat{\mu}_n)$. Also let $\hat{\phi}_a$ denote the *basis* function at a according to \hat{D} .

We will consider three hypothesis quantities,

$$h_A(x) = \sum_{a \in S} E_D[f \hat{\phi}_a] \hat{\phi}_a(x), \quad h_B(x) = \sum_{a \in S} E_{\hat{D}}[f \hat{\phi}_a] \hat{\phi}_a(x), \quad h_C(x) = \sum_{a \in R} E_D[f \phi_a] \phi_a(x),$$

where R and S are the sets of assignments for which the algorithm needs to estimate the ϕ_a 's and $\hat{\phi}_a$'s, respectively. Notice that from the proof of Theorem 4.4 and from the fact that $I_D(f) \leq 1$, for all monotone Boolean functions f , we have $|\hat{\phi}_a(x)| \leq M$.

Ideally we want to learn h_C , but because only approximations of the means μ_i can

be found we will try to learn h_B . Now since the example oracle $EX(f, D)$ gives the examples according to the original product distribution D and not \hat{D} we will instead learn h_A . Since the learning parameters (the number of coefficients) are computed for \hat{D} we have

$$\mathbf{E}_{\hat{D}}[(h_B(x) - f(x))^2] \leq \epsilon.$$

Suppose $\mu_i + \tau_i$ is the estimation for μ_i , where $|\tau_i| < M^{-(\log n+4)}$. Notice that

$$\begin{aligned} \frac{\hat{D}(x)}{D(x)} &= \prod_{x_i=1} \left(1 + \frac{\tau_i}{\mu_i}\right) \prod_{x_i=0} \left(1 - \frac{\tau_i}{1 - \mu_i}\right) \\ &\leq \left(1 + \frac{n^2}{M^{\log n+4}}\right)^n \\ &\in 1 + \mathcal{O}\left(\frac{1}{M^{\log n+3}}\right). \end{aligned}$$

In the above we have used the following simple upper bounds

$$1 + \frac{\tau_i}{\mu_i} \leq 1 + \frac{|\tau_i|}{\mu_i}, \quad 1 - \frac{\tau_i}{1 - \mu_i} \leq 1 + \frac{|\tau_i|}{1 - \mu_i},$$

and the fact that $\mu_i, 1 - \mu_i \geq n^{-2}$ and $(1 + x)^n \sim 1 + nx$, for x significantly smaller than 1, i.e., $x \ll 1$. We will also bound the expression $D(x)/\hat{D}(x)$ from above. First note that

$$1 - \frac{\tau_i}{\mu_i + \tau_i} \leq 1 + \frac{|\tau_i|}{\mu_i - |\tau_i|}, \quad 1 + \frac{\tau_i}{1 - \mu_i - \tau_i} \leq 1 + \frac{|\tau_i|}{1 - \mu_i - |\tau_i|},$$

and that

$$\mu_i - |\tau_i|, 1 - \mu_i - |\tau_i| \geq n^{-2} - M^{-(\log n+4)} \geq n^{-2}/2,$$

for sufficiently large n . Using these observations, we see that

$$\begin{aligned} \frac{D(x)}{\hat{D}(x)} &= \prod_{x_i=1} \left(1 - \frac{\tau_i}{\mu_i + \tau_i}\right) \prod_{x_i=0} \left(1 + \frac{\tau_i}{1 - \mu_i - \tau_i}\right) \\ &\leq \left(1 + \frac{2n^2}{M^{\log n+4}}\right)^n \\ &\in 1 + \mathcal{O}\left(\frac{1}{M^{\log n+3}}\right) \end{aligned}$$

Now we can bound the performance of h_B under the true distribution D as follows.

$$\begin{aligned}
\mathbf{E}_D[(h_B(x) - f(x))^2] &= E_{\hat{D}} \left[\frac{D(x)}{\hat{D}(x)} (h_B(x) - f(x))^2 \right] \\
&\leq E_{\hat{D}}[(h_B(x) - f(x))^2] \max_x \frac{D(x)}{\hat{D}(x)} \\
&\in \epsilon \left(1 + \mathcal{O} \left(\frac{1}{M^{\log n + 3}} \right) \right) \\
&= \epsilon + \mathcal{O} \left(\frac{\epsilon}{M^{\log n + 3}} \right).
\end{aligned}$$

Therefore h_B is also a good approximation of f with respect to the distribution D .

Now we show that h_A is good enough. We have

$$\begin{aligned}
|h_A(x) - h_B(x)| &= \left| \sum_{a \in S} E_D[f \hat{\phi}_a] \hat{\phi}_a(x) - \sum_{a \in S} E_{\hat{D}}[f \hat{\phi}_a] \hat{\phi}_a(x) \right| \\
&= \left| \sum_{a \in S} E_D[f \hat{\phi}_a (1 - (\hat{D}/D))] \hat{\phi}_a(x) \right| \\
&\leq |S| M^2 \max_x \left| 1 - \frac{\hat{D}(x)}{D(x)} \right| \\
&\leq |S| M^2 M^{-(\log n + 3)} \\
&\in \mathcal{O} \left(\frac{n}{M} \right),
\end{aligned}$$

because

$$|S| \leq \sum_{i=0}^k \binom{n}{i} \leq n^{k+1} = 2^{(k+1) \log n} = n M^{\log n}.$$

Therefore

$$\begin{aligned}
\mathbf{E}_D[(h_A(x) - f(x))^2] &\leq 2\mathbf{E}_D[(h_A(x) - h_B(x))^2 + (h_B(x) - f(x))^2] \\
&= 2\mathbf{E}_D[(h_A(x) - h_B(x))^2] + 2\mathbf{E}_D[(h_B(x) - f(x))^2] \\
&\in 2\epsilon + \mathcal{O} \left(\frac{n}{M} \right) \\
&= 2\epsilon + n^{-c},
\end{aligned}$$

for some constant $c > 0$. This completes the analysis for PAC learning monotone Boolean functions when the product distribution is unknown. We restate the theorem

in the following.

THEOREM 4.5. *For any $\epsilon, \delta > 0$, any monotone Boolean function over $\{0, 1\}^n$ is PAC learnable under an unknown product distribution with sample and time complexity of $\exp(\tilde{O}(\epsilon^{-1}\sqrt{n}\log(\epsilon\sqrt{n})))\log\delta^{-1}$ with an error of at most $\epsilon + n^{-c}$, for some constant c , and with confidence δ .*

4.4. Proving Near Optimal Performance

We will prove several statements showing that the monotone learning result in Theorem 4.4 is nearly best possible or optimal in terms of the time complexity, achievable error rate, and sample complexity.

4.4.1. Error Rates. First, we claim that the error rate achieved in the algorithm is the best possible for a subexponential time algorithm.

THEOREM 4.6. *Any PAC learning algorithm for monotone Boolean functions under the uniform distribution running in time 2^{cn} , for any $c < 1$, will output an approximation with an error of at least $\Omega\left(\frac{1}{\sqrt{n}\log n}\right)$.*

Proof There are at least $m(n) \doteq 2^{\binom{n}{n/2}} \geq 2^{d2^n/\sqrt{n}}$ monotone Boolean functions over n variables, for some constant $d < 1$. We have used here the approximation $\binom{n}{n/2} \sim 2^n/\sqrt{n}$. Suppose A is the ϵ -approximation algorithm for any monotone Boolean function. If A outputs a hypothesis h then h can ϵ -approximate at most

$$k(n) \doteq \sum_{i \leq \epsilon 2^n} \binom{2^n}{i} \leq 2^{2^n H(\epsilon)}$$

Boolean functions, by Fact 2.3. Assuming A runs in time 2^{cn} , for some constant $c < 1$, then A can output at most $2^{2^{cn}}$ possible hypotheses. Therefore we must have

$$2^{2^{cn}} k(n) \geq m(n)$$

which implies $2^{cn} + 2^n H(\epsilon) \geq \frac{d2^n}{\sqrt{n}}$. So for sufficiently large n , we have $H(\epsilon) \gg n^{-1/2}$. Note that $H(x) = x \log(1/x) + (1-x) \log(1/(1-x)) \sim x \log(1/x)$, for $x \sim 0$. This is

because $\lim_{x \rightarrow 0} (1-x) \log(1/(1-x)) = 0$. Using this to simplify, we get $\epsilon \log(1/\epsilon) \gg n^{-1/2}$. The change of variable $\xi = 1/\epsilon$ yields the equation $(\log \xi)/\xi \gg n^{-1/2}$ which implies $\xi \ll n^{1/2} \log \xi$. Chasing the last equation further, we get $\xi \ll n^{1/2} \log n$. This yields the inequality $\epsilon \gg (n^{1/2} \log n)^{-1}$. \square

The next corollary gives a lower bound for the error rate of any learning algorithm that runs in time bounded by $2^{\mathcal{O}(\sqrt{n})}$.

COROLLARY 4.7. *Any learning algorithm for monotone Boolean functions under the uniform distribution with a running time bounded by $2^{\mathcal{O}(\sqrt{n})}$ cannot achieve an error smaller than $\Omega(1/(n^{1/4} \log n))$.*

Proof Let A be an algorithm that runs in time $2^{\alpha\sqrt{n}}$, for some constant α , and that learns any monotone Boolean function over n variables within an error of $\epsilon(n)$, for any n . We will construct another algorithm B that learns any monotone Boolean function over n variables in time 2^{cn} , for some $c < 1$, and achieves an error of $2\epsilon((cn)^2)$. By Theorem 4.6, we must have $2\epsilon((cn)^2) = \Omega(1/(\sqrt{n} \log n))$, which implies the claim.

Let $m = (cn)^2$. The algorithm B with input $EX(f, U_n)$, where $f = f(x_1, \dots, x_n)$ is a Boolean function on n variables and U_n is the uniform distribution over $\{0, 1\}^n$, will use the algorithm A for functions over m variables. For this the algorithm B will pad the examples $(x, f(x))$ from $EX(f, U_n)$ into examples of the form $(x \circ \tilde{x}, f(x))$, where $x = (x_1, \dots, x_n)$ and $\tilde{x} = (x_{n+1}, \dots, x_m) \in U_{m-n}$. The error rate achievable by algorithm A to learn monotone Boolean functions on m variables is $\epsilon = \epsilon(m)$. The algorithm A outputs a hypothesis h with

$$\mathbf{E}_{x, \tilde{x}}[f(x \circ \tilde{x}) \neq h(x \circ \tilde{x})] < \epsilon.$$

Algorithm B proceeds by randomly and uniformly choosing values b_{n+1}, \dots, b_m and returning the hypothesis $h(x_1, \dots, x_n, b_{n+1}, \dots, b_m)$. Note that

$$\mathbf{E}_{x, \tilde{x}}[f(x \circ \tilde{x}) \neq h(x \circ \tilde{x})] = \mathbf{E}_{\tilde{x}}(\mathbf{E}_x[f(x \circ \tilde{x}) \neq h(x \circ \tilde{x})]) < \epsilon.$$

By using Markov's inequality, Fact 2.5, with probability at least $1/2$, a random b_{n+1}, \dots, b_m gives a $2\epsilon(m)$ -approximation to f . If necessary, the algorithm B may repeat this process often enough to increase the probability of obtaining a 2ϵ -approximation to f . \square

4.4.2. Bounds for the Low-degree Fourier Algorithm. We now investigate the best error rate of the low-degree Fourier algorithm. The first theorem shows that there exists a monotone Boolean function f for which any ϵ -approximation (in the Mean Square Error sense) that uses the low-degree Fourier coefficients f , for $\epsilon = n^{-1/2}$, must collect all coefficients of degree less or equal to cn , for some constant $c < 1$.

THEOREM 4.8. *For any constant $c < 1$, there is a monotone Boolean function f which satisfies*

$$\sum_{|a| \geq cn} \hat{f}^2(a) \geq \Omega(1/(\sqrt{n} \log n)).$$

Proof Assume for contradiction that there is some constant $c < 1$ such that for any monotone function f

$$\sum_{|a| \geq cn} \hat{f}^2(a) \leq \alpha \frac{1}{\sqrt{n} \log n}$$

for some constant α . This implies that the low-degree algorithm which searches all coefficients of degree at most cn will approximate f within an error of $\mathcal{O}(1/(\sqrt{n} \log n))$.

This contradicts Theorem 4.6 modulo constant factors. \square

The second theorem shows that to approximate the majority function with error $\sim n^{-1/2}$ we need to collect all of its Fourier coefficients of order $\mathcal{O}(\sqrt{n})$.

THEOREM 4.9. *The majority function MAJ satisfies $\sum_{|a| \geq \alpha\sqrt{n}} \widehat{MAJ}^2(a) \geq \alpha/\sqrt{n}$, for some absolute constant α .*

Proof Since the majority function $MAJ(x)$ is a symmetric function, the influences of all variables are equal. Thus we have $\sum_a |a| \widehat{MAJ}^2(a) = \sum_{i=1}^n I_i(MAJ) = nI_1(MAJ)$.

To get a bound on $I_1(MAJ)$, note that $I_1(MAJ) \geq 2^{-n} \binom{n}{n/2} \geq \frac{c}{\sqrt{n}}$, for some constant c . Now we have the following.

$$\begin{aligned} c\sqrt{n} &\leq \sum_a |a| \widehat{MAJ}(a)^2 = \sum_{|a| \geq \frac{c}{2}\sqrt{n}} |a| \widehat{MAJ}(a)^2 + \sum_{|a| < \frac{c}{2}\sqrt{n}} |a| \widehat{MAJ}(a)^2 \\ &\leq n \sum_{|a| \geq \frac{c}{2}\sqrt{n}} \widehat{MAJ}(a)^2 + \frac{c}{2}\sqrt{n}. \end{aligned}$$

Finally we obtain $\sum_{|a| \geq \frac{c}{2}\sqrt{n}} \widehat{MAJ}(a)^2 \geq \frac{c}{2\sqrt{n}}$. \square

4.4.3. Sample Complexity. So far we have considered the time complexity of PAC learning monotone Boolean functions. As a final result, we mention a known application of the Vapnik and Chernovenkis dimension (or VC dimension) to obtain a bound for the *sample complexity* of PAC learning monotone Boolean functions under an *arbitrary* distribution. Sample complexity refers to the number of random labelled examples that a learning algorithm requires to find a good approximation.

In the following we briefly recall some definitions and facts about the Vapnik and Chernovenkis dimension.

DEFINITION 4.2. (*Vapnik-Chernovenkis dimension*)

For a subset $A \subseteq \{0,1\}^n$ and a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, we denote $f|_A$ to be the function $f|_A : A \rightarrow \{0,1\}$ such that $f|_A(x) = f(x)$ if $x \in A$ and that is undefined otherwise.

If C is a class of Boolean functions over $\{0,1\}^n$ then C shatters $A \subseteq \{0,1\}^n$ if

$$\{f|_A : f \in C\} = 2^A,$$

or, alternatively, if for every Boolean function $g : A \rightarrow \{0,1\}$ there exists a Boolean function $f \in C$ such that $f|_A = g$, i.e., g equals f when the domain is restricted to A . The Vapnik-Chernovenkis dimension of a concept class C , $VCdim(C)$, is the cardinality of the largest subset A that is shattered by C .

The following general lower bound result was proved by Ehrenfeucht *et al.* [EHKV88].

THEOREM 4.10. [EHKV88] *Let C be a concept class. Then any PAC learning algorithm for C with accuracy ϵ and confidence δ must use a sample size of*

$$\Omega\left(\frac{1}{\epsilon} \ln \frac{1}{\delta} + \frac{1}{\epsilon} VCdim(C)\right)$$

We now note the following easy fact.

FACT 4.11. *The Vapnik-Chervonenkis dimension of the class of monotone Boolean functions on n variables is at least $\binom{n}{n/2} \sim 2^n/\sqrt{n}$.*

Proof It is easy to see that $A = \{a \in \{0,1\}^n : |a| = \lfloor n/2 \rfloor\}$ is shattered by all monotone Boolean functions on n inputs. \square

COROLLARY 4.12. *Any PAC learning algorithm for monotone Boolean functions under an arbitrary distribution with error ϵ and confidence δ (for sufficiently small ϵ and δ) requires at least $\Omega\left(\frac{2^n}{\epsilon\sqrt{n}} + \frac{1}{\epsilon} \ln \frac{1}{\delta}\right)$ examples.*

The above observation also has been made by Kearns and Valiant in [KV89, K]. Notice that the above gives a complexity bound that is nearly exponential, i.e., $2^n/\sqrt{n}$, which is larger than $2^{\sqrt{n}}$. This is because the bound above is for the distribution-free case. But as mentioned in [KV89], the class of monotone Boolean functions is not polynomially PAC learnable under the uniform distribution (this claim was attributed to Ehrenfeucht and Haussler).

4.5. Polynomial-time PAC Learning

In this section we shift our focus to efficient or polynomial-time PAC learning of classes of monotone Boolean functions. We will consider both the weak and strong variants of the PAC model and provide some new learning results in both models.

Kearns and Valiant [KV89] proved that all monotone Boolean functions are weakly PAC learnable under the uniform distribution with error $1/2 - 1/(2n)$. We will improve their result and simplify their proof. We show that there is a weak PAC learning algorithm with error $1/2 - \Omega(\log^2 n/n)$ under the uniform distribution and

there is a weak PAC learning algorithm with error $1/2 - \Omega(1/n)$ under any product distribution.

We will need the following result due to Kahn, Kalai and Linial [KKL88].

LEMMA 4.13. [KKL88] *Let f be a Boolean function with $p = \Pr[f(x) = 1] \leq 1/2$. Then*

$$\sum_{i=1}^n I_i(f)^2 \geq \frac{p^2 (\log n)^2}{5n}.$$

THEOREM 4.14. *There is a polynomial-time weak PAC learning algorithm with error $\epsilon = \frac{1}{2} - \Omega\left(\frac{\log^2 n}{n}\right)$ for any monotone Boolean function under the uniform distribution.*

Proof We will assume without loss of generality that $p = \Pr[f(x) = 1] \leq 1/2$, since we can take $\neg f(\neg x_1, \dots, \neg x_n)$ otherwise. This transformation does not affect the influences.

If $p < 1/4$ (we can estimate this) then the trivial algorithm that outputs the hypothesis $h \equiv 0$ is a weak PAC learning algorithm. Otherwise, if $p \geq 1/4$, since $I_i(f)^2 = \hat{f}^2(e_i)$ and using Lemma 4.13, $\sum_{i=1}^n \hat{f}^2(e_i) \geq p^2 \log^2 n / (5n) \geq \log^2 n / 80n$. Combining this with Fact 4.2, we use the LMN algorithm to estimate all Fourier coefficients of f on the set $A = \{e_i : i \in [n]\}$. This yields a weak learning algorithm for f with the claimed accuracy. \square

THEOREM 4.15. *For any constant k , there is a polynomial-time weak PAC learning algorithm with error $\epsilon = \frac{1}{2} - \frac{k}{n}$ for any monotone Boolean function under any product distribution.*

Proof Given k , we set $\alpha = \sqrt{2.124k}$ (the constant 2.124 comes from the constant in Theorem 3.12). If $\sum_{\|a\| \geq \alpha} \hat{f}^2(a) \leq 1/2$ then, by Fact 4.2, we immediately obtain a weak PAC learning algorithm with error $1/4$ by using the LMN algorithm to estimate all Fourier coefficients of f on the set $\{a : \|a\| \leq \alpha\}$.

On the other hand, if we have $\sum_{\|a\| \geq \alpha} \tilde{f}^2(a) > 1/2$, then by Theorem 3.12, this implies that

$$1.062 \frac{\sqrt{n}}{\alpha} I_D(f) \geq \frac{1}{2}.$$

Hence $\sum_i \tilde{f}^2(e_i) \geq \frac{\alpha^2}{2.124n} = k/n$. So we can use the LMN algorithm to approximate all Fourier coefficients of weight at most 1.

One detail left out in the arguments above is the fact that the learning algorithm must first estimate the means μ_i , $i \in [n]$, of the underlying product distribution. By an analysis similar to the ones given in Section 4.3, the above arguments can be adapted accordingly. \square

We now switch to *strong* PAC learning and consider proper subclasses of monotone Boolean functions. Sakai and Maruoka [SM94] proved that the class of monotone $\mathcal{O}(\log n)$ -term DNF formulas is PAC learnable under the uniform distribution. We improve their result in two ways; first, we will learn a larger subclass of monotone Boolean functions, and second, we will allow constant-bounded product distributions. Recall that a product distribution $D = (\mu_1, \dots, \mu_n)$ is called *constant-bounded* if there is a constant $c \in [0, 1/2]$, independent of n , such that $\mu_i \in [c, 1 - c]$, for all $i \in [n]$.

DEFINITION 4.3. (*Generalization of Monotone DNF and CNF*)

Let $\mathcal{MON}(k)$ be the representation class of Boolean functions of the form $f(T_1, \dots, T_k)$, where f is an arbitrary monotone Boolean function on $\mathcal{O}(k)$ inputs and each T_i is a monotone conjunction or a monotone disjunction over n variables.

THEOREM 4.16. *The class $\mathcal{MON}(\log n)$ is PAC learnable under constant-bounded product distributions.*

Proof Let $f \in \mathcal{MON}(\log n)$ be of the form $f(x) = g(\tau(x), \dots, \tau_k(x))$, where g is a monotone function on $k = c_1 \log n$ inputs, for some constant c_1 , and each τ_i is either a monotone disjunction or a monotone conjunction over the variables x_1, x_2, \dots, x_n .

Since the product distribution D is constant-bounded, say c -bounded, for some constant c , there is a way of locating efficiently (via sampling) the variables that appear in the target function f . From Lemma 3.8, the influence $I_{D,i}(f)$ of variable x_i on f under distribution D is given by

$$I_{D,i}(f) = \mathbb{E}_D[f(x) \neq f(x \oplus e_i)] = \frac{1}{4\sigma_i^2} \sum_{a:a_i=1} \tilde{f}(a)^2.$$

The first expression shows that $I_{D,i}(f)$ can be estimated via sampling. Since D is constant-bounded, σ_i is a constant and therefore if the influence of variable x_i on f , i.e., $I_{D,i}(f)$, is small then we may assume that $x_i = 0$. This will incur only a negligible approximation error (in the MSE sense) by Fact 4.1. In fact this can be done also for *projections* of f .

We call a variable x_i *relevant* for a Boolean function f with respect to a distribution D if there is $a \in \{0,1\}^n$ with $D(a) > 0$ so that $f(a) \neq f(a \oplus e_i)$, i.e., $I_{D,i}(f) > 0$. Our learning strategy is to collect *relevant* variables, i.e., variables with non-negligible influences, in a small depth decision tree. Due to sampling, we might need to impose a threshold at which we will deem an influence negligible or not (instead of nonzero or not).

Since f depends on $c_1 \log n$ monotone disjunctions or conjunctions that feed into some unknown monotone function g , a random assignment to a variable that appears in a monotone disjunction or conjunction will turn the latter into a constant with probability at least c . This is because $\Pr[x_i = 0], \Pr[x_i = 1] \geq c$, for c -bounded distributions.

We now build a Boolean decision tree of depth d based on the variables with *non-negligible* influences, i.e., they appear in f . Using one of the Chernoff bounds expressions

$$\Pr[X < (1 - \delta)\mu] < \exp(-\delta^2\mu/2),$$

we can force this to be bounded from above by ϵ/n^{c_1} . To this end we select $\delta = 1/2$

and

$$\mu \geq cd \geq \frac{2}{\delta^2} \ln \left(\frac{n^{c_1}}{\epsilon} \right).$$

This implies that we need a depth of at least

$$d \geq \frac{2}{c\delta^2} \ln \left(\frac{n^{c_1}}{\epsilon} \right) \in \mathcal{O}(\log n + \log(1/\epsilon)).$$

Using this we can claim that the probability that a root to leaf will, with probability at least $1 - \epsilon/n^{c_1}$, set all monotone conjunctions or disjunctions in f to constant. This is because if X denotes the number of *kills* (elimination of a conjunction or disjunction), then

$$\Pr[\text{leaf} \neq \text{constant}] = \Pr[X < c_1 \log n] \leq \Pr[X < (1 - \delta)\mu],$$

with the choices of δ and μ as above. Hence with probability at least $1 - \epsilon$, any root to leaf path will set all conjunctions and disjunctions to constant.

We remark that each *leaf* in the tree must make a decision of whether the projected function (induced by the unique path from the root to that leaf) is non-constant or not. The leaf is not expanded if sampling shows that the projected function is already constant.

So the hypothesis of this decision tree containing relevant variables from f has an error probability of at most ϵ (which is the event that some leaf is not constant).

The confidence parameter (for PAC learning) will get introduced in the high probability sampling steps. \square

For an alternative proof of the above result see [B95].

CHAPTER 5

Learning Bounded Width Branching Programs

A lot of people are afraid of heights.

Not me. I'm afraid of widths.

— Steven Wright

The branching program is a well-studied model of computation in complexity theory. These were used to study and prove non-trivial space lower bounds. In an early work, Borodin, Dolev, Fich, and Paul [BDFP86] conjectured that the majority function is not computable by branching programs with constant width and polynomial size. This was disproved by Barrington [Bar89] who proved that the Boolean functions computable by width five *permutation* branching programs are equivalent to the Boolean functions computable by families of bounded fan-in polynomial size and logarithmic depth Boolean circuits. This shows that the majority function is computable by width five branching programs since it is computable by an NC^1 circuit family. This result indicates the surprising power of bounded width branching programs.

In this chapter, we will use the branching program model as an alternative representation class for studying the learnability of Boolean functions. Using Barrington's result combined with the negative result of Angluin and Kharitonov [AK95], we have a boundary for non-learnability: we cannot hope to learn width five branching programs if we make some natural cryptographic assumptions. On the other hand, by using the alternative structure provided by branching programs, one can perhaps gain more insight into the learnability of Boolean functions that lie below the class NC^1 .

We approach the learnability of branching programs using different techniques. In the first part, we use Fourier techniques to show that the representation class of *monotone* width two branching programs is efficiently PAC learnable with membership queries under the uniform distribution. This can be regarded as an extension of Jackson's DNF learning result [J94]. Next we show that the representation class of width two branching programs with a constant number of sinks is efficiently exactly learnable from equivalence queries alone. A previous result of Bshouty, Tamon, and Wilson [BTW96] showed that width two branching programs with two sinks is efficiently PAC learnable under any distribution.

In the last part of this chapter we apply the novel technique of learning *multiplicity automata* due to Bergadano and Varricchio [BV94] to prove the exact learnability of several classes of bounded width permutation branching programs with equivalence and membership queries.

5.1. Characterizations of Width Two Branching Programs

We recall some definitions of subclasses of width two branching programs as introduced by Borodin, Dolev, Fich, and Paul [BDFP86].

DEFINITION 5.1. A width two branching program is *strict* if it has exactly one accepting sink and one rejecting sink. A width two branching program is *monotone* if it has exactly one rejecting sink.

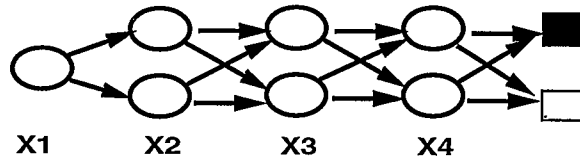


FIGURE 5.1. Example of a strict width 2 branching program

It is easy to see that any DNF formula can be converted into a width two monotone

branching program. Hence the representation class of DNF formulas is contained in the representation class of monotone width two branching programs. Moreover the inclusion is strict since the parity function is computable by a polynomial size strict width two branching program while it is known that it is not computable by polynomial size DNF formulas.

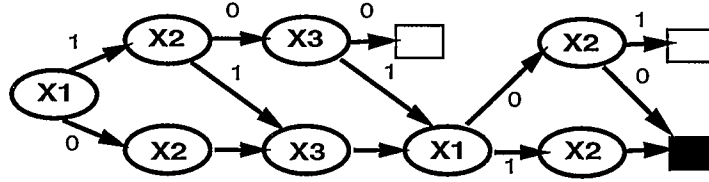


FIGURE 5.2. Example of a monotone width two branching program for DNF $x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2$

We mention an alternative characterization of strict width two branching programs in terms of parity decision lists as shown by Bshouty, Tamon, and Wilson [BTW96]. First we define notation for describing classes of parity functions. The class of parity functions that depend on at most k relevant inputs will be denoted \oplus_k . In notation, $\oplus_k = \{(a \cdot x) \oplus b \mid a \in \{0, 1\}^n, b \in \{0, 1\}, |a| \leq k\}$. Note that \oplus_1 is the set of literals (and including the constant functions) and \oplus_n is the set of all parity functions.

FACT 5.1. [BTW96] *The class SW_2 of strict width two branching programs is equivalent to the class (\oplus_2, \oplus_n) -DL. Moreover, any decision list in (\oplus_2, \oplus_n) -DL has at most n^2 nodes.*

5.2. Learning *Monotone* Width Two Branching Programs in the PAC Model

In this section we will prove that monotone width two branching programs are efficiently PAC learnable with membership queries under the uniform distribution. This result extends an earlier theorem of Jackson [J94] who proved that DNF formulas are efficiently PAC learnable with membership queries under the uniform distribution.

Our proof relies on the Fourier transform method. To avoid confusion, in this section we will adopt the following convention for stating Boolean functions. Recall that we denote “normal” Boolean functions, i.e., ones with the range $\{0,1\}$, with lower-case letters, such as $f : \{0,1\}^n \rightarrow \{0,1\}$, and their *corresponding* $\{-1,+1\}$ -range *counterparts* with upper-case letters, e.g., $F : \{0,1\}^n \rightarrow \{-1,+1\}$, and recall that they are related by $F = 2f - 1$ and $f = \frac{1+F}{2}$. Whenever necessary we will remind the reader of this convention.

5.2.1. The Harmonic Sieve Learning Algorithm of Jackson. As we have seen from previous chapters, there is a simple PAC learning algorithm due to Linial, Mansour, and Nisan [LMN93] that works by estimating all Fourier coefficients of the target function on a specific isolated region in the Boolean cube. For the class of AC^0 and monotone Boolean functions we know that this region is the set of all coefficients with Hamming weight at most polylogarithmic in n and square root of n , respectively.

Unfortunately there are classes of functions, such as polynomial-sized decision trees, which have a small collection of “heavy” frequencies but whose locations are dependent on the structure of the function. So one cannot use the algorithm in [LMN93] since the algorithm does not know where the important frequencies are. In 1991, Kushilevitz and Mansour [KM93] gave an important algorithmic procedure to search for heavy frequencies using membership queries. Let us call their algorithm the *KM algorithm*. Their method enables one to locate the significant Fourier coefficients without a priori knowledge of where they lie in $\{0,1\}^n$. So now any class of functions which has a small collection of important Fourier coefficients is efficiently learnable in the PAC model with membership queries *under the uniform distribution*. In particular this implies that the class of polynomial size decision trees are efficiently PAC learnable with membership queries under the uniform distribution.

The next progress in the Fourier-based learning algorithms came when Jackson applied the idea of boosting to the KM learning algorithm. He devised an algorithm

which he called the *Harmonic Sieve* algorithm that is capable of learning DNF formulas in the PAC model with membership queries under the uniform distribution. Intuitively, Jackson's idea is as follows. First one shows that a Boolean function has a mild correlation with a parity function. In the Fourier language this translates to: one of the frequencies is a good "weak" approximator to the function. Next, one uses the KM algorithm to find this frequency pattern. At this point we have already obtained a weak learning algorithm. To get a strong learning algorithm we apply a hypothesis boosting algorithm that combines several weak learning algorithms into a single strong learning algorithm. In his beautiful paper [J94] Jackson proved the following theorem.

THEOREM 5.2. [J94] *The class of DNF formulas is efficiently PAC learnable from membership queries under the uniform distribution.*

We outline the technical arguments used by Jackson and illustrate how we modify them to prove the learnability of monotone width two branching programs. The first key fact about DNF formulas is that each correlates well with *some* parity function χ_A , $A \in \{0,1\}^n$, under *any* distribution. Recall that $\chi_A(x) = (-1)^{\sum_{i \in A} x_i}$ defines a parity test on the bits designated by A viewed as a subset of $[n]$. Thus, if f is a DNF formula of size s (f has s terms) and D is an arbitrary distribution, then there is some A such that

$$|\mathbf{E}_D[F\chi_A]| \geq \frac{1}{2s+1}.$$

We remind the reader that F is the $\{-1, +1\}$ -version of f . Using the above inequality, since $\mathbf{E}_D[F\chi_A] = \Pr_D[F = \chi_A] - \Pr_D[F \neq \chi_A]$, we derive the following. Assume without loss of generality that $\mathbf{E}_D[F\chi_A]$ is positive (the other case is symmetrically similar). Then

$$\mathbf{E}_D[F\chi_A] = 1 - 2\Pr_D[F \neq \chi_A] \geq \frac{1}{2s+1} \implies \Pr_D[F \neq \chi_A] \leq \frac{1}{2} - \frac{1}{2(2s+1)}.$$

This is good news since it means that the parity function χ_A is a potential hypothesis for *weak* learning f . The problem is that we do not know the set A .

The second key fact is that there is an efficient algorithm due to Kushilevitz and Mansour [KM93] to find parities that correlate well with certain Boolean functions assuming that the underlying distribution is uniform. So weakly learning DNF under the uniform distribution is possible by combining these two facts [BFJ⁺94].

The third ingredient is a *boosting* algorithm, developed by Freund [F90], that can turn any weak learning algorithm into a strong learning algorithm. This does not solve the DNF learning problem immediately since the boosting algorithm assumes that the weak learning algorithm works under *arbitrary* distributions (not just the uniform distribution). This is because the boosting method works by running the weak learning algorithm on a carefully chosen set of modified distributions.

Jackson then supplied the missing pieces: he proved that the boosting algorithm of Freund combined with a modified version of Kushilevitz and Mansour's algorithm will still work since the distribution is not being perturbed too much (he quantified precisely this intuition in [J94]). Also by the first fact, DNF formulas are still guaranteed to correlate well with some parity when the distribution is slightly changed. In fact the only property that is ever needed about DNF formulas to get the learning result is the first fact. The resulting algorithm is the *Harmonic Sieve* learning algorithm.

To prove our PAC learning result we will show in the next section that the first fact holds for monotone width two branching programs. Using this we can then claim the following theorem.

THEOREM 5.3. *The class \mathcal{MW}_2 of monotone width two branching programs is efficiently PAC learnable with membership queries under the uniform distribution.*

5.2.2. A Fourier Correlation Lemma. In this section we prove the following lemma that states that any monotone width two branching program correlates well with some parity function under *any* distribution. The fact that the lemma is true

for any distribution is critical for the boosting stage in Jackson's harmonic sieve algorithm.

LEMMA 5.4. *For any $F \in \mathcal{MW}_2$ with s accepting sinks and for any distribution D there is a parity χ_C such that*

$$|\mathbf{E}_D[F\chi_C]| \geq \frac{1}{2sn^2 + 1}.$$

Proof Let $f \in \mathcal{MW}_2$ be computed by a monotone width two branching program with s sinks. Note that each accepting sink defines a subportion of the branching program that is a strict width two branching program. Let g_1, \dots, g_s be the functions computed by the s subportions associated with the s strict width two branching programs (see Figure 5.2). Note that

$$f = g_1 \vee g_2 \vee \dots \vee g_s.$$

Hence there is a subportion g_i so that

$$\Pr_D[g_i = 1] \geq \frac{1}{s} \Pr_D[f = 1].$$

We fix our attention on this subportion g_i and call it g .

Using Fact 5.1, $g \in \mathcal{SW}_2$ is equivalent to a decision list in (\oplus_2, \oplus_n) -DL. If g is defined as

$$G = [(\chi_{a_1}, \chi_{b_1}), (\chi_{a_2}, \chi_{b_2}), \dots, (\chi_{a_m}, \chi_{b_m})]$$

(recall that G is the $\{-1, +1\}$ -version of g) then it can be rewritten as

$$g = \sum_{i=1}^m \frac{1 + \chi_{a_i}}{2} \frac{1 + \chi_{b_i}}{2} \prod_{j=1}^{i-1} \frac{1 - \chi_{a_j}}{2}.$$

Let

$$h_i = \frac{1 + \chi_{a_i}}{2} \frac{1 + \chi_{b_i}}{2} \prod_{j=1}^{i-1} \frac{1 - \chi_{a_j}}{2}.$$

Then

$$\begin{aligned}
|\mathbf{E}_D[Fg]| &= \left| \mathbf{E}_D \left[F \left(\sum_{i=1}^m \frac{1 + \chi_{a_i}}{2} \frac{1 + \chi_{b_i}}{2} \prod_{j=1}^{i-1} \frac{1 - \chi_{a_j}}{2} \right) \right] \right| \\
&= \left| \mathbf{E}_D \left[F \sum_{i=1}^m h_i \right] \right| = \left| \sum_{i=1}^m \mathbf{E}_D[Fh_i] \right| \\
&\leq \sum_{i=1}^m |\mathbf{E}_D[Fh_i]| \leq m |\mathbf{E}_D[Fh_{i_0}]|
\end{aligned}$$

where $i_0 \in [m]$ is such that $|\mathbf{E}_D[Fh_{i_0}]|$ is maximum. We now rewrite

$$\prod_{j=1}^{i_0-1} \frac{1 - \chi_{a_j}}{2} = \mathbf{E}_{\alpha \in \{0,1\}^{i_0-1}} [(-1)^{|\alpha|} \chi_{A_\alpha}],$$

where $A_\alpha = \sum_{j=1}^{i_0-1} \alpha_j a_j$. The last summation operation is the addition operation over \mathbb{F}_2^n (bitwise exclusive OR). Thus we have

$$\begin{aligned}
h_{i_0} &= \frac{1 + \chi_{a_{i_0}}}{2} \frac{1 + \chi_{b_{i_0}}}{2} \prod_{j=1}^{i_0-1} \frac{1 - \chi_{a_j}}{2} \\
&= \frac{1 + \chi_{a_{i_0}}}{2} \frac{1 + \chi_{b_{i_0}}}{2} \mathbf{E}_{\alpha \in \{0,1\}^{i_0-1}} [(-1)^{|\alpha|} \chi_{A_\alpha}] \\
&= \mathbf{E}_S [(-1)^{|\alpha|} \chi_B]
\end{aligned}$$

where the probability space S is over α uniformly chosen from $\{0,1\}^{i_0-1}$ and β, γ uniformly chosen from $\{0,1\}$ and where $B = A_\alpha \oplus \beta a_{i_0} \oplus \gamma b_{i_0}$. Combining this with an earlier expression we get

$$|\mathbf{E}_D[Fh_{i_0}]| = |\mathbf{E}_D[F \mathbf{E}_S [(-1)^{|\alpha|} \chi_B]]| = |\mathbf{E}_S [(-1)^{|\alpha|} \mathbf{E}_D[F \chi_B]]| \leq \mathbf{E}_S [|\mathbf{E}_D[F \chi_B]|].$$

We may now claim that there is a choice $\alpha_0, \beta_0, \gamma_0$ with $C = A_{\alpha_0} \oplus \beta_0 a_{i_0} \oplus \gamma_0 b_{i_0}$ so that

$$|\mathbf{E}_D[F \chi_C]| \geq |\mathbf{E}_D[Fh_{i_0}]| \geq \frac{|\mathbf{E}_D[Fg]|}{m}.$$

Since g implies f we have the following relation (we remind the reader that F has range $\{-1, +1\}$ and g has range $\{0, 1\}$)

$$\mathbf{E}_D[Fg] = \mathbf{E}_D[g] = \Pr_D[g = 1] \geq \frac{1}{s} \Pr_D[f = 1] = \frac{\mathbf{E}_D[F] + 1}{2s}.$$

Hence, we have

$$|\mathbf{E}_D[F\chi_C]| \geq \frac{\mathbf{E}_D[F] + 1}{2sm}.$$

So either $|\mathbf{E}_D[F\chi_C]| \geq 1/(2sm + 1)$ or $\mathbf{E}_D[F] = \mathbf{E}_D[F\chi_{0_n}] \leq -1/(2sm + 1)$. Noting that by Fact 5.1 $m \leq n^2$, we obtain the desired claim. \square

5.3. Exact Learning Width Two Branching Programs with $\mathcal{O}(1)$ Sinks

The study of the learnability of bounded width branching programs was initiated by Ergün, Ravi Kumar, and Rubinfeld [ERR95]. They show that a restricted variant of width two branching program is efficiently PAC learnable under any distribution and is properly efficiently PAC learnable under the uniform distribution. These results were refined by Bshouty, Tamon, and Wilson [BTW96] who proved that width two branching programs with exactly *two* sinks, i.e., strict width two branching programs, are properly efficiently PAC learnable in the distribution-free model. In this section we show that the class of width two branching programs with a *constant* number of sinks is efficiently exactly learnable using equivalence queries.

We will use the notation k -sink \mathcal{W}_2 to denote the class of width two branching programs with at most k sinks.

THEOREM 5.5. *The class k -sink \mathcal{W}_2 of width two branching programs with k sinks is efficiently exactly learnable using equivalence queries.*

We will prove this theorem by transforming a width two branching program into a special type of decision list, and then prove that the latter type is efficiently exactly learnable with equivalence queries. In our proof we will require the notion of *rank* of a Boolean decision tree that was considered, among others, in the work of Ehrenfeucht and Haussler [EH89].

DEFINITION 5.2. (*Rank of a binary tree*)

Let T be a binary tree. The *rank* of T is defined as the rank of its root node. The

rank of a node is defined inductively as follows. For a non-leaf node v , let v_L and v_R be the left and right child, respectively, of v .

$$\text{rank}(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf} \\ 1 + \text{rank}(v_L) & \text{if } \text{rank}(v_L) = \text{rank}(v_R) \\ \max\{\text{rank}(v_L), \text{rank}(v_R)\} & \text{if } \text{rank}(v_L) \neq \text{rank}(v_R) \end{cases}$$

The notion of *rank* is meant to capture the *bushiness* of a binary tree. Note that a list has rank 1 whereas a complete binary tree of depth d has rank d .

In the first lemma we prove that the set of Boolean functions computable by k -sink width two branching programs is a subclass of Boolean functions computable by rank- k decision trees with parity nodes.

LEMMA 5.6. *For $k \geq 2$, the class of Boolean functions computable by k -sink width two branching programs is a subclass of the class of Boolean functions computable by rank- k decision trees with parity nodes, i.e., \oplus_n -DT.*

Proof We will prove the lemma by induction on k . For $k = 2$, the claim states that strict width two branching program or \mathcal{SW}_2 is a subclass of rank-2 \oplus_n -DT. But this is true by Fact 5.1, since \mathcal{SW}_2 is equivalent to (\oplus_2, \oplus_n) -DL, and an element $f \in (\oplus_2, \oplus_n)$ -DL can be turned into an element of rank-2 \oplus_n -DT (by trivially adding two new nodes for each leaf of f).

Assume that the claim is true for all width two branching programs with at most $k - 1$ sinks, $k \geq 3$. Consider a width two branching program B with k sinks. B can be decomposed into k strict width two branching programs. Let L_1 be the first strict width two branching program and let $b \in \{0, 1\}$ be the label of its sink. By Fact 5.1 L_1 can be converted into a decision list of type (\oplus_2, \oplus_n) -DL. By induction the remaining portion of B can be written as a rank- $(k - 1)$ decision tree T with parity nodes. We attach T to each leaf node of L_1 as follows. For each leaf node l of L_1 , we create an outgoing edge labeled with -1 (or 0) going into T and an outgoing edge

labeled with $+1$ going into the constant function b . Note that the resulting decision tree is of rank k , since the new rank of the leaves of L_1 is now k (the rank of T) and hence the new rank of the internal nodes of L_1 is also k . This completes the inductive argument and hence the lemma. \square

LEMMA 5.7. *The class of Boolean functions computable by rank- k decision trees with parity nodes is a subclass of Boolean functions computable by decision lists whose nodes are parity of monomials, where each monomial is of size at most k . In notation, rank- $k \oplus_n$ - \mathcal{DT} is a subclass of $\oplus \wedge_k$ - \mathcal{DL} .*

Proof The following proof is an adaptation of Blum's argument [Bl92]. Let T be a rank- k decision trees with parity nodes. Because of its rank, T has a leaf node that is of depth at most k (Lemma 1 in [Bl92]); call this leaf node x . Let p be the parent of x and let T_p be the other subtree of p .

Create a node n_x in the decision list that is labeled with a conjunction of at most k parity questions (induced by the root to leaf path ending in x). This conjunction of parities can be converted into a parity of conjunctions where each conjunction is of size at most k .

The next crucial step is that we can remove from T the nodes p and x , and reattach the parent of p directly to T_p . The resulting tree is still a rank- k' decision tree with parity nodes, where $k' \leq k$. If $k' = k$ then we may repeat the same process until the rank reduces to $k - 1$. At that point we appeal to an inductive hypothesis and complete the lemma. \square

Finally we show in the following lemma that $\oplus \wedge_k$ - \mathcal{DL} , and hence k -sink width two branching programs, are efficiently exactly learnable from equivalence queries. The idea is to use the algorithm for learning nested differences of intersection-closed concept classes due to Helmbold, Sloan, and Warmuth [HSW90].

LEMMA 5.8. *The class $\oplus \wedge_k$ - \mathcal{DL} of decision lists whose nodes are parities of monomials of size at most k is efficiently exactly learnable using equivalence queries.*

Proof By the transformation technique of Littlestone [L88], it suffices to prove the claim for the concept class of decision list with parity nodes, i.e., \oplus_n -DL. That is, we can create new variables for each k -subset of the variables and learn the target concept as a new function over at most $n + n^k$ variables.

To exactly learn \oplus_n -DL we will show that we can express any element of \oplus_n -DL as a nested difference of vector spaces over \mathbb{F}_2^n . Since vector subspaces are closed under intersection, we can appeal to an algorithm for exactly learning nested differences of intersection-closed concept classes due to Helmbold, Sloan, and Warmuth [HSW90].

Assume that the target concept $f \in \oplus_n$ -DL is given by

$$f = [(\chi_{a_1}, b_1), (\chi_{a_2}, b_2), \dots, (\chi_{a_k}, b_k)],$$

where $a_1, a_2, \dots, a_k \in \{0, 1\}^n$ and $b_1, b_2, \dots, b_k \in \{0, 1\}$. We *compress* consecutive leaves that output the same value. This is permissible since consecutive parity tests can be turned into a membership test for a subspace L that halts at the leaf if the test failed and proceeds to the next node if the test is passed. When the compression process is finished, we will end up with a decision list whose internal nodes are labeled with membership tests for subspaces. So assume that we have

$$f = [(L_1, c_1), (L_2, c_2), \dots, (L_t, c_t)],$$

where $t \leq k$, the L_i 's denote subspaces, and $c_1, \dots, c_t \in \{0, 1\}$ are alternating in value. Again we remind the reader that the value c_1 will be output if the example does not belong to the subspace L_1 , the value c_2 will be output if the example belonged to L_1 but not to L_2 , and so on. Assume without loss of generality that $c_1 = 0$ (the case when $c_1 = 1$ can be treated as easily). Then we have the following form

$$f = L_1 - (L_2 - (L_3 - (\dots))).$$

This completes the proof. \square

5.4. Exact Learning *Permutation* Branching Programs

In the next sections we study the problem of learning bounded-width *permutation* branching programs. First we define the notion of a permutation branching program introduced by Barrington [Bar89].

DEFINITION 5.3. (*Permutation branching program*)

Let P_w be a group of permutations on w elements. A *permutation branching program* (PBP) of width w and length l is given by a sequence of instructions $(j(i), g_i, h_i)$, for $0 \leq i < l$, where $1 \leq j(i) \leq w$ and $g_i, h_i \in P_w$. A permutation branching program has on each level i , w nodes $v_{i,1}, \dots, v_{i,w}$. On level i we realize $\sigma(x) = g_i$ if $x_{j(i)} = 0$ and $\sigma(x) = h_i$ if $x_{j(i)} = 1$. The branching program computes

$$\sigma(x) = \sigma_{l-1}(x)\sigma_{l-2}(x)\dots\sigma_0(x) \in P_w$$

on input $x \in \{0,1\}^n$. The permutation branching program computes a Boolean function f on n inputs via τ if $\sigma(x) = id$, for $x \in f^{-1}(0)$, and $\sigma(x) = \tau \neq id$, for $x \in f^{-1}(1)$.

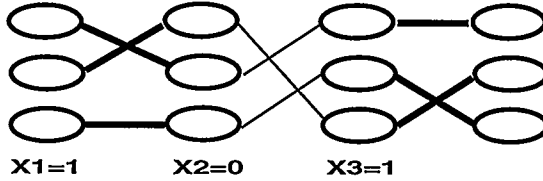


FIGURE 5.3. Example of S_3 -PBP computing the identity permutation

Barrington proved that any Boolean function computable by an NC^1 circuit, i.e., a Boolean circuit of polynomial size and logarithmic depth, is also computable by a width *five* permutation branching program. In this section we will show that permutation branching programs of width *three* and *four* are efficiently learnable in the exact identification model with equivalence and membership queries. So in fact we

come quite close to the non-learnability barrier given by Angluin and Kharitonov [AK95] for width five branching programs.

The technique that we will use comes from automata theory. In particular we will use the representation class of *multiplicity automata* over finite fields. The novel use of this representation class of automata was recently introduced by Bergadano, Catalano, and Varricchio [BV94, BCV96]. For more information on the applications and limitations of this method we refer the reader to [BBBKV96].

5.4.1. Multiplicity Automata. We describe relevant definitions from the theory of multiplicity automata and state a recent result on the learnability of multiplicity automata [BV94, BCV96, BBBKV96]. We will also prove a lemma that describes a non-trivial closure operation on this class of automata.

DEFINITION 5.4. (*Multiplicity automata*)

Let \mathcal{K} be a field. A nondeterministic automaton M with multiplicity is a five-tuple $M(\Sigma, Q, E, I, F)$ where Σ is a finite alphabet, Q is the finite set of states, $I, F : Q \rightarrow \mathcal{K}$ are two mappings associated with the initial and final states, respectively, and

$$E : Q \times \Sigma \times Q \rightarrow \mathcal{K}$$

is a map that associates a multiplicity to each edge of M . We will sometimes call M a \mathcal{K} -automaton for brevity. The *size* of M is the number of states, i.e., $|Q|$.

Let $x = (x_1, \dots, x_n) \in \Sigma^*$. A path for x is a sequence

$$p = (p_1, x_1, p_2), (p_2, x_2, p_3), \dots, (p_n, x_n, p_{n+1}),$$

where $p_i \in Q$, for all $0 \leq i \leq n + 1$. Let $Path_M(x)$ denote the set of all paths for x . The *behavior* of M is a mapping $S_M : \Sigma^* \rightarrow \mathcal{K}$ defined as follows: for each $x = (x_1, \dots, x_n) \in \Sigma^*$

$$S_M(x) = \sum_{p \in Path_M(x)} I(p_1) \left(\prod_{i=1}^n E(p_i, x_i, p_{i+1}) \right) F(p_{n+1}).$$

For a Boolean function over $f : \Sigma^* \rightarrow \{0, 1\}$, we say that a multiplicity automaton M computes f if for all $x \in \Sigma^*$ we have $S_M(x) = f(x)$. Alternatively one may think of f as a characteristic function of a language over Σ^* .

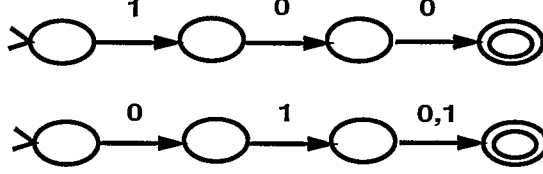


FIGURE 5.4. Example of \mathbb{F}_2 -automata for DNF $x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2$

In the following we will describe several operations on multiplicity automata, namely the Hadamard product, union, and scalar multiplication. In effect we will argue that multiplicity automata are closed under these operations.

DEFINITION 5.5. (*Closure operations*)

Let \mathcal{K} be a field. Let $M_1(\Sigma, Q_1, E_1, I_1, F_1)$ and $M_2(\Sigma, Q_2, E_2, I_2, F_2)$ be two \mathcal{K} -automata.

- (1) The *Hadamard product* of M_1 and M_2 , denoted by $M_1 \odot M_2$, is a \mathcal{K} -automaton $M(\Sigma, Q, E, I, F)$ where $Q = Q_1 \times Q_2$, and I, F, E are defined as $I(q_1, q_2) = I_1(q_1)I_2(q_2)$, $F(q_1, q_2) = F_1(q_1)F_2(q_2)$, and

$$E((q, p), a, (q', p')) = E_1(q, a, q')E_2(p, a, p').$$

Note that M has $|Q_1||Q_2|$ states. Moreover M satisfies

$$S_M(x) = S_{M_1}(x)S_{M_2}(x).$$

- (2) Assume that Q_1 and Q_2 are two disjoint sets of states. The *union* of M_1 and M_2 , denoted simply by $M_1 \cup M_2$, is a \mathcal{K} -automaton where $M(\Sigma, Q, E, I, F)$ where $Q = Q_1 \cup Q_2$, and I, F, E are defined as $I(q) = I_1(q)[q \in Q_1] + I_2(q)[q \in Q_2]$

$Q_2]$, $F(q) = F_1(q)[q \in Q_1] + F_2(q)[q \in Q_2]$, and

$$E(q, a, p) = E_1(q, a, p)[q, p \in Q_1] + E_2(q, a, p)[q, p \in Q_2].$$

Note that M has $|Q_1| + |Q_2|$ states. Moreover M satisfies

$$S_M(x) = S_{M_1}(x) + S_{M_2}(x).$$

- (3) For any $\lambda \in \mathcal{K}$, the automaton λM_1 is defined to be the \mathcal{K} -automaton $M(\Sigma, Q, E, I, F)$ where $Q = Q_1$, $I = \lambda I_1$, $F = F_1$, and $E = E_1$. Note that $|Q| = |Q_1|$ and that M satisfies

$$S_M(x) = \lambda S_{M_1}(x).$$

Next we prove a result that yields another closure operation, namely constant Boolean combinations of multiplicity automata.

LEMMA 5.9. *Let p be a fixed prime. Let g_1, g_2, \dots, g_k be Boolean functions that can be computed by \mathbb{F}_p -automata of size at most s . Then for any Boolean function f on k inputs, $f(g_1, g_2, \dots, g_k)$ can be computed by a \mathbb{F}_p -automaton with at most $2^k s^k$ states.*

Proof The function $f(g_1, g_2, \dots, g_k)$ can be written as

$$\sum_{\alpha \in \mathbb{Z}^k} \lambda_\alpha \prod_{i=1}^k g_i^{\alpha_i},$$

for some $\lambda_\alpha \in \mathbb{F}_p$. Since g_1, g_2, \dots, g_k take values $\{0, 1\}$, we may assume that $\alpha_1, \dots, \alpha_k \in \{0, 1\}$. Therefore we can write

$$f = \sum_{\beta \in \{0, 1\}^k} \lambda_\beta \prod_{i=1}^k g_i^{\beta_i},$$

for some $\lambda_\beta \in \mathbb{F}_p$. By the properties of Hadamard product $\prod_{i=1}^k g_i^{\beta_i}$ has a multiplicity \mathbb{F}_p -automaton of size at most s^k . The multiplication with λ_β admits another \mathbb{F}_p -automaton of size s^k . Then summing 2^k of such \mathbb{F}_p -automata gives an automaton with size at most $2^k s^k$. \square

A *multiplicity oracle* $MUL_M()$ for a \mathcal{K} -automaton M is an oracle that receives as input a string $x \in \Sigma^*$ and returns $S_M(x)$. For some concept classes, the multiplicity oracle reduces to the membership oracle. The following result was established in [BV94] and was further studied in [BBBKV96].

THEOREM 5.10. [BV94] *For any field \mathcal{K} , the class of behavior mappings $S_M : \Sigma^* \rightarrow \mathcal{K}$, for any \mathcal{K} -automaton $M(\Sigma, Q, E, I, F)$, is efficiently exactly learnable from equivalence and multiplicity queries. The learning complexity is polynomial in $|\Sigma|$, $|Q|$, the size of the longest counterexample seen, and the size of the field \mathcal{K} .*

5.4.2. Transformation to Small-depth Circuits. In this section we will show that small-width permutation branching programs are efficiently exactly learnable from equivalence and membership queries. Our technique for proving learnability is to use known connections between bounded-width permutation branching programs and small-depth Boolean circuits with modular and threshold gates and to prove that the latter classes are efficiently exactly learnable.

We introduce some notation for describing small depth Boolean circuits with modular and threshold gates. A mod_p gate over n Boolean inputs x_1, \dots, x_n is defined as follows:

$$mod_p(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^n x_i \equiv 0 \pmod{p} \\ 0 & \text{otherwise} \end{cases}$$

A *weighted threshold* gate with integer weights $\vec{a} = (a_1, a_2, \dots, a_n) \in \mathbb{Z}^n$ and a threshold $b \in \mathbb{Z}$ over n Boolean inputs, denoted by $TH_{\vec{a},b}^n$, is defined as follows:

$$TH_{\vec{a},b}^n(x) = \begin{cases} 1 & \text{if } a_1x_1 + a_2x_2 + \dots + a_nx_n \geq b \\ 0 & \text{otherwise} \end{cases}$$

That is, $TH_{\vec{a},b}^n(x) = [\sum_{i=1}^n a_i x_i \geq b]$. Note that $TH_k^n = TH_{1_n,k}^n$. The class of Boolean functions computable by a threshold gate with integer weights is denoted by \widehat{LT}_1 . We define the weight or size $w(TH_{\vec{a},b}^n)$ of a threshold function $TH_{\vec{a},b}^n$ to be $|b| + \sum_{i=1}^n |a_i|$. The *representation size* of a threshold function f is $w(f)$.

The class $\text{mod}_{\mathbb{Z}}$ is the class of mod_q functions, for all integers $q \in \mathbb{Z}$. For notational simplicity, we will adopt the following convention. For two classes of gates or functions A and B , we denote A - B circuits to be the class of Boolean functions computable by a depth two Boolean circuit with a gate from A at the top and gates from B at the bottom level. For example, a mod_p - $\text{mod}_{\mathbb{Z}}$ circuit is a depth two Boolean circuit that has a mod_p gate at the top and arbitrary mod_q gates, $q \in \mathbb{Z}$, at the bottom level. Note that we allow mod_q gates with possibly different q 's at the bottom level.

THEOREM 5.11. *For any fixed prime p , the class of mod_p - $\text{mod}_{\mathbb{Z}}$ circuits is efficiently exactly learnable using equivalence and membership queries.*

Proof It suffices to exhibit a multiplicity automaton for the target mod_p - $\text{mod}_{\mathbb{Z}}$ circuit. Let $\mathcal{K} = \mathbb{F}_p$. We construct for each mod_q gate, $q \in \mathbb{Z}$, a \mathcal{K} -automaton that accepts it. Next we combine these automata into a single \mathcal{K} -automaton by taking the union of all the automata for the mod_q gates. By Fermat's Little theorem, the Hadamard product of M with itself $p - 1$ times computes the target mod_p - $\text{mod}_{\mathbb{Z}}$ circuit. \square

Next we will show that Boolean functions computable by threshold gates with integer weights can be represented by a multiplicity automaton.

LEMMA 5.12. *The class \widehat{LT}_1 admits a representation as an \mathbb{F}_p -automaton, for any prime p .*

Proof Suppose that $f(x) = [\sum_{i=1}^n a_i x_i \geq b]$ where $\vec{a} \in \mathbb{Z}^n$ and $b \in \mathbb{Z}$. Let $A = |a_1| + |a_2| + \dots + |a_n| + 1$. We construct the automaton M with state set $Q = \{q_{i,j} : i \in [-A, A], j \in [n+1]\}$. The edge set contains only the following edges (assigned a multiplicity of 1, while other edges are assigned 0 multiplicity):

$$(q_{i,j}, 0, q_{i,j+1}), (q_{i,j}, 1, q_{i+a_j, j+1}) \in E$$

for all $i \in [A]$ and $j \in [n]$. Set $I = \{q_{0,0}\}$ and $F = \{q_{i,n+1} \mid i \geq b\}$. \square

Using the above lemma we can claim (as before) that the class of Boolean functions computable by mod_p or a constant Boolean combination of threshold functions is efficiently exactly learnable.

COROLLARY 5.13. *For any fixed prime p , the class $\text{mod}_p\text{-}\widehat{LT}_1$ is efficiently exactly learnable using equivalence and membership queries.*

We remark that proving the learnability of the class of Boolean functions computable by $\widehat{LT}_1\text{-mod}_p$ circuits will prove the learnability of DNF formulas. This is because Krause and Pudlák [KP94] (see also [J94]) have proved that any DNF formula can be expressed as a majority of parities.

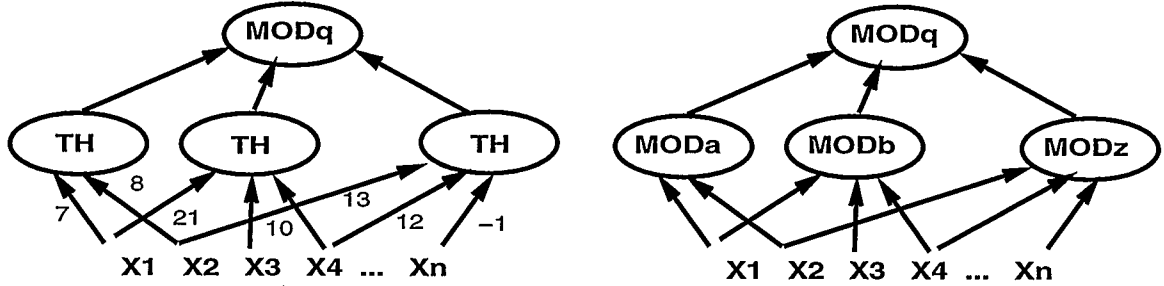


FIGURE 5.5. Examples of $\text{mod}_q\text{-}\widehat{LT}_1$ and $\text{mod}_q\text{-mod}_Z$ circuits

Let S_3 be the symmetric group on $[3]$ and A_4 be the alternating group on $[4]$. We will exploit some known circuit characterizations of permutation branching programs to prove the learnability of S_3 and A_4 permutation branching programs. The following fact about S_3 permutation branching programs was proved by Barrington in his thesis [B86].

FACT 5.14. *The class of Boolean functions computable by S_3 permutation branching programs is equivalent to the class of Boolean functions computable by $\text{mod}_3\text{-mod}_2$ circuits.*

THEOREM 5.15. *The class of Boolean functions computable by S_3 permutation branching programs is efficiently exactly learnable using equivalence and membership queries.*

Proof Follows from Theorem 5.11. \square

In [B], it is mentioned that A_4 -PBP is equivalent to a $(\text{mod}_2, \text{mod}_2)$ - mod_3 circuit, i.e., a depth “two” circuit consisting of mod_3 gates at the bottom level coming into two mod_2 gates at the second level. The outputs of the two mod_2 gates are then combined using an AND gate.

FACT 5.16. *The class of Boolean functions computable by A_4 permutation branching programs is equivalent to the class of Boolean functions computable by \wedge -($\text{mod}_2, \text{mod}_2$)- mod_3 circuits.*

To prove that A_4 permutation branching programs are efficiently exactly learnable using equivalence and membership queries, we prove the following general result.

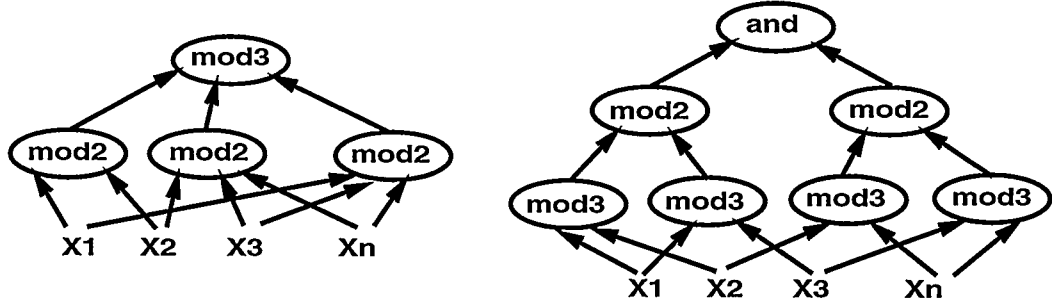


FIGURE 5.6. Barrington's circuit characterization of S_3, A_4 -PBPs

THEOREM 5.17. *Let g_1, g_2, \dots, g_k be Boolean functions that can be computed by a multiplicity \mathbb{F}_p -automata of size at most s . Then for any Boolean function f on k inputs, $f(g_1, g_2, \dots, g_k)$ is exactly learnable using equivalence and membership queries in time $s^{O(k)}$. Thus, learning is efficient if $k \in O(1)$.*

Proof Follows from Lemma 5.9 and Theorem 5.10. \square

Learning Boolean Functions with the \mathcal{NP} Oracle

In the PAC learning model, it is a known result that if $\mathcal{P} = \mathcal{NP}$ then there is an efficient learning algorithm for the class of DNF formulas as well as for the class of Boolean circuits. This is because in the PAC model to guarantee learnability, among others, it suffices to be *consistent*, i.e., that the following problem can be solved efficiently.

Consistency Problem for Class C

Input: A list $\{(a_1, b_1), \dots, (a_m, b_m)\}$ so that $a_i \in \{0, 1\}^n$ and $b_i \in \{0, 1\}$, for each $i \in [m]$.

Output: A representation $f \in C$ so that for all pairs (a_i, b_i) , $f(a_i) = b_i$.

If no such f exists then output *No*.

Informally, a consistent $f \in C$ is an approximation with accuracy ϵ and confidence δ for a sample of size $\frac{1}{\epsilon} \ln \frac{|C|}{\delta}$. So, if $\ln |C| \in n^{\mathcal{O}(1)}$ then this yields a polynomial-time PAC learning algorithm for C , assuming that the consistency problem can be solved in polynomial-time in the input size.

Unless otherwise specified we will assume that the concept class C is polynomial-time evaluable, i.e., for any concept f in C and any assignment a , there is a polynomial-time algorithm for computing $f(a)$. Note that the above consistency problem is solvable by a nondeterministic polynomial time Turing machine since after guessing a representation $f \in C$, it is easy to verify that the representation is consistent with the input list.

This result is not that obvious in the exact learning model because of the *adversarial* nature of the counterexamples as well as the exact identification criterion, i.e., *being consistent* is not always a good strategy in the exact learning model.

The main question asked in this chapter is how helpful can an \mathcal{NP} oracle be to exactly learn polynomial size Boolean circuits and DNF formulas. The answer turns out to be positive: we can learn exactly Boolean circuits and DNF formulas with the help of an \mathcal{NP} oracle and equivalence queries. Moreover any Boolean function that is exactly learnable with polynomially many membership queries (with no bounds on the time complexity) is exactly learnable with the help of an \mathcal{NP} oracle and membership queries in polynomial time.

The idea used to show these results is a combination of a standard majority-vote algorithm called the Halving algorithm and a method of randomly generating combinatorial structures due to Jerrum, Valiant, and Vazirani [JVV86].

A consequence of the first result is that if the class of polynomial size DNF formulas is not exactly learnable then P is not equal to \mathcal{NP} . The same statement holds for polynomial size Boolean circuits. More surprising is the consequence observed by Watanabe: if each language in \mathcal{NP} is solvable using a family of polynomial size Boolean circuits then the polynomial-time hierarchy collapses to $\mathcal{ZPP}^{\mathcal{NP}}$.

6.1. Uniform Generation of Polynomial-time Structures

In this section we motivate the problem of uniformly generating elements from some combinatorial structure. Let R be a polynomial-time computable relation, i.e., suppose there is a polynomial-time algorithm that decides if xRy , for any given x and y . There are two natural questions that have been asked in complexity theory:

- *Existence:* Given x , does there exist a y so that xRy ?
- *Counting:* Given x , how many y 's are there so that xRy ?

A famous example of the existence-type question is the Boolean formulas *satisfiability* question. Given a Boolean formula in conjunctive normal form ϕ , does there exist a

satisfying assignment for ϕ ? The famous example for the counting-type question is the problem of counting the number of satisfying assignments for a Boolean formula.

Jerrum, Valiant, and Vazirani [JVV86] introduced the following intermediate problem:

- *Uniform generation:* Given x , pick a y uniformly at random so that xRy .

An example of this is, given a Boolean formulas ϕ , to find a random satisfying assignment for ϕ . The authors of [JVV86] came up with a beautiful method for *approximately* uniformly sampling combinatorial structures using a probabilistic oracle Turing machine that has access to an \mathcal{NP} -oracle.

DEFINITION 6.1. (*Approximately Uniform Distribution*)

Let D be a probability distribution on a discrete probability space Ω and $S \subseteq \Omega$. Then D is *uniform on S* if, for all $x \in \Omega$:

$$D(x) = \begin{cases} 1/|S| & \text{if } x \in S \\ 0 & \text{otherwise} \end{cases}$$

Also, for $\epsilon \in (0, 1]$, D is *approximately uniform on S with tolerance ϵ* if, for all $x \in \Omega$:

$$D(x) = \begin{cases} c & \text{where } (1 + \epsilon)^{-1} \frac{1}{|S|} \leq c \leq (1 + \epsilon) \frac{1}{|S|} \text{ if } x \in S \\ 0 & \text{otherwise} \end{cases}$$

The main result from [JVV86] that we need is the following theorem, which we have stated in an alternative form.

THEOREM 6.1. *Let $\{C_I\}_I$ be an indexed family of sets. Suppose that there is an algorithm that, on input f and I , determines whether or not $f \in C_I$ in time polynomial in $|I|$.*

Then there exists a probabilistic oracle algorithm that uses an \mathcal{NP} oracle and, on input I and ϵ , runs in time polynomial in $|I|$, $\log \frac{1}{\epsilon}$, and outputs f according to a distribution that is approximately uniform on C_I with tolerance ϵ .

We remark that technical arguments must be made regarding the randomness required by the probabilistic Turing machine in Theorem 6.1, i.e., assuming only the presence of fair random coins, is sometimes not enough. Whenever necessary we will assume an extended model for randomized algorithms as described by Sinclair in his thesis [Si].

In one particular application of Theorem 6.1 we need to generate a random DNF formula f on n inputs that is consistent with a set I of labeled examples, where $I \subseteq \{0,1\}^n \times \{0,1\}$. It is easy to see that there is a polynomial-time algorithm that can decide if f is consistent with I ; simply scan the list I and check that f agrees with all labeled examples. This is true also for the case when f is a Boolean circuit. So Theorem 6.1 provides a method of sampling consistent concepts (DNF or Boolean circuits) according to a distribution that is almost uniform. We will then show that this combined with a method due to Kannan [K93] is enough to yield an exact learning algorithm using equivalence queries. But the key algorithmic idea will rest on a generalization of the Halving algorithm which we discuss in the following section.

6.2. The Halving Algorithm Revisited

The *Halving algorithm* [Ang88, L88] is an exact learning algorithm using equivalence queries that can be applied to learn any concept class C by asking at most $\log |C|$ equivalence queries. The idea of this algorithm is simple: it maintains at each step the collection of concepts in C that are *consistent* with the counterexamples received so far. It then asks an equivalence query that is the majority of all these remaining concepts. Any counterexample to this equivalence query will eliminate at least half of the consistent concepts. In the end there will be only one concept remaining and it will be consistent with all the counterexamples.

Following Littlestone [L88] we introduce some necessary notation to describe the Halving algorithm that will also be useful in later sections.

DEFINITION 6.2. (*Consistent concepts C_I*)

For a concept class C over $\{0,1\}^n$, $x \in \{0,1\}^n$, and $b \in \{0,1\}$ define $C_{(x,b)} = \{f \in C \mid f(x) = b\}$. For a set of labeled examples $I \subseteq \{0,1\}^n \times \{0,1\}$, define

$$C_I = \{f \in C : f(x) = b, \text{ for all } (x, b) \in I\}.$$

Given a concept class C , the majority of all concepts in C , in notation $MAJ(C)$ is defined to be the following function.

$$MAJ(C)(x) = \begin{cases} 1 & \text{if } |\{f \in C : f(x) = 1\}| \geq |C|/2 \\ 0 & \text{otherwise} \end{cases}$$

Although the Halving algorithm is a general-purpose exact learning algorithm, it has several undesirable features. First, it is not guaranteed to be a proper learning algorithm, i.e., the concept class may not be closed under taking majorities. Second, the size of its hypothesis to the equivalence query can be quite large, e.g., initially it is the majority of all concepts in the target class.

In a very nice paper Kannan [K93] proposed a randomized version of the Halving algorithm. His algorithm extends the Halving algorithm in two ways.

- Instead of discarding at least half of the concepts at every step, it discards some, perhaps, smaller fraction $\delta > 0$.
- Instead of using the majority vote on all remaining concepts, it uses a majority vote of a small random subcollection of the concepts.

Kannan proved that at each step, there is a small subcollection of the consistent concepts whose majority vote behaves almost as well as the majority vote on all concepts. In the following we will formally elaborate details of Kannan's algorithm along with some extensions.

DEFINITION 6.3. (*A δ -good hypothesis*)

Let C be a concept class and let $\delta \in [0, \frac{1}{2}]$. A hypothesis h is δ -good for C if any counterexample to an equivalence query of h eliminates at least a δ fraction of elements

from C . Note that the majority hypothesis is 0.5-good for any concept class.

DEFINITION 6.4. Let C be a concept class, let x be an assignment from $\{0,1\}^n$, and let $b \in \{0,1\}$ be a bit value. Then we define

$$\gamma_{(x,b)}^C = \frac{|C_{(x,b)}|}{|C|}, \quad \gamma_x^C = \min(\gamma_{(x,0)}^C, \gamma_{(x,1)}^C).$$

Thus C_I is the set of concepts in C that properly classify all examples in the labeled example set I and $\gamma_{(x,b)}^C$ is the fraction of C that classifies example x with label b .

We can define the δ -Halving algorithm to be a variant of the Halving algorithm that repeatedly asks an equivalence query that is a δ -good hypothesis for the set of concepts not discarded before. Starting with concept class C , after i queries the number of concepts left is at most $(1 - \delta)^i |C|$. So at most

$$\ln |C| / \ln \frac{1}{1 - \delta} \leq \frac{1}{\delta} \ln |C|$$

equivalence queries are required to isolate the target concept.

The next improvement introduced by Kannan, and further generalized in [BCKT94, BCG⁺], is to use the notion of *amplifiers* in obtaining a δ -good hypothesis. The concept of amplification was studied by Valiant and Boppana [V84a, Bop89]. Amplification was also used in learning by Goldman, Kearns, and Schapire (see [S]).

DEFINITION 6.5. (*Amplifier*)

Let $0 \leq p' < p < q < q' \leq 1$. A (Boolean) function $G(y_1, \dots, y_m)$ is a $(p, q) \rightarrow (p', q')$ *amplifier* if:

- (1) When y_1, \dots, y_m are each independently set to 1 with probability *at least* q , $\Pr[G(y_1, \dots, y_m) = 1] \geq q'$;
- (2) When y_1, \dots, y_m are each independently set to 1 with probability *at most* p , $\Pr[G(y_1, \dots, y_m) = 1] \leq p'$.

The following lemma is an improvement on Kannan's observations on the connection between amplification and equivalence queries.

LEMMA 6.2. *Let $G(y_1, \dots, y_m)$ be a $(\delta, 1-\delta) \rightarrow (2^{-2n}, 1-2^{-2n})$ amplifier. Let C be a concept class over $\{0, 1\}^n$ and f_1, \dots, f_m be functions selected from C independently and uniformly at random. Then, with probability at least $1 - 2^{-n}$, $G(f_1, \dots, f_m)$ is δ -good for C .*

Proof Note that, if $\delta \leq \gamma_{(x,1)}^C \leq 1-\delta$ then, if x is returned as a counterexample to any equivalence query, a δ -fraction of the elements of C are guaranteed to be eliminated.

Now, let x be any value for which $\gamma_{(x,1)}^C < \delta$. Then, if x is returned as a counterexample to some f_i for which $f_i(x) = 1$, less than a δ -fraction of the elements of C will be eliminated; otherwise, more than a δ -fraction. For a $f_i \in C$ chosen uniformly at random, $\Pr[f_i(x) = 1] < \delta$. Therefore, since $G(y_1, \dots, y_m)$ is a $(\delta, 1-\delta) \rightarrow (2^{-2n}, 1-2^{-2n})$ amplifier,

$$\Pr[G(f_1, \dots, f_m)(x) = 1] < 2^{-2n}.$$

Thus, the probability that less than a δ -fraction of the elements of C are eliminated when x is returned as a counterexample is $< 2^{-2n}$.

A similar argument applies for any x such that $\gamma_{(x,1)}^C > 1 - \delta$.

Therefore, the probability that there exists an $x \in \{0, 1\}^n$ which, when returned as a counterexample to the equivalence query $G(f_1, \dots, f_m)$ eliminates less than a δ -fraction of the elements of C , for uniformly and independently chosen f_1, \dots, f_m , is less than $2^n \cdot 2^{-2n} = 2^{-n}$. \square

The next lemma describes two potential amplifiers that are useful in conjunction with the δ -Halving algorithm.

LEMMA 6.3. [K93, BCG⁺]

- (1) *The majority function $MAJ(y_1, \dots, y_{48n})$ is a $(\frac{1}{4}, \frac{3}{4}) \rightarrow (2^{-2n}, 1-2^{-2n})$ amplifier.*
- (2) *Define $A(y_1, \dots, y_m)$ as a $(2n/\log n)$ -ary \wedge of $(2n/\log n)$ -ary \vee s of distinct variables. (Thus, the number of inputs to the formula is $m = \frac{4n^2}{\log^2 n}$.) Then $A(y_1, \dots, y_m)$ is a $(\frac{1}{n^2}, 1 - \frac{1}{n^2}) \rightarrow (2^{-2n}, 1-2^{-2n})$ amplifier.*

Proof We will use Chernoff bounds, Fact 2.6, to prove the above lemma.

To prove the first statement note that if each of $48n$ random variables are chosen with each $p_i = 1/4$, then the probability that the sum of the random variables exceeds $24n$ is $F^+(12n, 1) = (e/4)^{12n} < 2^{-2n}$. Similarly, if $48n$ random variables are chosen each with $p_i = 3/4$, then the probability that the sum of the random variables falls below $24n$ is given by $F^-(36n, 1/3) = e^{-2n} < 2^{-2n}$.

To prove the second statement note that if $p_i \leq \frac{1}{n^2}$ then the probability that any particular V-gate will compute a 1 is at most $(1/n^2) \cdot (2n/\log n) = 2/n \log n$. The probability that all of the V-gates will compute a 1 (and hence the circuit will compute a 1) is at most $(2/n \log n)^{2n/\log n}$ which is at most 2^{-2n} . If $p_i \geq 1 - \frac{1}{n^2}$ the probability that a particular V-gate will not compute a 1 is at most $(1/n^2)^{2n/\log n} = 2^{-4n}$, and the probability that some V-gate will not compute a 1 is at most $(2n/\log n)2^{-4n}$ which is no more than 2^{-2n} . \square

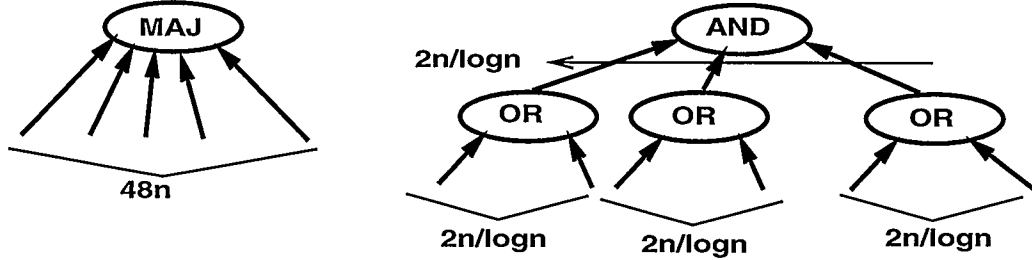


FIGURE 6.1. Examples of amplifiers

Next we state the well-known fact that the majority function on n inputs can be computed by a Boolean circuit of size $\mathcal{O}(n \log n)$.

FACT 6.4. *The majority function $MAJ(x_1, x_2, \dots, x_n)$ is computable by a Boolean circuit of size $\mathcal{O}(n \log n)$.*

Proof The idea is to successively add the bits x_1, x_2, \dots, x_n using adders for numbers that are $\mathcal{O}(\log n)$ bits long. It is known that there are addition circuits of linear size

and constant depth [W87]. Once we obtain the $c \log n$ -bit number $y = x_1 + x_2 + \dots + x_n$, the majority function of x_1, \dots, x_n is the OR of the top half highest order bits of y .

□

By noting that $MAJ(y_1, \dots, y_{48n})$ is computable by a Boolean circuit of size $\mathcal{O}(n \log n)$, and $A(f_1, \dots, f_m)$ is a depth-3 \wedge - \vee - \wedge formula when f_1, \dots, f_m are DNF formulas, we obtain the following result.

COROLLARY 6.5. *The following learning tasks can be accomplished with polynomially many equivalence queries:*

- (1) *Learning DNF formulas of size s using equivalence queries that are depth-3 \wedge - \vee - \wedge formulas of size $\mathcal{O}(sn^2 / \log^2 n)$.*
- (2) *Learning Boolean circuits of size s using equivalence queries that are Boolean circuits of size $\mathcal{O}(sn + n \log n)$.*

δ -Halving algorithm

let G_δ be a $(\delta, 1 - \delta) \rightarrow (2^{-2n}, 1 - 2^{-2n})$ amplifier.

input: concept class \mathcal{C} .

- (1) pick a *small* uniform sample \mathcal{A} from \mathcal{C} .
- (2) ask equivalence query with $G_\delta(\mathcal{A})$.
- (3) if the answer is *yes* then **halt**.
- (4) otherwise let (x, b) be the counterexample.
- (5) update $\mathcal{C} = \mathcal{C}_{(x,b)}$ and go to step (1).

FIGURE 6.2. The δ -Halving algorithm using amplifiers.

The computational difficulty in implementing the above learning algorithm is in uniformly selecting the formulas from \mathcal{C} , which is, in general, an exponentially large

set. This is the point where we called upon the uniform generation result of Jerrum, Valiant, and Vazirani.

6.3. Exact Learning with the Equivalence and \mathcal{NP} Oracles

In this section we show that there exists a randomized algorithm that learns Boolean circuits and DNF formulas from equivalence queries and an \mathcal{NP} oracle. We cannot apply Theorem 6.1 directly to select f_1, \dots, f_m for Lemma 6.2, because the sampling provided by Theorem 6.1 is not exactly uniform. The following lemmas imply that approximately uniform sampling suffices.

LEMMA 6.6. *Let C be a concept class over $\{0,1\}^n$, and let U be an approximately uniform generator for C with tolerance ϵ . Let f be the random function output by U . If $x \in \{0,1\}^n$ with $\gamma_{(x,b)}^C \leq \delta$ then the probability that $f(x) = b$ is at most $\delta(1 + \epsilon)$ for any $\delta \geq 0$ and $b \in \{0,1\}$.*

Proof Suppose $x \in \{0,1\}^n$ such that $\gamma_{(x,b)}^C \leq \delta$ for some δ and for some b . Then if a function f is chosen uniformly at random from C , $\Pr[f(x) = b] \leq \delta$. Since U can at most oversample the functions f such that $f(x) = b$ by a factor of $(1 + \epsilon)$, if f is the output of U , $\Pr[f(x) = b]$ is bounded by $\delta(1 + \epsilon)$. \square

LEMMA 6.7. *Let $G(y_1, \dots, y_m)$ be a $(\delta, 1 - \delta) \rightarrow (2^{-2n}, 1 - 2^{-2n})$ amplifier. Let C be a concept class and U an approximately uniform generator for C with tolerance ϵ . If f_1, \dots, f_m are selected independently using U then, with probability at least $1 - 2^{-n}$, $G(f_1, \dots, f_m)$ is $\delta/(1 + \epsilon)$ -good for C .*

Proof The proof is immediate from Lemma 6.6. For any $x \in \{0,1\}^n$ that has $\gamma_{(x,0)}^C \leq \delta/(1 + \epsilon)$, $\Pr[f_i(x) = 0] \leq \delta$. Thus the probability that $G(f_1, \dots, f_m)(x) = 0$ is at most 2^{-2n} . A similar analysis holds when the ‘0’ is replaced by a ‘1’. Thus applying the argument used to prove Lemma 6.2 the result follows. \square

THEOREM 6.8. *The following learning tasks can be accomplished with high probability by probabilistic polynomial-time algorithms that have access to an \mathcal{NP} oracle and that make polynomially many queries:*

- (1) *Learning DNF formulas of size s using equivalence queries that are depth-3 $\wedge\text{-}\vee\text{-}\wedge$ formulas of size $\mathcal{O}(sn^2/\log^2 n)$.*
- (2) *Learning Boolean circuits of size s using equivalence queries that are Boolean circuits of size $\mathcal{O}(sn + n \log n)$.*

Proof We use the amplifiers provided by Lemma 6.3 and apply them to the output of the generator of Theorem 6.1, with (say) $\epsilon = 1$. By Lemma 6.7, an equivalence query that is $\frac{\log n}{2n^2}$ -good (for the first part) and $\frac{1}{8}$ -good (for the second part) is generated with probability $1 - 2^{-n}$. We note that the probability that every equivalence query asked by the algorithm is *good* is at least $1 - n^{\mathcal{O}(1)}/2^n$. So with the same probability, the learning algorithm discovers the target concept within polynomially many steps.

□

We remark that Angluin [Ang90] has shown that DNF formulas are not properly exactly learnable using only equivalence queries regardless of the computational power of the learning algorithm. Thus the use of the above hypothesis that is a depth three formula is, in some sense, the best we can do.

6.4. Exact Learning with the Membership and \mathcal{NP} Oracles

In this section we consider scenarios where the learner can only use membership queries and has an access to an \mathcal{NP} oracle. Exact learning with only membership queries is also known as black box interpolation in some papers. We present the following two main results.

- If a concept class is exactly learnable with polynomially many membership queries then it is exactly learnable with high probability in expected polynomial time by an algorithm that uses membership queries and has access to an \mathcal{NP} oracle.

- If a concept class is exactly learnable with equivalence and membership queries then it is exactly learnable with membership queries and access to an \mathcal{NP} oracle.

The first result implies that unlimited computing time for an exact learning algorithm using only membership queries can be replaced with an efficient relativized computing time using an \mathcal{NP} oracle and random bits. The second result allows one to trade the equivalence oracle for an \mathcal{NP} oracle.

6.4.1. Trading Unlimited Time for the \mathcal{NP} Oracle. We will show that a membership-query learner with *unlimited computational* power can be replaced with a membership-query learner with an access to an \mathcal{NP} oracle and a random source. First we will define a parametrization of concept classes that are exactly learnable from membership queries only.

DEFINITION 6.6. Let \mathcal{LMQ}_n^k be the set of concept classes C over $\{0,1\}^n$ which are exactly learnable using at most n^k membership queries (with unlimited computational power) and such that for any given a set of labeled examples $I \subseteq \{0,1\}^n \times \{0,1\}$, there is an algorithm that, on input f and I , decides whether or not $f \in C$ and this decision algorithm runs in time polynomial in $|I|$.

Note that we impose the same conditions on \mathcal{LMQ}_n^k as we did with concept classes that are samplable with the uniform generation algorithm of Jerrum, Valiant, and Vazirani. An easy fact about the class \mathcal{LMQ}_n^k is that it is closed under taking subsets.

FACT 6.9. If $C \in \mathcal{LMQ}_n^k$ then for any $C' \subseteq C$ we have $C' \in \mathcal{LMQ}_n^k$.

DEFINITION 6.7. An assignment or point $a \in \{0,1\}^n$ is *k-good* for $C \in \mathcal{LMQ}_n^k$ if

$$\gamma_a^C \geq n^{-k} \left(1 - \frac{1}{|C|}\right).$$

The second fact we need is that there is always a *k-good* point for any $C \in \mathcal{LMQ}_n^k$.

FACT 6.10. *Let $C \in \mathcal{LMQ}_n^k$. Then for any $C' \subseteq C$ with $|C'| \geq 2$ there is $a \in \{0,1\}^n$ which is k -good for C' .*

Proof Assume there is $C' \subseteq C$ so that for all $a \in \{0,1\}^n$ a is not k -good for C' , i.e. $\gamma_a^{C'} < n^{-k}(|C'| - 1)/|C'|$. We will show that $C' \notin \mathcal{LMQ}_n^k$ which (by the fact above) will imply $C \notin \mathcal{LMQ}_n^k$. Let A be an arbitrary learning algorithm for C' which uses at most n^k membership queries. Consider the following adversarial strategy for answering queries by A : given the query $MQ(a)$, answer $b \in \{0,1\}$ so that $\gamma_{(a,b)}^{C'} < n^{-k}(|C'| - 1)/|C'|$. This strategy allows A to eliminate less than a $n^{-k}(|C'| - 1)/|C'|$ fraction of C' each time. After n^k steps A can eliminate less than $|C'| - 1$ elements of C' implying there are at least two concepts remaining uneliminated. Since A is arbitrarily chosen, $C' \notin \mathcal{LMQ}_n^k$ as required. \square

As a corollary to the second fact we get that any subset $C' \subset C$, with $|C'| \geq 2$, has an assignment $a \in \{0,1\}^n$ which satisfies

$$\gamma_a^{C'} \geq \frac{1}{2}n^{-k}.$$

The following is the main theorem in this section, which states that there is a randomized expected polynomial time exact learning algorithm for any concept class $C \in \mathcal{LMQ}_n^k$.

THEOREM 6.11. *There is a probabilistic expected polynomial time algorithm with access to an \mathcal{NP} oracle that learns any $C \in \mathcal{LMQ}_n^k$ using at most n^{2k} membership queries.*

Proof Let $C \in \mathcal{LMQ}_n^k$. Set $N \doteq n^k$, $\alpha \doteq 1/16N$ and $m \doteq N^2$.

We say a membership point a is a ϵ -splitter for C if $\gamma_a^C \geq \epsilon$. Recall that the r -th threshold function on n variables TH_r^n is defined as

$$TH_r^n(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^n x_i \geq r \\ 0 & \text{otherwise} \end{cases}$$

Let U be an approximately uniform distribution on C with tolerance $\epsilon = 1$. By a similar argument as in Lemma 6.6, we claim that U can undersample by a factor of at most $(1 + \epsilon)^{-2}$, i.e., if $\gamma_{(x,b)}^C \geq \delta$ then $\Pr_U[f(x) = b] \geq \delta(1 + \epsilon)^{-2}$. We sample independently m functions from C according to U , say $F = \{f_i\}_{i=1}^m \in {}^U C^m$.

Define $T_1 = TH_{\alpha m}^m(F)$ and $T_2 = TH_{(1-\alpha)m}^m(F)$. We prove that with high probability the event $T_1 \not\equiv T_2$ occurs. Since C has a k -good $a \in \{0,1\}^n$, i.e. $\gamma_a^C \geq (2N)^{-1}$, the event $T_1 \equiv T_2$ implies $T_1(a) = T_2(a)$. By Chernoff bounds, Fact 2.6, we get

$$\begin{aligned} \Pr[T_1(a) = T_2(a)] &= \Pr[T_1(a) = 0] + \Pr[T_2(a) = 1] \\ &< 2F^-(m/8N, 1/2) \\ &\leq 2e^{-\Omega(N)}. \end{aligned}$$

Thus with probability $1 - e^{-\Omega(N)}$ we have $T_1 \not\equiv T_2$. Next we show that conditioning on $T_1 \not\equiv T_2$, the event that for all $a \in T_1 \Delta T_2$, $\gamma_a^C \geq (32N)^{-1}$, occurs with high probability. Calling the latter event A , by the union bound and Chernoff bounds again, we have

$$\begin{aligned} \Pr[\bar{A} \mid T_1 \not\equiv T_2] &\leq \sum_{a \in T_1 \Delta T_2} \Pr[T_1(a) \neq T_2(a), \gamma_a^C < (32N)^{-1} \mid T_1 \not\equiv T_2] \\ &\leq 2^n F^+(m/32N, 1) \leq 2^n e^{-\Omega(N)}. \end{aligned}$$

The probability that we failed (at some step) to locate a $(32N)^{-1}$ -splitter is at most $\Pr[T_1 \equiv T_2] + \Pr[\bar{A} \mid T_1 \not\equiv T_2] \leq e^{-\Omega(n)}$.

We use the \mathcal{NP} oracle, for the second time, to find a $(32N)^{-1}$ -splitter $a \in \{0,1\}^n$, which allows progress to be made in learning. We run the above for N^2 times. The probability that at every step we succeed in locating a $(32N)^{-1}$ -splitter (for different invocations of C) is at least $1 - N^2 e^{-\Omega(n)} \geq 1 - e^{-\Omega(n)}$. Thus with probability $1 - e^{-\Omega(n)}$ we will finish (i.e. reduce C to one element) within $N^2 = n^{2k}$ steps. \square

6.4.2. Trading the Equivalence Oracle for the \mathcal{NP} Oracle. We will show a result that allows us to replace or trade an equivalence query oracle with an \mathcal{NP}

oracle. But for this we need to describe a method called the *Monotone theory* due to Bshouty [B93].

DEFINITION 6.8. (*Monotone of a Boolean function*)

Let f be a Boolean function over $\{0,1\}^n$. The monotone of f , denoted $\mathcal{M}(f)$, is defined as $\mathcal{M}(f)(x) = \bigvee_{y \leq x} f(y)$. It can be shown that $\mathcal{M}(f)$ is the unique smallest monotone function that contains f .

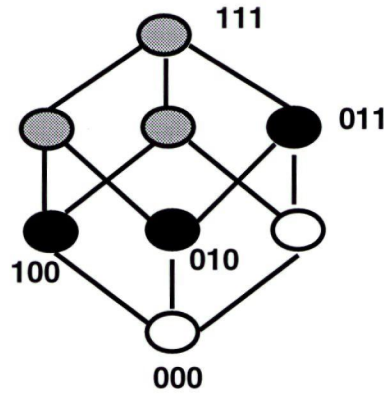


FIGURE 6.3. Monotone of $x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2$

For the next definition we will assume some familiarity with terminology from order theory.

DEFINITION 6.9. (*The partial order \leq_a*)

Let $\{0,1\}^n$ be the Boolean n -cube. For an assignment $a \in \{0,1\}^n$ the partial ordering \leq_a on $\{0,1\}^n$ is defined as follows: for all $x, y \in \{0,1\}^n$

$$x \leq_a y \iff (x \oplus a) \leq (y \oplus a).$$

The a -monotone $\mathcal{M}_a(f)$ of f is defined as $\mathcal{M}_a(f)(x) = \mathcal{M}(f(x \oplus a))(x \oplus a)$. A main characterization theorem of the monotone theory is that any Boolean function f is expressible as a conjunction of its monotone components $\mathcal{M}_a(f)$, for all a 's.

FACT 6.12. [B93] *For any Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ we have*

$$f(x) = \bigwedge_{a \in \{0,1\}^n} \mathcal{M}_a(f)(x).$$

This leads naturally to a definition of the monotone dimension of a concept class.

DEFINITION 6.10. (*Monotone dimension*)

The monotone dimension $\mathcal{Mdim}(f)$ of f is the size of the smallest subset $A \subseteq \{0,1\}^n$ so that $f(x) = \bigwedge_{a \in A} \mathcal{M}_a(f)(x)$. The monotone dimension of a concept class C , denoted by $\mathcal{Mdim}(C)$, is the size of the smallest subset $A \subseteq \{0,1\}^n$ so that for all $f \in C$

$$f(x) = \bigwedge_{a \in A} \mathcal{M}_a(f)(x).$$

Duality is a very powerful tool in order theory. Using duality we can define dual notions for the above definitions.

DEFINITION 6.11. Let f be a Boolean function over $\{0,1\}^n$. The dual monotone of f , denoted $\mathcal{M}^\partial(f)$, is defined as $\mathcal{M}^\partial(f)(x) = \bigwedge_{y \geq x} f(y)$. It can be shown that $\mathcal{M}^\partial(f)$ is the unique largest monotone function that is contained in f .

It is true that for any Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ we have

$$f(x) = \bigvee_{a \in \{0,1\}^n} \mathcal{M}_a^\partial(f)(x).$$

The dual monotone dimension $\mathcal{M}^\partial \dim(f)$ of f is the size of the smallest subset $A \subseteq \{0,1\}^n$ so that $f(x) = \bigvee_{a \in A} \mathcal{M}_a^\partial(f)(x)$. The dual monotone dimension of a concept class C , denoted by $\mathcal{M}^\partial \dim(C)$, is the size of the smallest subset $A \subseteq \{0,1\}^n$ so that for all $f \in C$

$$f(x) = \bigvee_{a \in A} \mathcal{M}_a^\partial(f)(x).$$

The main algorithmic result in [B93] is stated in the following.

FACT 6.13. *Let C be any concept class and let $f \in C$ be a target concept from C . Then there is an exact learning algorithm that runs in time polynomial in $\text{size}_{DNF}(f)$*

and $\mathcal{M}dim(C)$ and uses

$$\mathcal{E} \doteq size_{DNF}(f)\mathcal{M}dim(C)$$

equivalence queries and $\mathcal{E} \cdot n^2$ membership queries. Moreover any hypothesis h issued by this algorithm satisfies $h \leq f$.

There is also a dual algorithm that runs in time polynomial in $size_{CNF}(f)$ and $\mathcal{M}^\partial dim(C)$ and uses

$$\mathcal{E}^\partial \doteq size_{CNF}(f)\mathcal{M}^\partial dim(C)$$

equivalence queries and $\mathcal{E}^\partial \cdot n^2$ membership queries. Moreover any hypothesis h issued by this dual algorithm satisfies $f \leq h$.

The following lemma describes a technique to combine two exact learning algorithms that use equivalence and membership queries into one exact learning algorithm that uses only membership queries with an access to an \mathcal{NP} oracle. This technique eliminates the need for the equivalence oracle at the expense of introducing the \mathcal{NP} oracle.

LEMMA 6.14. *Let C be a concept class over $\{0,1\}^n$. Let L_1 and L_2 be two exact learning algorithms which use equivalence and membership queries to learn C . Suppose that any hypotheses h_1 and h_2 issued by both are known to satisfy $h_1 \not\equiv h_2$ (except on the last step). Then there is an algorithm that uses membership queries and the \mathcal{NP} oracle to learn C .*

Proof The idea is to run L_1 and L_2 in parallel until the first equivalence query is issued by each, say h_1 and h_2 , respectively. Since we know $h_1 \not\equiv h_2$, we can use the \mathcal{NP} oracle to find an assignment $c \in \{0,1\}^n$ such that $h_1(c) \neq h_2(c)$. This can be done as follows. We can find out the first bit of c by asking which of the following is true:

$$h_1|_{x_1 \leftarrow 0} \neq h_2|_{x_1 \leftarrow 0}, \quad h_1|_{x_1 \leftarrow 1} \neq h_2|_{x_1 \leftarrow 1}$$

The bit that keeps the non-equivalence is the first bit of c . So after n such questions, i.e., n \mathcal{NP} queries, we will discover c completely. This technique is also known as *self-reduction* in complexity theory. A membership query at c will establish which algorithm may continue its execution (we suspend the other). We then repeat the process again until the continued algorithm issues its next equivalence query. By assumption, the suspended equivalence query and the new one are still not equal. Again we use the \mathcal{NP} oracle to find a counterexample for one of them, and so on. In this way we never ask any equivalence queries but we incur the expense of spending n \mathcal{NP} queries and one membership query every time we need to ask an equivalence query. \square

We observe that if we run both algorithms from Fact 6.13 in the manner as described in the previous lemma, then the hypotheses issued by both algorithms will never be equal except when they are equal to the target function. Hence we can conclude the following.

THEOREM 6.15. *Let C be a concept class over $\{0,1\}^n$ and let $f \in C$ be a target concept from C . Then there is an algorithm that learns C using at most*

$$n(n+1)(\text{size}_{DNF}(f)\mathcal{M}dim(C) + \text{size}_{CNF}(f)\mathcal{M}^\partial dim(C))$$

membership queries and

$$n(\text{size}_{DNF}(f)\mathcal{M}dim(C) + \text{size}_{CNF}(f)\mathcal{M}^\partial dim(C))$$

\mathcal{NP} queries.

Proof The factor of $(n+1)$ in the number of calls to the \mathcal{NP} oracle is to account for one call to check if the two hypotheses are equal and n calls to find a counterexample if they are not equal. \square

COROLLARY 6.16. *The following classes are exactly learnable with membership queries and an \mathcal{NP} oracle in time polynomial in n and in the maximum of the DNF*

and CNF sizes of the target concept.

- (1) *Monotone Boolean functions.*
- (2) $\mathcal{O}(\log n)\text{-CNF} \cap \mathcal{O}(\log n)\text{-DNF}$.

Proof We use the fact that the class C of monotone Boolean functions satisfies $\mathcal{M}dim(C) = \mathcal{M}^\partial dim(C) = 1$, which proves the first claim. For the second claim, we have the fact that the class $\mathcal{O}(\log n)\text{-CNF}$ is known to have polynomial (in n) monotone dimension while the class $\mathcal{O}(\log n)\text{-DNF}$ is known to have polynomial (in n) dual monotone dimension [B93]. \square

In the next lemma we provide a lower bound on the number of membership queries required to learn Boolean concept classes. We show that a certain class of monotone *read-twice* DNF formulas, i.e., where each variable can appear at most twice, is not polynomial-time exactly learnable from only membership queries. The latter class is known to be exactly learnable from equivalence and membership queries (see Aizenstein and Pitt [AP91], Hancock [H91], and Pillaipakkamnatt and Raghavan [PR95]). This is also in contrast to the fact that the class of monotone read-once Boolean formulas is exactly learnable in polynomial-time from membership queries alone [AHK93].

LEMMA 6.17. *Let f be a Boolean function over $\{0,1\}^n$. Any learning algorithm that exactly identifies f using only membership queries requires at least*

$$\Omega(\max\{size_{DNF}(f), size_{CNF}(f)\})$$

membership queries.

Proof We use an adversarial argument on the following class of monotone read-twice DNF formulas

$$\mathcal{C} = \{f \mid f = \bigvee_{i=1}^k T_i \vee T\}.$$

For each i , $1 \leq i \leq k$, let T_i be the conjunction of all variables in the i th block where each block contains n/k variables. Let V_i denote this i th block of variables,

i.e., $V_i = \{x_{(i-1)(n/k)+1}, \dots, x_{i(n/k)}\}$. The last term T consists of all variables except that it is missing exactly one variable from each T_i . So $|\mathcal{C}| = (n/k)^k$.

For the lower bound argument we give away to the learner the information about all the terms T_i , $1 \leq i \leq k$, but not any information about the term T , i.e., where the missing variables are. Suppose the learner asks a membership query with $a \in \{0, 1\}^n$. Note that a cannot be all one in any T_i since then f is satisfied and the learner knows this already. If a contains more than one zero in some V_i then the adversary says 0 if and only if a falsifies all T_i , $1 \leq i \leq k$, and 1 otherwise. This conveys no information about T since a falsifies T . Hence the learner must ask membership queries where there is precisely one zero in each V_i . There are $(n/k)^k$ such questions and the adversary may answer 0 except for the last one.

Recall that an assignment $\beta \in \{0, 1\}^n$ is called a *maxterm* of a Boolean function f if $f(\beta) = 0$ and β contains a minimal number of zeroes, i.e., there is no $\alpha < \beta$ that satisfies $f(\alpha) = 0$. It is also known that the CNF size of a monotone Boolean function is characterized by the number of maxterms. We note that the maxterms of any $f \in \mathcal{C}$ are exactly those assignments $a \in \{0, 1\}^n$ which have precisely one zero in each V_i . Hence $\text{size}_{\text{CNF}}(f) = (n/k)^k$ and $\text{size}_{\text{DNF}}(f) = k + 1$, which completes the claim. \square

Conclusions and Open Problems

Beware of bugs in the above code; I have only proved it correct, not tried it.

– Donald Knuth

7.1. Summary

In this thesis we have considered the following main question in computational learning theory:

Are Boolean functions represented as Disjunctive Normal Form formulas or as Boolean circuits efficiently learnable?

The original question of Valiant is whether DNF formulas are PAC learnable under any distribution [V84b]. This question is still open to this date. The closest answer to this question was given by Jackson [J94]. Jackson proved that DNF formulas are efficiently PAC learnable if the underlying distribution is the *uniform* distribution and if the learning algorithm is given access to a membership oracle.

- In the first part of the thesis we have shown that monotone DNF formulas are PAC learnable under *product* distributions in subexponential time. Thus the price that we pay for allowing a more general distribution and for not allowing the learning algorithm to ask membership questions is efficiency.
- In the second part of this thesis we show that Jackson's result can be extended to a representation class that includes DNF formulas as a strict subclass. More specifically we prove that the representation class of monotone

width two branching programs is efficiently PAC learnable under the uniform distribution with membership queries.

- In the third part of this thesis we prove that DNF formulas are *exactly* learnable using only equivalence queries assuming the learning algorithm has access to an \mathcal{NP} oracle. This implies that if $\mathcal{P} = \mathcal{NP}$ then DNF formulas are exactly learnable using only equivalence queries and if DNF formulas are *not* exactly learnable using equivalence queries then $\mathcal{P} \neq \mathcal{NP}$.

Recently, Bshouty [B96] proved that DNF formulas are exactly learnable from equivalence queries in subexponential time, i.e., $2^{\tilde{O}(\sqrt{n})}$, without the \mathcal{NP} oracle.

In his original paper, Valiant [V84b] also mentioned that the existence of a cryptographic object called *pseudorandom functions* implies that the class of polynomial-sized Boolean circuits is not efficiently PAC learnable under any distribution. Since then Valiant's observation has been improved to show that the class of polynomial-sized Boolean circuits with *logarithmic depth*, or NC^1 , is not PAC learnable under any distribution even if we allow the learning algorithm access to a membership oracle [AK95].

- In the first part of the thesis we prove that any monotone Boolean circuit (without any size and depth constraints) is PAC learnable under product distributions in subexponential time. This is similar to the result of Linial, Mansour, and Nisan [LMN93] that gave an $n^{\text{poly}(\log n)}$ time PAC learning algorithm for AC^0 functions.
- In the second part of the thesis we study the learnability of classes that are *below* NC^1 or width five branching programs. Using some characterization results developed by Barrington, we investigate the problem of learning permutation branching programs with widths strictly less than five.
- In the third part of the thesis we prove that the class of Boolean circuits is exactly learnable using only equivalence queries assuming the learning algorithm

has access to an \mathcal{NP} oracle. This again showed that if the class of Boolean circuits is not exactly learnable using only equivalence queries then $\mathcal{P} \neq \mathcal{NP}$.

From the perspective of complexity theory, we have developed a new Fourier spectrum characterization of monotone Boolean functions by proving that their average sensitivity is at most \sqrt{n} , we have established a connection between exact learning and the difficult question $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$, and have provided some new applications of branching program characterizations to learning theory.

7.2. Minor Extensions

We mention in this section some extensions to results in this thesis.

Noise tolerance and Group learning. It is known that the LMN algorithm fall in the category of a statistical query algorithm (see Kearns [K93]). Hence any learning result that we obtain using this algorithm will be *noise tolerant* in Kearns's *statistical query* model. Kearns [K] also introduced the notion of *group* learning and proved that it is equivalent to the notion of *weak* learning. Informally, in group learning we require that the learning algorithm to find a hypothesis that is accurate in classifying a polynomial-sized group of examples that are either all positive or all negative, instead of being accurate on only one example. Using results from Chapter 4 we can claim that the class of all monotone Boolean functions is group learnable under product distributions in polynomial-time.

Locally monotone and Total orders. Some of the results in Chapters 3 and 4 can be extended to *unate* (or locally monotone) Boolean functions. A Boolean function f is unate if there is an assignment $a \in \{0, 1\}^n$ so that $f(x \oplus a)$ is a monotone function. Notice that a monotone Boolean function is a unate Boolean function with $a = 0_n$. Some of the results in Chapter 4 can also be extended to non-Boolean monotone functions over a total order. A monotone Boolean function is a monotone function over the total order $0 < 1$.

Nested differences. The algorithm that we use to learn width two branching programs with $\mathcal{O}(1)$ sinks is the algorithm for learning nested differences of intersection-closed concept classes. Recently, Auer [A95] has shown that this algorithm can be made robust against malicious noise. In this noise model, the counterexample provided by the equivalence oracle is not always correct, i.e., this oracle may *lie* once in a while. Usually learning is parametrized by the number of lies that the equivalence oracle can make. We believe that the results from Chapter 5 can be made to work in the malicious noise model.

7.3. Open Questions

In the following we describe some natural open questions raised by this thesis.

Fourier spectrum vs. Circuit complexity. Can we prove a Fourier spectrum result on monotone Boolean functions that takes into account *circuit complexity*? Linial, Mansour and Nisan [LMN93] proved a remarkable connection between the Fourier spectrum and circuit complexity of AC^0 functions. They proved that any Boolean function f from AC_d^0 of size m must satisfy

$$\sum_{|a| \geq k} \hat{f}(a)^2 \leq 2m \exp(-k^{1/d}/20).$$

It is not clear how to prove a similar result for monotone Boolean functions.

Learning monotone Boolean functions. In view of Jackson's DNF learning result [J94], are monotone DNF formulas PAC learnable under the uniform distribution *without* membership queries? The key reason for membership queries in Jackson's result is that we don't know where the relevant Fourier coefficients are and thus the need to invoke Kushilevitz and Mansour's KM algorithm. If the DNF formula is monotone then we know a bit more about the location of the relevant coefficients with respect to the uniform distribution, i.e., they are at the unit vectors. Unfortunately, we don't know what happens to them once the boosting stages take place.

Learning bounded width branching programs. The results of [BTW96] showed that the class of strict width two branching programs is properly learnable in the distribution-free PAC model. We do not know, at the time of writing, if this can be extended to the exact identification model. Recently, Nakamura [N96] has solved this open question and proved that SW_2 is efficiently properly exactly learnable. The results from Chapter 5 showed that the class of width two branching programs with $\mathcal{O}(1)$ sinks is learnable in the exact identification model but not properly. We do not know if this can be extended to proper learning.

Collapse consequences. Are Boolean circuits exactly learnable in deterministic polynomial-time with the aid of an \mathcal{NP} oracle? Note that an affirmative answer to this question would yield a collapse of the polynomial-time hierarchy to $\mathcal{P}^{\mathcal{NP}}$. Köbler and Watanabe [KW95] have recently extended Watanabe's result on the collapse consequences of $\mathcal{NP} \subseteq P/poly$.

Monotone duality. It was shown in Chapter 6 that monotone Boolean functions are exactly learnable using only membership queries provided that the learning algorithm has access to an \mathcal{NP} oracle. The learning complexity depended on the sum of the DNF and CNF sizes of the target Boolean function. Can we obtain the same result without using the \mathcal{NP} oracle? Recently, Fredman and Khachiyan [FK94] have proved that a related problem, called monotone duality, can be solved in slightly superpolynomial time. The problem of monotone duality requires an algorithm to decide if two monotone Boolean functions, one given in DNF and the other given in CNF, are equivalent. Their result directly implies an algorithm running in time $m^{o(\log m)}$ for the problem of learning monotone Boolean functions from membership queries. Whether this result can be improved to $m^{\mathcal{O}(1)}$ is still open.

Bibliography

- [Ang88] Dana Angluin. Queries and Concept Learning. *Machine Learning*, 2:319–342, 1988.
- [Ang90] Dana Angluin. Negative Results for Equivalence Queries. *Machine Learning*, 5:121–150, 1990.
- [Ang92] Dana Angluin. Computational Learning Theory: Survey and Selected Bibliography. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, 351–369, 1992.
- [A95] Peter Auer. Learning Nested Differences in the Presence of Malicious Noise. In *Proceedings of the 6th International Workshop on Algorithmic Learning Theory*, Lecture Notes in Artificial Intelligence No. 997, Springer, 123–137, 1995.
- [AHK93] Dana Angluin, Lisa Hellerstein, and Marek Karpinski. Learning Read-Once Formulas with Queries. In *Journal of ACM*, 40(1):185–210, 1993.
- [AK95] Dana Angluin and Michael Kharitonov. When Won’t Membership Queries Help? In *Journal of Computer and System Sciences*, 50:336–355, 1995.
- [AM91] William Aiello and Milena Mihail. Learning the Fourier spectrum of Probabilistic Lists and Trees. In *Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms*, 291–299, 1991.
- [AP91] Howard Aizenstein and Leonard Pitt. Exact Learning of Read-Twice DNF Formulas. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, 170–179, 1991.
- [AS92] Noga Alon and Joel Spencer. *The Probabilistic Method*. John Wiley and Sons, 1992.

- [B86] David A. Barrington. Bounded-Width Branching Programs. PhD thesis, Massachusetts Institute of Technology, 1986.
- [Bar89] David A. Barrington. Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1 . In *Journal of Computer and System Sciences*, 38:150–164, 1989.
- [B] David A. Mix Barrington. Personal communication, 1996.
- [B96] Nader H. Bshouty. Towards the Learnability of DNF Formulae. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, 131–140, 1996.
- [Bop89] Ravi Boppana. Amplification of Probabilistic Boolean Formulas. In *Advances in Computing Research*, Silvio Micali (ed.), 5(4):27–45, 1989.
- [Bl94] Avrim Blum. Separating Distribution-Free and Mistake-Bound Learning Models over the Boolean Domain. In *SIAM Journal on Computing*, 23(5):990–1000, 1994.
- [Bl92] Avrim Blum. Rank- r Decision Trees are a Subclass of r -Decision Lists. In *Information Processing Letters*, 42:183–185, 1992.
- [B93] Nader H. Bshouty. Exact Learning via the Monotone Theory. In *Proceedings of the 34th IEEE Symposium on the Foundations of Computer Science*, 302–311, 1993.
- [B95] Nader H. Bshouty. Simple Learning Algorithms using Divide and Conquer. In *Proceedings of the 8th Annual ACM Workshop on Computational Learning Theory*, 447–453, 1995.
- [BBBKV96] Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, Stefano Varricchio. On the Applications of Multiplicity Automata in Learning. To appear in *Proceedings of the 37th IEEE Symposium on the Foundations of Computer Science*, 1996.
- [BBTV96] Francesco Bergadano, Nader H. Bshouty, Christino Tamon, and Stefano Varricchio. On Learning Branching Programs and Small Depth Circuits. In *Electronic Colloquium on Computational Complexity*, TR96-09, 1996.
- [BCG⁺] Nader H. Bshouty, Richard Cleve, Ricard Gavaldà, Sampath Kannan, and Christino Tamon. Oracles and Queries that are Sufficient for Exact Learning. In

- a special issue for COLT94 *Journal of Computer System and Sciences*, 52(3):421-433, 1996.
- [BCKT94] Nader H. Bshouty, Richard Cleve, Sampath Kannan, and Christino Tamon. Oracles and Queries that are Sufficient for Exact Learning. In *Proceedings of the 7th Annual ACM Workshop on Computational Learning Theory*, 130–139, 1994.
- [BCV96] Francesco Bergadano, D. Catalano, and Stefano Varricchio. Learning Sat- k -DNF Formulas from Membership Queries. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, 126–130, 1996.
- [BDFP86] Allan Borodin, Danny Dolev, Faith Fich, and Wolfgang Paul. Bounds for Width Two Branching Programs. In *SIAM Journal on Computing*, 15(2):549–560, 1986.
- [BEHW89] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred Warmuth. Learnability and the Vapnik-Chernovenkis Dimension. In *Journal of the Association for Computing Machinery*, 36(4):929–965, 1989.
- [BFJ⁺94] Avrim Blum, Merrick Furst, Jeffrey Jackson, Michael Kearns, Yishay Mansour, and Steven Rudich. Weakly Learning DNF and Characterizing Statistical Query Learning using Fourier Analysis. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, 253–262, 1994.
- [BT95] Nader H. Bshouty and Christino Tamon. On the Fourier Spectrum of Monotone Functions. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, 219–228, 1995. Final version to appear in *Journal of the ACM*, 43:4, July 1996.
- [BTW96] Nader H. Bshouty, Christino Tamon, and David K. Wilson. On Learning Width Two Branching Programs. In *Proceedings of the 9th Annual ACM Conference on Computational Learning Theory*, 224–227, 1996.
- [BV94] Francesco Bergadano and Stefano Varricchio. Learning Behaviors of Automata from Multiplicity and Equivalence Queries. In *Proceedings of the 2nd Italian Conference on Algorithms and Complexity (CIAC 94)*, Lecture Notes in Computer Science No. 778, Springer-Verlag, 1994.
- [CG88] Benny Chor and Mihaly Geréb-Graus. On the Influence of Single Participant in Coin Flipping Schemes. In *SIAM Journal on Discrete Mathematics*, 1, 1988.

- [EH89] Andrzej Ehrenfeucht and David Haussler. Learning Decision Trees from Random Examples. In *Information and Computation*, **82**(3):231-246, 1989.
- [EHKV88] Andrzej Ehrenfeucht, David Haussler, Michael Kearns, and Leslie Valiant. A General Lower Bound on the Number of Examples Needed for Learning. In *Proceedings of the 1st Workshop on Computational Learning Theory*, 139-154, 1988.
- [ERR95] Funda Ergün, S. Ravi Kumar, and Ronitt Rubinfeld. On Learning Bounded-Width Branching Programs. In *Proceedings of the 8th Annual ACM Conference on Computational Learning Theory*, 361-368, 1995.
- [F90] Yoav Freund. Boosting a Weak Learning Algorithm by Majority. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory*, 202-216, 1990.
- [FJS91] Merrick Furst, Jeffrey Jackson, and Sean Smith. Improved Learning of AC^0 Functions. In *Proceedings of the 4th Annual Workshop on Computational Learning Theory*, 317-325, 1991.
- [FK94] Michael Fredman and Leonid Khachiyan. On the Complexity of Dualization of Monotone Disjunctive Normal Forms. Technical Report LSCR-TR-225, Department of Computer Science, Rutgers University, 1994.
- [GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP -completeness*. W.H. Freeman, 1979.
- [H91] Thomas Hancock. Learning 2μ DNF Formulas and $k\mu$ Decision Trees. In *Proceedings of the 4th Annual Workshop on Computational Learning Theory*, 199-209, 1991.
- [HM91] Thomas Hancock and Yishay Mansour. Learning Monotone $k\text{-}\mu$ DNF Formulas on Product Distributions. In *Proceedings of the 4th Annual Workshop on Computational Learning Theory*, 179-183, 1991.
- [HSW90] David Helmbold, Robert Sloan, and Manfred Warmuth. Learning Nested Differences of Intersection-Closed Concept Classes. In *Machine Learning*, 5:165-196, 1990.

- [J] David S. Johnson. A Catalog of Complexity Classes. In *Handbook of Theoretical Computer Science*, Volume A: Algorithms and Complexity, J. van Leeuwen (ed.), pages 67–161, MIT Press/Elsevier, 1990.
- [J94] Jeffrey Jackson. An Efficient Membership-Query Algorithm for Learning DNF with Respect to the Uniform Distribution. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 42–53, 1994.
- [JVV86] Mark Jerrum, Leslie Valiant, and Vijay Vazirani. Random Generation of Combinatorial Structures from a Uniform Distribution. In *Theoretical Computer Science*, 43:169–188, 1986.
- [K93] Sampath Kannan. On the Query Complexity of Learning. In *Proceedings of the 6th Annual ACM Conference on Computational Learning Theory*, 58–66, 1993.
- [KL80] Richard M. Karp and Richard J. Lipton. Some Connections Between Nonuniform and Uniform Complexity Classes. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, 302–309, 1980.
- [KLPV87] Michael Kearns, Ming Li, Leonard Pitt, and Leslie Valiant. On the Learnability of Boolean Formulae. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, 1987.
- [KM93] Eyal Kushilevitz and Yishay Mansour. Learning Decision Trees using the Fourier Spectrum. In *SIAM Journal on Computing*, 22(6):1331–1348, 1993.
- [K] Michael Kearns. *The Computational Complexity of Machine Learning*. MIT Press, 1990.
- [K93] Michael Kearns. Efficient Noise Tolerant Learning from Statistical Queries. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, 392–401, 1993.
- [KKL88] Jeff Kahn, Gil Kalai, and Nathan Linial. The Influence of Variables on Boolean Functions. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, 68–80, 1988.
- [KP94] Matthias Krause and Pavel Pudlák. On the Computational Power of Depth 2 Circuits with Threshold and Modulo gates. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, 48–57, 1994.

- [KV89] Michael Kearns and Leslie Valiant. Cryptographic Limitations on Learning Boolean Formulae and Finite Automata. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, 433–444, 1989.
- [KW95] Johannes Köbler and Osamu Watanabe. New Collapse Consequences of \mathcal{NP} Having Small Circuits. In *Proceedings of 22nd International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, No. 944, Springer-Verlag, 196–207, 1995.
- [L88] Nick Littlestone. Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm. *Machine Learning*, **2**:285–318, 1988.
- [LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant Depth Circuit, Fourier Transform and Learnability. *Journal of ACM*, **40**(3):607–620, 1993.
- [M94] Yishay Mansour. Learning Boolean Functions via the Fourier Transform. Tutorial Notes for the *Workshop on Computational Learning Theory*, 1994.
- [N96] Atsuyoshi Nakamura. Query Learning of Bounded-Width OBDDs. To appear in *Algorithmic Learning Theory (ALT)*, 1996.
- [P94] Christos H. Papadimitriou. Computational Complexity. *Addison-Wesley*, 1994.
- [PR95] Khrisnan Pillaipakkamnatt and Vijay Raghavan. Read-Twice DNF Formulas are Properly Learnable. In *Information and Computation*, **122**(2):236–267, 1995.
- [PV88] Leonard Pitt and Leslie Valiant. Computational Limitations on Learning from Examples. In *Journal of the ACM*, **35**(4):965–984, 1988.
- [R90] Prabhakar Raghavan. Lecture Notes on Randomized Algorithms. *Research Report, IBM Research Division*, RC15340 (#68237), 1/9/90.
- [R87] Ronald Rivest. Learning Decision Lists. In *Machine Learning*, 2:229–246, 1987.
- [R92] Steven Roman. *Coding and Information Theory*. Springer-Verlag, 1992.
- [Si] Alistair Sinclair. *Algorithms for Random Generation and Counting*. Birkhäuser, 1993.
- [S] Robert E. Schapire. The Design and Analysis of Efficient Learning Algorithms. *MIT Press*, 1992.

- [SM94] Yoshifumi Sakai and Akira Maruoka. Learning Monotone Log-Term DNF Formulas. In *Proceedings of the 7th Annual ACM Conference on Computational Learning Theory*, 165–172, 1994.
- [T95] Christino Tamon. A Short Proof of a Fourier Theorem. Research Report No. 95/576/28, Dept. Computer Science, Univ. Calgary, November 1995.
- [V84a] Leslie Valiant. Short Monotone Formulae for the Majority Function. In *Journal of Algorithms*, 5:363–366, 1984.
- [V84b] Leslie Valiant. A Theory of the Learnable. In *Communications of the ACM*, 27(11):1134–1142, 1984.
- [W87] Ingo Wegener. The Complexity of Boolean Functions. *Wiley-Teubner*, 1987.

Closing Credits

The purpose of this epilogue is to explain my personal involvement in the research that is presented in this thesis. My thanks to Professor Bruce MacDonald for pointing out the ambiguity surrounding this issue.

The research presented in Chapter 3 started when Jeff Jackson explained to me a proof outline for the expression $I_i(f) = \sum_{a:a_i=1} \hat{f}(a)^2$. I noticed that this proof could be extended to product distributions. This extension was originally used to provide an alternative proof of Theorem 3.10 (not presented in the thesis).

In an attempt to improve some results in [KV89], I proposed the use of Cauchy-Schwarz inequality to simplify some expressions that I was struggling with. This led to the proof of Theorem 3.12 which I developed together with Nader. Shortly after, Yishay Mansour and Eyal Kushilevitz suggested another formulation of Theorem 3.12 using average sensitivity. This led to the fact that the average sensitivity of any monotone Boolean function is at most \sqrt{n} . At the same time Dan Boneh suggested to us stating things in terms of *influence norm*.

I came across a well-known identity in Fourier analysis, called the cross correlation identity, that could be used to derive most other identities, such as Parseval, auto-correlation, and convolution. In fact by *brute force* I used it in the proof of Theorem 3.10 (a folklore theorem of Kahn, Kalai, and Linial); in the uniform distribution case, this provides a simpler and more direct proof.

In Chapter 4, following standard lines from Linial, Mansour, and Nisan's paper [LMN93], we traced the details to for the PAC learning of monotone Boolean functions under arbitrary product distributions. We did the analysis presented in Section 4.3 after a watchful comment

from a JACM referee. I made a premature conjecture that the Fourier spectrum bound of \sqrt{n} for monotone Boolean function might be tight. I tried looking at the case when Cauchy-Schwarz attains equality, but this did not help and the conjecture remained unsolved. This led to the statement on the majority function. We then looked at other lower bounds. With Jeff Jackson's help, I noticed an application of Kahn et al's lower bound result on influences for improving Kearns and Valiant's weak learning result for all monotone Boolean functions. It also provided an alternative proof to the one presented in [KV89].

My research involvement in studying the learnability of branching programs started with a project on learning strict width two branching programs done in collaboration with Nader and David Wilson. I noticed that, in the width two case, the number of sinks had something to do with the *rank* of certain decision trees. I thought of applying Blum's famous argument for transforming the decision tree into a decision list. Together with my coauthors, we found the right argument that constitutes the proof for learning constant sink width two branching programs. I supplied some of the technical details that constitute the proof of Lemma 5.4 (that is an extension of Jeff Jackson's observation [J94]).

I thought about looking at branching programs of widths bigger than two, i.e., three and four, but realized that, since it will be as hard as solving the learnability of DNF formulas, perhaps only permutation branching programs could give something. I got in email contact with David Mix Barrington where he mentioned and explained to me some alternative forms of width three and four permutation branching programs. Nader then put me in touch with the recent results using multiplicity automata based on the work of my other coauthors, Francesco Bergadano and Stefano Varricchio. Using the alternate forms of widths three and four permutation branching programs, together we solved the details of their learnability.

My involvement in research on learning with the \mathcal{NP} oracle started when Nader suggested on looking at learning Boolean functions with the membership and the NP oracle. I noted the application of self-reducibility arguments for our purpose, e.g., Lemma 6.14, in reducing search to decision. At that time, we were working on an extension to the Monotone Theory (based on lattice theory) that subsequently appeared elsewhere. I proposed some simplifications to the probabilistic analysis used in Theorem 6.11. I also contributed to the technical expositions on learning with the equivalence and \mathcal{NP} oracle.