A Complexity Theoretic Approach to Learning

by

Adam Richard Klivans

B.S., Math/Computer Science, Carnegie Mellon University (1997)
M.S., Mathematics, Carnegie Mellon University (1997)

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2002

Author . . . .
Department of Mathematics
May 14, 2002

Certified by . . . .
Daniel A. Spielman
Assistant Professor
Thesis Supervisor

Accepted by . . . .
Chair, Applied Mathematics

Accepted by . . . .
Tomasz S. Mrowka
Chairman, Department Committee on Graduate Students

A Complexity Theoretic Approach to Learning

by

Adam Richard Klivans

Submitted to the Department of Mathematics
on May 14, 2002, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis details a new vantage point for attacking longstanding problems in machine learning. We use tools from computational complexity theory to make progress on problems from computational learning theory. Our methods yield the fastest and most expressive algorithms to date for learning several fundamental concept classes:

- We show that any s-term DNF over n variables can be computed by a polynomial threshold function of order $O(n^{1/3} \log s)$. As an immediate consequence we obtain the fastest known DNF learning algorithm which runs in time $2^{\tilde{O}(n^{1/3})}$.

- We give the first polynomial time algorithm to learn an intersection of a constant number of halfspaces under the uniform distribution to within any constant error parameter. We also give the first quasipolynomial time algorithm for learning any function of a constant number of halfspaces with polynomial bounded weights under *any* distribution.

- We give an algorithm to learn constant-depth polynomial-size circuits augmented with majority gates under the uniform distribution using random examples only. For circuits which contain a polylogarithmic number of majority gates the algorithm runs in quasipolynomial time. Under a suitable cryptographic assumption we show that these are the most expressive circuits which will admit a non-trivial learning algorithm.

Our approach relies heavily on giving novel representations of well known concept classes via complexity theoretic reductions. We exploit the fact that many results in computational learning theory have a complexity theoretic analogue or implication. As such, we also obtain new results in computational complexity including (1) a proof that the 30 year old lower bound due to Minsky and Papert [88] on the degree of a perceptron computing a DNF formula is tight and (2) improved constructions of pseudo-random generators, mathematical objects which play a fundamental role in cryptography and derandomization.

Thesis Supervisor: Daniel A. Spielman
Title: Assistant Professor

Acknowledgments

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The question "What does it mean for a machine to learn?" is one of the great subjects of general scientific inquiry. The problem has attracted interest from researchers in many fields including computer science, cognitive science, statistics, linguistics, and mathematics. Attempts to answer this question vary widely from discipline to discipline. For example, a computer capable of beating stronger and stronger chess opponents may be described as learning how to play the game. A robot navigating unknown terrain may be described as learning its environment, or a personal computer adapting to its user's preferences may be described as learning some (admittedly mundane) aspects of human behavior.

In his seminal 1984 paper "A Theory of the Learnable" Valiant proposed a concrete, robust answer to this question. He defined a model of computational data prediction whereby a computer, given access to a set of data points labelled positive and negative, attempts to classify new, unlabelled data points correctly. In other words, given access to a training set of data labelled according to some particular unknown function, can a machine *learn* how the function classifies data and correctly label future data points by itself? The key to Valiant's model is the requirement that this task be carried out in a computationally efficient manner. A function is considered to be learnable in this model if a computer program, after seeing how the function classifies some initial training data, can efficiently produce a hypothesis that will mimic the behavior of the function on future random examples.

In order to evaluate the utility of such a model, we might ask the following questions: (1) How well does this model correspond to our notion of what it means to learn? (2) Can we learn any "interesting" or "expressive" functions in this model? (3) What practical applications do learning algorithms developed for this model have? and finally (4) What insights has this model provided with respect to computer science and computing theory as a whole?

Research concerning Valiant's model, now often referred to simply as computational learning theory, attempts to answer these questions. The framework introduced in his paper has been widely celebrated. In the last ten years alone, there have been over 600 research articles from such diverse areas as philosophy, computational biology, brain and cognitive sciences, information theory, and circuit design, to name a few, citing his original paper.

His paradigm for learning has been embraced primarily by theoretical computer scientists and mathematicians, as it demands rigorous, provably correct learning algorithms and values polynomial-time computational complexity. Still, researchers from both pure and applied fields have influenced or been influenced by work in this area. Scientists from artificial intelligence, mathematics, and other disciplines find the definitions in Valiant's model highly general, intuitively appealing, and mathematically satisfying. In the last twenty years, researchers have developed a bevy of efficient algorithms and techniques for learning different functions (usually referred to as concept classes). The often theoretical underpinnings of the area have lent themselves, perhaps somewhat surprisingly, to practical application in computer vision and economics. Many purely mathematical results have given us new ways to understand the power of computing, randomization and derandomization, cryptography, and combinatorics.

That is not to say that the model is without its drawbacks. In practice, some algorithms which have a guarantee of polynomial time efficiency do not run as quickly as some commonly used heuristics, algorithms which have no mathematical guarantee of success. Furthermore, certain learning phenomena, such as language processing, do not appear to be natural instantiations of the model.

Another drawback of the model is that the limitations of what concept classes or functions are efficiently learnable seem severe. Results from cryptography and computational complexity show that any concept class capable of assigning labels to data in a sufficiently pseudo-random way (and relatively innocuous looking concept classes are capable of this) cannot be learned unless certain problems thought to be intractable are actually easy. Furthermore, certain restricted types of learning in this model for even the simplest concept classes has been shown to be intractable unless $\mathsf{P} = \mathsf{NP}$ (which would violate one of the most widely held mathematical assumptions).

Even for concept classes that lie outside of these limitations, progress on learning the most expressive of them has been relatively limited. Consider the following example of a concept class that *is* learnable:

Assume we are given data points in the form of n bit binary strings, each one labelled either positive or negative. Furthemore, assume that each data point is classified according to a very simple rule: if a particular subset of its bits are all 1, then the data point is labelled positive, otherwise negative. That is to say, the label of each data point is determined by some unknown conjunction of its n bits (if the unknown conjunction were $x_1x_5x_8$ then every data point with a 1 in position one, five, and eight would be labelled positive). A learning algorithm in this model is subjected to the following task:

- The algorithm first receives some randomly chosen labelled data points. The data points are chosen according to some unknown probability distribution $\mathcal{D}$. The algorithm will choose as many points as are needed to achieve a particular level of accuracy.

- After some computation, the algorithm outputs a hypothesis which should be a good approximation to the unknown concept (in this case conjunction) and thus correctly classify future data points chosen according to $\mathcal{D}$.

The task at hand here, to learn an unknown conjunction, is relatively simple: every time we see a data point $(x_1, \ldots, x_n) \in \{0,1\}^n$ labelled *positive*, we know that no variable x_i equal to 0 in this data point could be in our unknown conjunction.

Our candidate hypothesis starts as the conjunction of all variables, and, after seeing some labelled examples, we output the conjunction of all variables that have not yet been ruled out. One can prove that after receiving a relatively small number of random examples, with high probability, we will output a conjunction that is approximately equal to the unknown conjunction and hence correctly classify future random examples.

At this stage we should make two points. First, the algorithm may get "unlucky" and see a set of labelled examples that are not representative of the unknown concept. For example, seeing the same data point over and over again will not give much information about the unknown concept. Therefore we allow the learning algorithm some small probability of failure. Secondly, this model does not require that the hypothesis output by the algorithm correctly classify *all* remaining examples. We only require that it can approximate the unknown concept to within ϵ for any $\epsilon > 0$. As such we allow the algorithm to run in time proportional to $1/\epsilon$. For example, the algorithm above may not output the unknown conjunction, but it can be shown to output a conjunction which is approximately equal (with respect to $\mathcal{D}$) to the unknown conjunction. It is for these two reasons that Valiant's model is called the PAC model or Probably Approximately Correct model of learning.

To continue, assume now that each data point is labelled according to the union of several unknown conjunctions (as opposed to simply one conjunction). These concepts are called DNF or Disjunctive Normal Form formulae. For example, if the unknown DNF is $x_1x_5x_8$ OR $x_2x_3x_6$ then data points with ones in positions one, five, and eight OR points with ones in positions two, three, and six get labelled positive.

Perhaps surprisingly, the complexity of learning this more complicated concept is still unknown. It is a major open problem (perhaps *the* major open problem of computational learning theory) to develop a polynomial time algorithm for learning DNF formulae.

This phenomena, the tremendous increase in the difficulty of learning a concept built from innocent primitives, is not uncommon in computational learning theory. Another famous example is the intersection of halfspaces problem:

Assume that a data point $(x_1, \ldots, x_n) \in \{0,1\}^n$ is labelled according to whether it lies on the "positive" side of an unknown halfspace. If, for example, the unknown halfspace is $3x_1 - 2x_2 + x_3 \geqslant 0$, then all data points satisfying this inequality get labelled positive and all others are labelled negative. Learning an unknown halfspace, as this problem is called, is a classical problem from mathematics whose first solution[1] dates back to work done by Rosenblatt [98], Block [17], Novikoff [93] and others in the 1950s and early 60s and has many applications.

If, however, we let our concept be the intersection of a number of halfspaces (a data point is labelled positive if and only if it satisfies several inequalities) the problem becomes much harder. Very little is known about the complexity of this problem and learning such a concept class would be a major breakthrough– intersections of halfspaces capture a myriad of interesting structures for encoding information (we will show later that this problem can be viewed as a generalization of the DNF learning problem).

1.1 Our Contribution

In this thesis we make the first significant progress on these and other fundamental computational learning problems in years. For example, we give a $2^{\tilde{O}(n^{1/3})}$ time algorithm for learning DNF formulae (the previous best being $2^{\tilde{O}(n^{1/2})}$) and the first polynomial time algorithm for learning the intersection of a constant number of halfspaces under the uniform distribution (to within any constant error parameter).

We believe that our results enrich the literature of computational learning theory and demonstrate that learning non-trivial concept classes is still an achievable goal. Our approach to designing new learning algorithms comes primarily from the field of computational complexity theory. We believe that computational complexity provides a unique vantage point for understanding the structure of particular concept classes and allows the learning theorist to design new, powerful algorithms.

[1] The first true PAC learning algorithm is due to Blumer et al. [25] in 1989.

1.2 A Complexity Theoretic Approach

One of the great triumphs of complexity theory has been the establishment of novel representations for traditional problems via an elaborate set of reductions. For example, the notion of interactive proofs (as defined by Goldwasser, Micali, and Rackoff [51] and Babai and Moran [8]) revolutionized the way we think of a mathematical proof and has led to spectacular results in theoretical computer science, most notably the work on Probabilistically Checkable Proofs and their application to showing the inapproximability of NP-hard optimization problems [4]. Work concerning the role of randomness in computation has led to startling new algorithms for all sorts of fundamental problems [89], and the closely related field of derandomization has provided new explicit constructions of well-studied combinatorial objects [110].

In this thesis we show that well known structural ideas usually associated with the complexity theoretic literature can be used to create new algorithms for problems in computational learning theory that had heretofore remained intractable. More specifically we have found that analyzing certain concept classes such as DNF formulae or intersections of halfspaces in terms of their computational complexity leads to simple learning algorithms.

We also emphasize that there are close relationships between seemingly disparate techniques from each field. As such, we obtain new results in complexity theory by applying methods from computational learning. For example, our work on learing DNF implies that the 30 year old lower bound due to Minsky and Papert [88] on the degree of a polynomial representing a DNF formulae is tight. We also show how to use tools from learning theory to give improved constructions of pseudo-random generators, mathematical objects which play a fundamental role in cryptography and derandomization.

1.3 Complexity Theoretic Tools

Our main complexity theoretic tools fall into three categories:

- **Representing Boolean Functions as Polynomials**

  Let $f(x_1, \ldots, x_n)$ be a Boolean function and $p(x_1, \ldots, x_n)$ a polynomial over the Reals. If, for every $x_1, \ldots, x_n \in \{0,1\}^n$, the value of $f(x_1, \ldots, x_n)$ is equal to the *sign* of $p(x_1, \ldots, x_n)$ we say that f is computed by a polynomial threshold function p. Complexity theorists have for years studied how well polynomials can represent boolean functions. Of particular interest are questions such as "What is the lowest degree polynomial threshold function that can compute or at least approximate a particular family of Boolean functions?" and "Does a polynomial require large coefficients to compute a particular function?" Results related to these questions have applications to many areas of complexity theory including lower bounds, proof checking, error correction and derandomization. Here we consider how well polynomial threshold functions can compute certain concept classes. For example, what is the lowest degree polynomial threshold function that can compute a polynomial-size DNF formula? We will see that answers to these questions have profound learning theoretic implications.

- **Boosting and Hard Core Set Construction**

  We establish a relationship between a fundamental technique from computational complexity theory, *hard-core set construction*, and computational learning, *boosting*. Informally speaking, boosting algorithms convert learning algorithms which give a weak approximation to the underlying scheme or concept class into an algorithm which can strongly approximate the concept class. It turns out that these types of algorithms also appear in the complexity theory literature as hard-core set constructions, a method of amplifying the intractability of boolean functions (which is then useful for building pseudo-random generators). Exploiting this relationship allows us to develop faster algorithms for boosting based results in computational learning theory.

- **Fourier Analysis and Noise Stability of Boolean Functions**

  Examining the Fourier spectrum of Boolean functions has been a useful technique in many fields of theoretical computer science including complexity theory

[55]. Applying Fourier analysis to a concept class has been a standard technique in computational learning theory for years (see for example [81, 61]). If we know, for example, that a concept has almost all of its Fourier weight concentrated on just a few coefficients, then we can learn the concept by approximating these few coefficients. In this thesis we go further and combine Fourier analysis with other techniques such as boosting and noise stability, tools which are complexity theoretic in nature, to give new characterizations for well known concept classes such as Boolean circuits and intersections of halfspaces. This, in turn, leads to signficant progress in developing learning algorithms for these well studied objects.

In this thesis we apply these complexity theoretic tools, sometimes in combination with each other, to achieve new results for old learning problems. We will see that viewing these learning problems through a complexity theoretic lens leads to surprisingly simple solutions. In fact, the actual learning algorithms we present in this thesis are by themselves not new. For example, we use, as a "black-box," well known polynomial time algorithms for solving linear programs and for estimating the Fourier spectra of Boolean functions. What is new is the fact that these well known algorithms are useful for learning a much broader range of concept classes than was once thought. Only by giving new representations of concept classes via complexity theoretic reductions have we discovered this.

1.4 Structure of the Thesis

- **Chapter 2 – Preliminaries**

  In this chapter we give a brief introduction to Valiant's model and a history of important results.

- **Chapter 3 – Learning DNF in Time $2^{\tilde{O}(n^{1/3})}$**

  We show that any s-term DNF over n variables can be computed by a polynomial threshold function of degree $O(n^{1/3} \log s)$. This upper bound matches, up to a logarithmic factor, the longstanding lower bound given by Minsky and Papert in their 1968 book *Perceptrons*. This structural result reduces the problem of learning polynomial size DNF to the problem of learning a linear halfspace with at most $2^{\tilde{O}(n^{1/3})}$ terms. As an immediate consequence we obtain the fastest known DNF learning algorithm which runs in time $2^{\tilde{O}(n^{1/3})}$.

This chapter is joint work with Rocco Servedio.

- **Chapter 4 – Boosting and Hard-Core Sets**

  Boosting algorithms, as mentioned earlier, transform learning algorithms which are only marginally good at prediction into algorithms which can make strong, accurate predictions. It turns out that hard-core set construction, introduced and used by Impagliazzo as a tool for amplifying the intractability of a Boolean function, actually *is* a boosting algorithm. By exploiting this connection, we can achieve the fastest known algorithm for learning DNF formulae in a well studied variant of Valiant's model (in this variant, which we describe more in chapter 4, the learner is able to query the formula at points of his/her choosing and is only required to learn when the random examples of data are chosen according to a uniform distribution). The relationship we establish is fruitful in both directions: applying known boosting algorithms gives improved hard-core set constructions.

  This chapter is joint work with Rocco Servedio.

- **Chapter 5 – Learnability Beyond AC^0**

  In 1989 Linial, Mansour, and Nisan gave a remarkable Fourier based quasipolynomial time algorithm for learning the class AC^0 of constant-depth, polynomial-size Boolean circuits (a DNF is a depth-2 AC^0 circuit). We give an algorithm to learn constant-depth polynomial-size circuits augmented with majority gates under the uniform distribution using random examples only. For circuits which contain a polylogarithmic number of majority gates the algorithm runs in quasipolynomial time. This is the first algorithm for learning a more expressive circuit class than the class AC^0 of constant-depth polynomial-size circuits since the Linial, Mansour, and Nisan result. Furthermore, we can also show, under a suitable cryptographic assumption, that these are the most expressive circuits which will admit a non-trivial learning algorithm. Our approach combines an extension of some of the Fourier analysis from Linial *et al.* with hypothesis boosting.

  This chapter is joint work with Jeff Jackson and Rocco Servedio.

- **Chapter 6 – New Algorithms for Learning Intersections of Halfspaces**

  We give the first polynomial time algorithm to learn any function of a constant number of halfspaces under the uniform distribution to within any constant error parameter. As a corollary we can learn any neural network in which there are k bottom level gates that the inputs feed into in time $n^{O(k^2/\epsilon^2)}$. We also give the first quasipolynomial time algorithm for learning any function of a constant number of halfspaces with polynomial bounded weights under *any* distribution. The only previous result for this problem is due to Baum [9] who gave a polynomial time algorithm for learning the intersection of two halfspaces with respect to any symmetric distribution. Our techniques involve analyzing the *noise stability* of Boolean functions to give a novel Fourier based algorithm for learning linear threshold functions. In addition we give new representations of intersections of halfspaces as polynomial threshold functions.

  This chapter is joint work with Ryan O'Donnell and Rocco Servedio.

- **Chapter 7 – Conclusions and Open Problems**

  We propose directions for further research and state several interesting open problems.

A treatment of Chapters 3 and 4 appears in Rocco Servedio's Ph.D. thesis [106].

Chapter 2

Preliminaries

In this chapter we recall Valiant's PAC model of learning and highlight some important algorithms related to his model. For a more comprehensive introduction to computational learning theory we refer the reader to [68]. For background regarding computational complexity theory such as definitions of circuits and other models of computation or a treatment of NP-hardness we refer the reader to [95].

2.1 Definitions

Here we give a brief definition of Valiant's PAC model. The goal of the learner is to output an approximation to an unknown Boolean function chosen from some concept class. The learner is allowed to see labeled examples from the unknown function via an *example oracle* which gives the learner randomly chosen input/output pairs chosen according to some distribution $\mathcal{D}$ and labeled according to the unknown function. The learner's task is to then output a hypothesis which is approximately equal to the unknown function. The learner is also allowed some probability of failure corresponding to a case where "unhelpful" examples are retrieved from the example oracle. The accuracy of the hypothesis is measured by the ϵ parameter, and the probability of failure is measured by the δ parameter. A more formal treatment follows.

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function and $\mathcal{D}$ a probability distribution on

$\{0,1\}^n$. We write $EX(f,\mathcal{D})$ to denote a random example oracle which, when invoked, outputs a labeled example $\langle x, f(x)\rangle$ where $x \in \{0,1\}^n$ is chosen according to $\mathcal{D}$.

A concept class $C = \cup_{n\geqslant 1} C_n$ is a collection of Boolean functions where each $f \in C_n$ is a Boolean function on $\{0,1\}^n$. For every concept class C we assume there exists an agreed upon encoding of elements of the class and size(f) denotes the length of the encoding of f in C. Let $f \in C_n$ be an unknown member of C and let A be a learning algorithm which takes as input n and accuracy and confidence parameters ϵ, δ and can invoke the oracle $EX(f,\mathcal{D})$.

We say that A *learns* C *with respect to* $\mathcal{D}$ if A satisfies the following condition: for all $n \geqslant 1$ and $0 < \epsilon, \delta < 1$, if A is given access to $EX(f,\mathcal{D})$ then with probability at least $1-\delta$ A outputs a hypothesis $h : \{0,1\}^n \rightarrow \{0,1\}$ such that $\Pr[f(x) = h(x)] \geqslant 1-\epsilon$. We say that A runs in time t if A takes at most t time steps to output h and for any input x, $h(x)$ can be evaluted in time t.

If A can learn C with respect to any distribution $\mathcal{D}$ in time polynomial in $\epsilon^{-1}, \delta^{-1}, n$, and size$(f)$ then we say that C is PAC learnable. If A succeeds when $\mathcal{D}$ is the uniform distribution on $\{0,1\}^n$ (denoted $\mathcal{U}$) then we say that C is learnable under the uniform distribution in polynomial time, and we say that A is a polynomial time uniform distribution learning algorithm.

A natural way to augment the power of the learner is to allow it *membership queries*. An algorithm can make membership queries if it is allowed to query the unknown function on an input *of its own choosing.* It seems that membership queries aid the learner significantly– it is known how to learn DNF formulae and intersections of halfspaces in polynomial time if membership queries are allowed (see [78, 62]).

We will also consider *weak* learning algorithms which work for a broader range of distributions but can only generate hypotheses which are slightly more accurate than random guessing. For $\gamma > 0$ we say that A is a *weak learning algorithm for* C *with advantage* γ if A satisfies the following condition: for any distribution $\mathcal{D}$, for any $0 < \delta < 1$, if A is given access to $EX(f,\mathcal{D})$ then with probability at least $1-\delta$ A outputs a hypothesis $h : \{0,1\}^n \rightarrow \{0,1\}$ such that $\Pr_{\mathcal{D}}[f(x) = h(x)] \geqslant 1/2 + \gamma$. As we will see the advantage γ can depend on parameters such as n, the size of a circuit for f, and the L_∞ norm of the distribution $\mathcal{D}$.

2.2 A History of Some Results

Here we mention some algorithms which learn the most expressive or interesting concept classes to date. We do not give definitions of several concept classes here. They can be found either in the appropriate chapter (for example an extensive treatment of DNF formulae can be found in Chapter 3) or in the Ph.D. thesis of Michael Kearns [65] which has an encyclopedic description of various well studied concept classes.

Unfortunately very few concept classes are known to be PAC learnable (in polynomial time). We state two results where the learning algorithm runs in superpolynomial but still subexponential time and learns with respect to any distribution. Ehrenfeucht and Haussler [41] and Blum [18] have given $n^{\log s}$ time algorithms for learning size-s decision trees over n Boolean variables. Improving on results of Bshouty [28] and Tarui and Tsukiji [109], Klivans and Servedio [74] (see Chapter 3) have given an algorithm for learning s-term Disjunctive Normal Form (DNF) formulae in time $2^{O(n^{1/3} \log s \log n)}$.

There are many more learning results when considering only uniform distribution learning algorithms. For example, Jackson has given an algorithm called the Harmonic Sieve [62] which learns s-term DNF formulae in time polynomial in n and s but requires that the learning algorithm be allowed to make membership queries. Linial *et al.* [81] have given an algorithm which learns size-s depth-d AND/OR/NOT circuits in time $n^{(\log s)^d}$ without membership queries. Servedio [104] has shown how to learn monotone $2^{O(\sqrt{\log n})}$-term DNF formula in polynomial time under the uniform distribution.

In work due to Blum *et al.* [19], an algorithm is given to weakly learn an arbitrary monotone Boolean function under the uniform distribution. They give a polynomial time algorithm which is guaranteed to have error at most $1/2 - \Omega(1/\sqrt{n})$.

The above algorithms do not output hypotheses which belong to the same concept class as the unknown function being learned. Algorithms which do output a hypothesis from the same concept class as the unknown function are known as proper learning algorithms. Very strong negative results are known with respect to proper learning. For example, Pitt and Valiant prove that learning k-term DNF by outputing a hypothesis which is a k-term DNF is NP-hard [97]. As such, interesting learning algorithms rely heavily on giving alternate representations for functions from a given concept class. Limitations on what is learnable under a cryptographic assumption were first pointed out in work due to Kearns and Valiant [67], and we give a more detailed description of this in Chapter 5.

Chapter 3

Learning DNF in Time $2^{\tilde{O}(n^{1/3})}$

3.1 Introduction

3.1.1 Learning DNF

One of the outstanding problems in computational learning theory is the problem of learning DNF formulae. DNF are attractive from a learning theory perspective because of their high expressive power (any Boolean function can be represented as a DNF) and because they seem to be a natural form of knowledge representation for humans. Valiant first posed the question of whether DNF are efficiently learnable in his seminal 1984 paper introducing the PAC learning model [111]. More than fifteen years later this question is widely regarded as one of the most important open problems in learning theory. While many partial results have been given for restricted versions of the DNF learning problem (see e.g. [22, 24, 56, 62, 69, 75, 79, 99, 113, 114]), the difficulty of the unrestricted DNF learning problem is evidenced by the fact that, prior to the current work, only two algorithms were known which improve on the naive 2^n time bound [28, 109]. Our work on learning DNF formulae will make critical use of polynomial threshold functions.

3.1.2 Polynomial Threshold Functions

Let f be a Boolean function $f : \{0,1\}^n \to \{-1,1\}$ and let p be a degree d polynomial in n variables with rational coefficients. If the sign of $p(x)$ equals $f(x)$ for every $x \in \{0,1\}^n$, then we say that f is computed by a *polynomial threshold function* of degree d. In their well known 1968 book *Perceptrons*, Minsky and Papert studied some computational aspects of polynomial threshold functions from an Artificial Intelligence perspective [88]. They proved, among other things, that no polynomial threshold function of degree less than n can compute the parity function on n variables, and that there is a read-once DNF formula which cannot be computed by any polynomial threshold function of degree less than $\Omega(n^{1/3})$. Since then, complexity theorists have used these and related properties of polynomial threshold functions to prove several important results in both circuit and structural complexity [5, 10, 16, 48, 115].

In the computational learning theory community, learning a polynomial threshold function from labeled examples has long been a central problem and continues to be an active area of research. A special focus of attention has been directed toward learning polynomial threshold functions of degree 1, which are known as *linear threshold functions*. The problem of learning a linear threshold function over $\{0,1\}^n$ can be formulated as a linear programming problem and thus can be solved in poly(n) time in both the PAC model of learning from random examples and in the model of exact learning from equivalence queries [25, 86]. Refinements of the basic linear programming approach have led to polynomial-time algorithms for PAC learning linear threshold functions in the presence of classification noise [21, 35]. Much attention has also been given to fast, simple heuristics, most notably the Winnow and Perceptron algorithms, for learning linear threshold functions [33, 47, 70, 83, 102, 103].

The first subexponential time algorithm for learning DNF was due to Bshouty [28], who gave an algorithm which learns any s-term DNF over n variables in time $2^{O((n \log s)^{1/2} \log^{3/2} n)}$. At the heart of Bshouty's algorithm is a structural result which shows that any s-term DNF can be expressed as an $O((n \log n \log s)^{1/2})$-decision list; armed with this result, Bshouty uses a standard algorithm [57] for learning decision lists to obtain his DNF learning result.

Tarui and Tsukiji [109] gave a completely different proof of a similar time bound for learning DNF. They adapted the machinery of "approximate inclusion/exclusion" developed by Linial and Nisan [82] to show that for any s-term DNF f and any distribution $\mathcal{D}$ over $\{0,1\}^n$, there is a conjunction C of size $O(n^{1/2}\log s)$ which has $|\Pr_{x\in\mathcal{D}}[C(x)=f(x)]-\frac{1}{2}| = 2^{-O(n^{1/2}\log n\log s)}$. Using this result in conjunction with Freund's "boost-by-majority" algorithm [45], Tarui and Tsukiji obtained an algorithm for learning s-term DNF in time $2^{O(n^{1/2}\log n\log s)}$.

3.1.3 A New Approach: Learning DNF via Polynomial Threshold Functions

In this paper we approach the DNF learning problem by representing a DNF formula as a low-degree polynomial threshold function. As we observe in Section 2, we can use known polynomial-time algorithms for learning linear threshold functions to learn polynomial threshold functions of degree d in time $n^{O(d)}$. Thus, upper bounds on the degree of polynomial threshold functions which compute DNF translate directly into bounds on the running time of a DNF learning algorithm.

Viewing DNF formulae as polynomial threshold functions immediately yields a new interpretation of the DNF learning algorithms of Bshouty [28] and Tarui and Tsukiji [109]. Since any r-decision list is equivalent to a polynomial threshold function of degree r [42], in the language of polynomial threshold functions Bshouty's structural result implies that any s-term DNF can be expressed as a polynomial threshold function of degree $O((n\log n\log s)^{1/2})$. In the case of Tarui/Tsukiji, it can be shown as a corollary of their results that any s-term DNF can be expressed as a polynomial threshold function of degree $O(n^{1/2}\log s)$. Thus, each of these earlier learning algorithms implies an $O(n^{1/2}\log n)$ upper bound on the degree of a polynomial threshold function for any polynomial-size DNF. A substantial gap remained, though, between these $O(n^{1/2}\log n)$ upper bounds and the $\Omega(n^{1/3})$ lower bound due to Minsky and Papert.[1]

3.1.4 Our Results

Our first result is the following theorem:

Theorem 3.1.1 *Any s-term DNF over $\{0,1\}^n$ in which each conjunction is of size at most t can be expressed as a polynomial threshold function of degree $O(t^{1/2}\log s)$.*

A useful feature of Theorem 5.1.1 is that the degree bound depends on $t^{1/2}$ which can be much smaller than $n^{1/2}$. Close inspection of the results due to Tarui/Tsukiji reveal that a similar theorem can be derived from their analysis. An advantage of our proof (which is self-contained and does not use approximate inclusion-exclusion or boosting) is that it highlights this dependence which plays a crucial role in our later results.

We then use Theorem 1 to give several new results about the degree of polynomial threshold functions which compute various classes of Boolean formulas.

By combining Theorem 1 with a decomposition technique due to Bshouty [28] we obtain our main result:

Theorem 3.1.2 *Any s-term DNF over $\{0,1\}^n$ can be expressed as a polynomial threshold function of degree $O(n^{1/3}\log s)$.*

Theorem 5.1.2 essentially closes the gap which was left open by the $O(n^{1/2}\log n)$ upper bounds implicit in [28, 109]; it shows that the Minsky-Papert lower bound is in fact tight, up to a logarithmic factor, for *all* polynomial-size DNF. Theorem 3.1.3 also yields a $2^{O(n^{1/3}\log^2 n)}$-time algorithm for learning polynomial-size DNF, which improves on the algorithms of Bshouty and Tarui/Tsukiji and is the fastest known algorithm for the unrestricted DNF learning problem.

We can improve upon the bounds of Theorem 5.1.2 for read-once DNF:

[1]Beigel et al. stated in [13] that Minsky and Papert gave an $\Omega(n^{1/2})$ lower bound for DNF but this was in error [12].

Theorem 3.1.3 *Any read-once DNF over $\{0,1\}^n$ can be expressed as a polynomial threshold function of degree $O(n^{1/3}\log^{2/3} n)$.*

Finally, since a DNF formula is simply a Boolean circuit of depth 2, it is natural in this context to also consider Boolean circuits of fixed depth $d \geqslant 3$. We would ultimately like to prove upper bounds on the degree of polynomial threshold functions for arbitrary constant depth circuits; however this seems to be quite difficult. As a first step in this direction, we prove

Theorem 3.1.4 *For $d \geqslant 3$, any read-once Boolean formula of depth d over $\{\wedge, \vee, \neg\}$ can be computed by a polynomial threshold function of degree $\tilde{O}(n^{1-\frac{1}{3\cdot 2^{d-3}}})$.*

Theorem 3.1.4 implies that the class of read-once constant depth formulas can be learned in subexponential time.

3.2 Preliminaries

3.2.1 DNF, Decision Lists, Decision Trees, and Polynomial Threshold Functions

A *disjunctive normal form* formula or DNF is a disjunction $T_1 \vee \cdots \vee T_s$ of conjunctions of Boolean literals. An s-term DNF is one which has at most s conjunctions (also known as terms) and a t-DNF is one in which each term is of size at most t. A DNF (or Boolean formula) is *read-once* if it contains at most one occurrence of each variable.

A *k-decision list* is a list $L = (T_1, f_1), \ldots, (T_m, f_m)$ where each T_i is a term of size at most k and each f_i is a Boolean function on $\{0,1\}^n$. Given an input $x \in \{0,1\}^n$ the value of $L(x)$ is $f_j(x)$ where $j \geqslant 1$ is such that $T_j(x) = 1$ and $T_i(x) = 0$ for $i < j$. If $T_i(x) = 0$ for all $1 \leqslant i \leqslant m$ then $L(x) = 1$.

A *k-decision tree* is a rooted binary tree where each internal node has 2 children and is labeled with a term of size at most k and each leaf is labeled with a Boolean function. A decision tree represents a Boolean function as follows: if the root is labeled with a term T then then to compute the value of the tree on an input $x \in \{0,1\}^n$ we go left from the root if $T(x) = 0$ and go right if $T(x) = 1$. We continue in this fashion until reaching a leaf ℓ labeled with some function f_ℓ and then output $f_\ell(x)$.

The *rank* of a decision tree T is defined inductively as follows:

- If T is a single leaf then $rank(T) = 0$.
- If T has subtrees T_0 and T_1 then

$$rank(T) = \begin{cases} \max\left(rank(T_0), rank(T_1)\right) & \text{if } rank(T_0) \neq rank(T_1) \\ rank(T_0) + 1 & \text{otherwise.} \end{cases}$$

The following lemma will be useful:

Lemma 3.2.1 *[18] Let f be computed by a 1-decision tree of rank r whose leaves are labeled with the functions $f_1, \ldots, f_m$. Then there is an r-decision list $(T_1, f_1), \ldots, (T_m, f_m)$ which is equivalent to f.*

A *polynomial threshold function* is defined by a real-valued multivariate polynomial $p(x_1, \ldots, x_n)$. The output of the polynomial threshold function on input $x \in \{0,1\}^n$ is 1 if $p(x_1, \ldots, x_n) \geqslant 0$ and is -1 otherwise. The *degree* of a polynomial threshold function is simply the degree of the polynomial p. If each coefficient a_α of the polynomial is an integer, then the *weight* of the polynomial threshold function is $\sum |a_\alpha|$.

3.2.2 Learning theory background

We consider two widely studied learning models: the *Probably Approximately Correct* (PAC) model introduced by Valiant [111] and the model of *exact learning from equivalence queries* introduced by Angluin [2] and Littlestone [83]. In each of these models a *concept class* C is a collection of Boolean functions $c : \{0,1\}^n \to \{-1,1\}$.

In the PAC model, for Boolean functions c, h on $\{0,1\}^n$ and $\mathcal{D}$ a distribution on $\{0,1\}^n$, we say that h is an *ϵ-approximator for c under $\mathcal{D}$* if $\Pr_{\mathcal{D}}[c(x) = h(x)] \geqslant 1 - \epsilon$. The learning algorithm has access to an *example oracle* $EX(c, \mathcal{D})$ which, when queried, provides a labeled example $\langle x, c(x) \rangle$ where x is drawn from $\{0,1\}^n$ according to the distribution $\mathcal{D}$ and $c \in C$ is the unknown target concept which the algorithm is trying to learn. The goal of the learner is to generate an ϵ-approximator for c under $\mathcal{D}$. An algorithm A is a *PAC learning algorithm for a concept class C* if the following condition holds: for any $c \in C$, any distribution $\mathcal{D}$ on $\{0,1\}^n$, and any $0 < \epsilon, \delta < 1$, if A is given ϵ and δ and has access to $EX(c, \mathcal{D})$, then with probability at least $1 - \delta$ algorithm A outputs an ϵ-approximator for c under $\mathcal{D}$. We say that *A PAC learns C in time t* if A runs for at most t time steps and outputs a hypothesis h which can be evaluated on any point $x \in \{0,1\}^n$ in time t.

In the model of exact learning from equivalence queries, learning proceeds in a sequence of stages. In each stage the learning algorithm submits an *equivalence query* (a Boolean hypothesis function h) to the teacher. If h is equivalent to the target concept c then the teacher answers "YES" and learning halts; otherwise the teacher sends back a point $x \in \{0,1\}^n$ such that $h(x) \neq c(x)$. A learning algorithm *A learns concept class C in time t* if for all $c \in C$, algorithm A can exactly identify the target c in at most t time steps, using at most t equivalence queries, with hypotheses h which each can be represented with t bits and can be evaluated on any point $x \in \{0,1\}^n$ in time t.

The following fact is well known:

Fact 3.2.2 ([25, 86]) *In both the PAC model and the model of exact learning from equivalence queries, there are algorithms which learn the class of linear threshold functions over $\{0,1\}^n$ in time poly(n).*

The algorithms of Fact 6.5.5 are based on polynomial time algorithms for linear programming. We will need the following extension of Fact 6.5.5:

Fact 3.2.3 *Let C be a class of functions each of which can be expressed as an degree-d polynomial threshold function over $\{0,1\}^n$. Then in both the PAC learning model and the model of exact learning from equivalence queries, there is a learning algorithm for C which runs in time $n^{O(d)}$.*

Proof: For the model of exact learning from equivalence queries, we run the polynomial time algorithm for learning linear threshold functions over an expanded version of the input space. Since $z^2 = z$ for $z \in \{0,1\}$ we can suppose without loss of generality that the target polynomial threshold function is a multilinear polynomial of degree d. Such a polynomial threshold function can be viewed as a linear threshold function over the space of all multilinear monomials of degree at most d. There are $N = \sum_{i=1}^{d} \binom{n}{i} \leqslant n^d$ such monomials and hence by Fact 6.5.5 we can learn such a polynomial threshold function in the model of exact learning from equivalence queries by running a poly(N)-time algorithm for learning linear threshold functions over the domain $\{0,1\}^N$ where $N \leqslant n^d$.

For the PAC learning model, we can use a standard transformation from the model of exact learning from equivalence queries to the PAC learning model [2]. It is easy to verify that this transformation (in which equivalence queries are simulated by testing the hypothesis against a set of examples drawn from the PAC oracle) preserves running time up to polynomial factors. Alternatively, using the fact [3] that the Vapnik-Chervonenkis dimension of the class of polynomial threshold functions of degree d over n variables is $\sum_{i=0}^{d} \binom{n}{d}$, a direct proof can be given along the lines of [25]. ■

3.2.3 The Minsky Papert Lower Bound

It is clear that any depth-1 circuit over $\{\wedge, \vee, \neg\}$ can be expressed as a linear threshold function. In contrast, Minsky and Papert gave a $\Omega(n^{1/3})$ lower bound on the degree of any polynomial threshold function which computes a particular read-once DNF. For completeness we give their simple proof.

Theorem 3.2.4 (Minsky & Papert [88]) *Let $f = T_1 \vee \cdots \vee T_m$ be an m-term DNF over $\{0,1\}^n$ where each term T_i is a conjunction over $4m^2$ variables, each variable appears in precisely one term, and $n = 4m^3$. Then any polynomial threshold function which computes f must have degree at least m.*

Proof: Let $p(x_1, \ldots, x_n)$ be a polynomial of degree d such that for all $x \in \{0,1\}^n$ we have $p(x) \geqslant 0$ iff x satisfies f. For $i = 1, \ldots, m$ let S_i be the set of $4m^2$ variables which appears in term T_i. It is clear that for any permutations $\pi_1, \ldots, \pi_m$ over a set of size $4m^2$, we have $p(S_1, \ldots, S_m) \geqslant 0$ iff $p(\pi_1(S_1), \ldots, \pi_m(S_m)) \geqslant 0$. Consequently the polynomial

$$q(x_1, \ldots, x_n) = \sum_{\pi_1, \ldots, \pi_m} p(\pi_1(S_1), \ldots, \pi_m(S_m))$$

is of degree at most d and has $q(x_1, \ldots, x_n) \geqslant 0$ iff x satisfies f. Since $q(x)$ is symmetric in the elements of each set S_i, one can straightforwardly show that there is a polynomial $r(\sum_{S_1} x_j, \cdots, \sum_{S_m} x_j)$ of degree at most d such that $r(\sum_{S_1} x_j, \ldots, \sum_{S_m} x_j) = q(x_1, \ldots, x_n)$ for all $x \in \{0,1\}^n$. It follows from the definition of f that for all $(a_1, \ldots, a_m) \in \{0, 1, \ldots, 4m^2\}^m$, we have $r(a_1, \ldots, a_m) \geqslant 0$ iff some $a_i = 4m^2$. Let $s(t)$ be the univariate polynomial $r(a_1(t), \ldots, a_m(t))$ where $a_i(t) = 4m^2 - (2i - 1 - t)^2$ for $i = 1, \ldots, m$. Then the degree of s is at most $2d$, and moreover we have that $s(0), s(2), s(4), \ldots, s(2m) < 0$ and $s(1), s(3), \ldots, s(2m-1) \geqslant 0$. Consequently s has at least $2m$ real zeros, so $2d \geqslant \deg(s) \geqslant 2m$. ■

3.3 An Optimal Bound for Representing DNF by Polynomial Threshold Functions

In this section we prove our main result: any s-term DNF over $\{0,1\}^n$ can be computed by a polynomial threshold function of degree $O(n^{1/3} \log s)$.

3.3.1 Low-Degree Polynomial Threshold Functions for DNF with Small Terms

We start by proving Theorem 1:

Theorem 1 *Any s-term t-DNF can be expressed as a polynomial threshold function of degree $O(t^{1/2} \log s)$.*[2]

This theorem plays an important role in the proof of the main result. We discuss some other consequences of Theorem 5.1.1 in Section 3.4.

[2]We analyze the size of the coefficients of the polynomial threshold function in Section 3.4.

The main tools used in the proof are the Chebyshev polynomials of the first kind. These polynomials play an important role in numerical analysis and approximation theory; here we will need only a few simple properties. The d-th Chebyshev polynomial of the first kind, $C_d(x)$, is a univariate degree-d polynomial which satisfies the following conditions [34]:

Fact 3.3.1 *The polynomial $C_d(x)$ satisfies*

- $|C_d(x)| \leqslant 1$ *for* $|x| \leqslant 1$ *with* $C_d(1) = 1$;
- $C_d'(x) \geqslant d^2$ *for* $x > 1$ *with* $C_d'(1) = d^2$.

Proof of Theorem 5.1.1: Let $f = T_1 \vee T_2 \vee \cdots \vee T_s$ be an s-term t-DNF. The *arithmetization* of a Boolean literal ℓ is x_j if $\ell = x_j$ and is $1 - x_j$ if $\ell = \overline{x}_j$. Let S_i denote the sum of the arithmetizations of the literals appearing in T_i and let t_i denote the number of literals in T_i. We define the polynomial

$$Q_i(x) = p\left(\frac{S_i}{t_i}\right)$$

where

$$p(y) = C_d\left(y\left(1 + \frac{1}{t}\right)\right).$$

Here C_d is the d-th Chebyshev polynomial of the first kind and $d = \lceil t^{1/2} \rceil$. Fact 3.3.1 now implies that $p(1) \geqslant 2$ but $|p(y)| \leqslant 1$ for $y \in [0, 1 - \frac{1}{t}]$.

Consider the polynomial threshold function "$P(x) \geqslant s + \frac{1}{2}$" where

$$P(x) = \sum_{i=1}^{s} Q_i(x)^{\log 2s}.$$

Since C_d is a polynomial of degree $d = t^{1/2}$ and S_i is a polynomial of degree 1, this polynomial threshold function has degree $t^{1/2} \log 2s$. We will show that this polynomial threshold function computes the DNF f exactly.

Fix any element $x \in \{0,1\}^n$.

- If $f(x) = 0$ then in each term T_i at least one arithmetized literal takes value 0 on x. Thus for each $i = 1, \dots, s$ we have $S_i/t_i \leqslant (t_i - 1)/t_i \leqslant 1 - \frac{1}{t}$ and hence each $|Q_i(x)| \leqslant 1$. Consequently $P(x) \leqslant s$.

- If $f(x) = 1$ then some term T_i must be satisfied by x so $S_i/t_i = 1$. Consequently $Q_i(x) \geqslant 2$ and hence $Q_i(x)^{\log 2s}$ contributes at least $2s$ to $P(x)$. Since $Q_i(x)^{\log 2s} \geqslant -1$ for all i, we have $P(x) \geqslant s + 1$. ∎

3.3.2 From DNF to Decision Trees

Let f be an arbitrary s-term DNF over n variables. As the first step in our construction of a polynomial threshold function for f, we transform f into a 1-decision tree in which each leaf is a DNF with small terms; this is a refinement of a transformation given by Bshouty in [28]. Our original proof gave a bound on the *size* of the resulting decision tree. S. Lokam [84] has observed that a slightly stronger bound can be obtained by considering the *rank* of the decision tree instead. We use Lokam's approach in the following lemma:

Lemma 3.3.2 *Let $f : \{0,1\}^n \rightarrow \{-1,1\}$ be an s-term DNF. For any value $1 \leqslant t \leqslant n$, f can be expressed as a 1-decision tree T where*

- *each leaf of T contains an s-term t-DNF,*

- *T has rank at most $(2n/t)\ln s + 1$.*

Proof of Lemma 3.3.2: Let $T_1, \dots, T_p$ be the terms of f that have size at least t. Since each term T_i contains at least t literals, there must be some variable x_i that occurs (either negated or unnegated) in at least pt/n of these terms. This variable x_i is placed in the root of the decision tree, and the left and right children of x_i will be decision trees for the restrictions $f|_{x_i \leftarrow 0}$ and $f|_{x_i \leftarrow 1}$ respectively. This construction is recursively carried out for each of the functions $f|_{x_i \leftarrow 0}$ and $f|_{x_i \leftarrow 1}$, stopping when a DNF with no terms larger than t is obtained.

It is clear that this recursive procedure generates some 1-decision tree T. Since the function obtained by fixing some subset of variables of an s-term DNF is an s-term DNF, we have that each leaf of T contains an s-term t-DNF.

Let $r(n,p)$ be the maximum (taken over all DNFs f on n variables with p terms having size at least t) rank of the decision tree generated by the above procedure. We bound $r(n,p)$ using the following simple observation: if T_a is a term of f which contains an unnegated (negated) variable x_i ($\overline{x}_i$), then the restriction $f|_{x_i \leftarrow 0}$ ($f|_{x_i \leftarrow 1}$) causes the term T_a to vanish. Since the variable x_i at the root of T occurs in at least pt/n terms of size at least t, for at least one of the bit values $b \in \{0,1\}$ the restriction $f|_{x_i \leftarrow b}$ will be a DNF which has at most $p(1-\frac{t}{2n})$ terms of size at least t. Let T_0 (T_1) denote the subtree of T which corresponds to the restriction $f|_{x_i \leftarrow 0}$ ($f|_{x_i \leftarrow 1}$), and suppose without loss of generality that $f|_{x_i \leftarrow 0}$ is a s-term DNF which has at most $p(1-\frac{t}{2n})$ terms of size at least t. Note that $rank(T_0) \leqslant r(n-1, p(1-\frac{t}{2n}))$ and $rank(T_1) \leqslant r(n-1,p)$. We consider several cases:

- If $rank(T_0) < rank(T_1)$, then $rank(T) = rank(T_1)$ and hence $rank(T) \leqslant r(n-1,p)$.
- If $rank(T_0) > rank(T_1)$, then $rank(T) = rank(T_0)$ and hence $rank(T) \leqslant r(n-1,p(1-\frac{t}{2n}))$.
- If $rank(T_0) = rank(T_1)$, then $rank(T) = rank(T_0) + 1$ and hence $rank(T) \leqslant r(n-1,p(1-\frac{t}{2n})) + 1$.

Thus $r(n,p)$ obeys the recurrence relation $r(n,p) \leqslant r(n-1,p(1-\frac{t}{2n}))+1$. To establish initial conditions for the recurrence relation we consider the case $p = 1$. In this case there is one term in f which contains more than t variables; without loss of generality we suppose that this term is $v_1 v_2 \dots v_\ell$. Then the 1-decision list

$$(\overline{v}_1, f|_{v_1 \leftarrow 0}), \dots, (\overline{v}_\ell, f|_{v_\ell \leftarrow 0})$$

is equivalent to a rank-1 decision tree in which each leaf contains an s-term t-DNF. Hence for any n we have $r(n,1) = 1$.

Solving the recurrence relation for $r(n,p)$ shows that $r(n,p) \leqslant (2n/t)\ln p + 1$. Since $p \leqslant s$ the lemma is proved. ∎

3.3.3 An Optimal Bound for Representing DNF by Polynomial Threshold Functions

Theorem 5.1.2 *Let f be an s-term DNF over n variables. Then f can be expressed as a polynomial threshold function of degree $O(n^{1/3}\log s)$.*

Proof: From Lemma 3.3.2 and Theorem 5.1.1, we know that f can be expressed as a 1-decision tree T of rank $(2n/t)\ln s + 1$ where each leaf contains a polynomial threshold function of degree $O(t^{1/2}\log s)$ (the value of t will be fixed later). From Lemma 3.2.1 we know that this decision tree T can be expressed as an r-decision list where $r = (2n/t)\ln s + 1$ and each output of the decision list is a polynomial threshold function of degree $O(t^{1/2}\log s)$. Call this decision list L.

Let $C_1, \ldots, C_R$ be the conjunctions contained in the successive nodes of L and let $P_1(x), \ldots, P_R(x)$ be the corresponding polynomials for the associated polynomial threshold functions at the outputs, i.e. the polynomial threshold function corresponding to the j-th conjunction C_j computes the function "$P_j(x) \geqslant 0$." If $P_j(x) = 0$ for some $x \in \{0,1\}^n$ then we can replace $P_j(x)$ by $P_j(x) + \delta/2$, where $\delta = \min\{-P_j(x) \ : \ x \in \{0,1\}^n \text{ and } P_j(x) < 0\}$, without changing the function computed by the polynomial threshold function. Now by scaling each P_j by an appropriate multiplicative factor we can suppose without loss of generality that for each $j = 1, \ldots, R$ we have $\min_{x\in\{0,1\}^n} |P_j(x)| \geqslant 1$.

Consider the polynomial

$$Q(x) = A_1\tilde{C}_1(x)P_1(x) + A_2\tilde{C}_2(x)P_2(x) + \cdots + A_R\tilde{C}_R(x)P_R(x). \qquad (3.1)$$

Here $\tilde{C}_j$ is the zero/one valued polynomial which corresponds to the monomial C_j (e.g. if C_j is $x_3\overline{x}_4x_5$ then $\tilde{C}_j(x)$ is $x_3(1-x_4)x_5$). Each value A_j is a positive constant chosen so as to satisfy the following conditions:

$$
\begin{aligned}
A_R &= 1, \\
A_{R-1} &> \max_{x \in \{0,1\}^n} |A_R \tilde{C}_R(x) P_R(x)|, \\
&\vdots \\
A_j &> \max_{x \in \{0,1\}^n} |A_{j+1} \tilde{C}_{j+1}(x) P_{j+1}(x) + \cdots + A_R \tilde{C}_R(x) P_R(x)|. \\
&\vdots \\
A_1 &> \max_{x \in \{0,1\}^n} |A_2 \tilde{C}_2(x) P_2(x) + \cdots + A_R \tilde{C}_R(x) P_R(x)|.
\end{aligned}
$$

Then the polynomial threshold function "$Q(x) \geqslant 0$" computes exactly the same function as the decision list L. To see this, fix an input $x \in \{0,1\}^n$. If j is the index of the first conjunction C_j which is satisfied by x, then $\tilde{C}_1(x) = \tilde{C}_2(x) = \cdots = \tilde{C}_{j-1}(x) = 0$, so the only terms of (1) which make a nonzero contribution to $Q(x)$ are $A_i \tilde{C}_i(x) P_i(x)$ for $i \geqslant j$. Since $\tilde{C}_j(x) = 1$ and $|P_j(x)| \geqslant 1$, the choice of A_j ensures that the sign of $Q(x)$ will be the same as the sign of $P_j(x)$.

The degree of the polynomial $Q(x)$ is at most $(2n/t)\ln s + 1 + O(t^{1/2}\log s)$. If we take $t = n^{2/3}$ then this value is $O(n^{1/3}\log s)$. ■

Applying Fact 6.5.6 gives our main DNF learning result:

Corollary 3.3.3 *The class of s-term DNF formulae over $\{0,1\}^n$ can be learned (in both the PAC model and the model of exact learning from equivalence queries) in time $2^{O(n^{1/3}\log s \log n)}$.*

Remark: Several algorithms are known [21, 35] for PAC learning linear threshold functions over $\{0,1\}^n$ in the presence of classification noise in time poly(n). It follows that our time bounds for learning DNF continue to hold in the presence of classification noise.

Corollary 3.3.4 *The $\Omega(n^{1/3})$ lower bound given by Minsky and Papert for the degree of a polynomial threshold function required to compute a polynomial size DNF over $\{0,1\}^n$ is tight up to a logarithmic factor.*

3.4 Discussion

Since $t \leqslant n$ in Theorem 5.1.1, Fact 6.5.6 implies that there is a linear-programming based algorithm for PAC learning DNF which takes $2^{O(n^{1/2}\log n \log s)}$ time steps. Tarui and Tsukiji gave an identical time bound for a different algorithm based on hypothesis boosting using conjunctions. In this section we note that the proof of Theorem 5.1.1 gives an upper bound on the weight of the resulting polynomial threshold function. This observation can be used to prove correctness of the Tarui/Tsukiji boosting-based algorithm and to show that simpler algorithms such as Winnow or Perceptron can be used to learn $\tilde{O}(n^{1/2})$ degree polynomial threshold functions which compute a DNF (instead of boosting algorithms or algorithms for solving linear programs).

The d-th Chebyshev polynomial $C_d(x) = \sum_{i=0}^{d} a_i x^i$ has all integer coefficients with each $|a_i| \leqslant 2^d$ [34]. By inspection of the proof of Theorem 5.1.1 we obtain

Corollary 3.4.1 *Any s-term t-DNF can be expressed as a polynomial threshold function of degree $O(t^{1/2}\log s)$ and weight $t^{O(t^{1/2}\log s)}$.*

Using this corollary we obtain an easy proof of one of the main theorems from [109], described in Section 3.1.1, which asserts that for any DNF f and any probability distribution $\mathcal{D}$ there exists some short conjunction which is noticeably correlated with f under $\mathcal{D}$. We use a simple lemma due to Goldmann, Hastad and Razborov ([50] Lemma 4) which states that if a function f over $\{0,1\}^n$ can be expressed as a majority of at most W ± 1-valued functions (possibly with repetitions) drawn from a set H, then for any distribution $\mathcal{D}$ over $\{0,1\}^n$ there is some function $h \in H$ such that $|\Pr_{x\in\mathcal{D}}[h(x) = f(x)] - \frac{1}{2}| \geqslant \frac{1}{W}$. In our setting we take H to be the set of all conjunctions of length $O(t^{1/2}\log s)$ and their negations. There is a clear correspondence between polynomial threshold functions with integer coefficients and depth-2 circuits with a MAJORITY gate at the root and (possibly negated) AND gates at depth 1. Corollary 3.4.1 gives the required bound on W, and we obtain

Corollary 3.4.2 *Given any s-term t-DNF f and any distribution $\mathcal{D}$ over $\{0,1\}^n$, there is a conjunction C of size at most $O(t^{1/2}\log s)$ such that $|\Pr_{x\in\mathcal{D}}[C(x) = f(x)] - \frac{1}{2}| = 2^{-O(t^{1/2}\log t \log s)}$.*

Taking $t = n$ gives Tarui and Tsukiji's Theorem 1.1, which immediately implies the existence of a boosting-based algorithm for learning DNF in time $2^{\tilde{O}(n^{1/2})}$.

Finally, we observe that the weight bound given in Corollary 3.4.1 implies that we do not need to solve linear programs (or even to use boosting algorithms) in order to learn polynomial-sized DNF in time $2^{\tilde{O}(n^{1/2})}$. If f is a polynomial threshold function of degree 1 and weight W over the domain $\{0,1\}^N$, then either the Perceptron algorithm or the Winnow algorithm can be used to learn f in $\mathrm{poly}(N, W)$ time steps [70, 83]. As in Fact 6.5.6, we can view an degree-d polynomial threshold function over $\{0,1\}^n$ as a degree-1 polynomial threshold function over $\{0,1\}^{n^d}$, and thus we can in fact use either the Perceptron or Winnow algorithm to learn s-term DNF in time $2^{O(n^{1/2}\log n \log s)}$.

3.5 Low-Degree Polynomial Threshold Functions for Read-Once DNF

As seen in Section 3.2.3 the Minsky-Papert $\Omega(n^{1/3})$ lower bound on polynomial threshold function degree for polynomial size DNF is proved using a read-once DNF. Since any read-once DNF can have at most n terms, Theorem 5.1.2 implies that any read-once DNF can be expressed as a polynomial threshold function of degree $O(n^{1/3}\log n)$. Here we give a slightly better bound:

Theorem 3.1.3 *Any read-once DNF over variables $x_1, \ldots, x_n$ can be expressed as a polynomial threshold function of degree $O(n^{1/3}\log^{2/3} n)$.*

To prove Theorem 3.1.3 we use the following sharper version of Lemma 3.3.2:

Lemma 3.5.1 *Let $f : \{0,1\}^n \to \{-1,1\}$ be a read-once DNF. For any value $1 \leqslant t \leqslant n$, f can be expressed as a 1-decision tree T where each leaf of T contains a read-once t-DNF and T has rank at most n/t.*

Proof of Lemma 3.5.1: Let $T_1, \ldots, T_p$ be the terms of f that have size at least t. We use the same decomposition procedure as in Lemma 3.3.2, and we let $r(n,p)$ be the maximum (taken over all read-once DNFs f on n variables with p terms having size at least t) rank of the decision tree generated by the decomposition procedure. Since each variable occurs in at most one term, the recurrence which we obtain in this setting is $r(n,p) \leqslant r(n-1,p-1)+1$. As before the initial condition is $r(n,1)=1$ for all n, and thus $r(n,p) \leqslant p$. Since f is read-once we have that $p \leqslant n/t$, and the lemma is proved. (Lemma 3.5.1) ■

Proof of Theorem 3.1.3: Let f be a s-term read-once DNF over $\{0,1\}^n$. Lemma 3.5.1, Theorem 5.1.1 and Lemma 3.2.1 together imply that f is computed by a (n/t)-decision list where each output of the decision list is a polynomial threshold function of degree $O(t^{1/2}\log s)$. As in the proof of Theorem 5.1.2 there is a polynomial threshold function for f which is of degree $n/t + O(t^{1/2}\log s)$. Since f is read-once s is at most n, and taking $t = n^{2/3}/\log^{2/3} n$ proves the theorem. (Theorem 3.1.3) ■

By the arguments given in Section 2, we immediately have

Corollary 3.5.2 *The class of read-once DNF can be learned (in both the PAC model and the model of exact learning from equivalence queries) in time $2^{O(n^{1/3}\log^{5/3} n)}$.*

Remark: Standard reductions are known in learning theory which reduce the problem of PAC learning DNF to that of PAC learning read-once DNF. We note that applying these reductions here does *not* yield a $2^{\tilde{O}(n^{1/3})}$-time algorithm for learning arbitrary polynomial-size DNF. The reductions work by converting a DNF with $p(n)$ total occurrences of variables to a read-once DNF over $p(n)$ variables, and thus if used in conjunction with Corollary 3.5.2 would yield a $2^{\tilde{O}(p(n)^{1/3})}$-time algorithm for learning such a DNF.

3.6 Future Work

Many directions remain for further research. From a learning theory perspective, an obvious goal is to construct learning algorithms for DNF which have even lower time complexity than the algorithm of this paper. The Minsky-Papert lower bound implies that our time bounds are essentially optimal for algorithms which work by learning polynomial threshold functions. It would be interesting to close the remaining gap between the Minsky-Papert $\Omega(n^{1/3})$ lower bound and our $O(n^{1/3} \log n)$ upper bound on the degree of polynomial threshold functions for polynomial-size DNF.

Another goal is to establish a bound on polynomial threshold function *weight* to go along with our degree bound from Theorem 3.1.3. Is every polynomial-size DNF computed by a polynomial threshold function of degree $\tilde{O}(n^{1/3})$ and weight $2^{\tilde{O}(n^{1/3})}$? As in Section 3.4, an affirmative answer to this question would mean that the Perceptron or Winnow algorithm could be used instead of a linear programming based algorithm.

A natural generalization of our results for DNF would be to prove upper bounds on the degree of polynomial threshold functions which compute circuits of size s and depth d for fixed values of $d \geqslant 3$. Obtaining such upper bounds, however, seems to be quite difficult. We note that the parity function on n variables is known to require polynomial threshold functions of degree n [88, 5], and hence any proof of even an $n-1$ upper bound on the degree of polynomial threshold functions for constant depth polynomial size circuits would immediately yield a new proof of the well-known fact that parity is not computable by constant depth polynomial size circuits [49, 54].

As a first step towards proving upper bounds on polynomial threshold function degree for constant depth circuits, we show how the techniques of this paper can be used to obtain a nontrivial upper bound on the degree of polynomial threshold functions for read-once constant depth formulae.

Theorem 3.1.4 *For $d \geqslant 2$, any read-once Boolean formula of depth d over $\{\wedge, \vee, \neg\}$ can be computed by a polynomial threshold function of degree $O(n^{1-\frac{1}{3\cdot 2^{d-3}}} \log^{\frac{1}{3\cdot 2^{d-3}}} n)$.*

Proof: The proof is by induction on d. The base case $d = 2$ is supplied by Theorem 3.1.3. We suppose that the theorem holds for $d = 2, \ldots, k-1$ and prove it for $d = k$.

Let f be a depth-k read-once formula. We say that a *term* is a gate at the bottom level of f together with the literals that feed into it. Since f is read-once there can be at most n/t terms of size greater than t. We apply the decomposition procedure described in the proof of Lemma 3.3.2 to transform f into a 1-decision tree whose leaves each contain a depth-k read-once formula in which each term is of size at most t. As in Lemma 3.5.1 this decision tree is of rank at most n/t.

In each leaf of this tree, we replace each term with a new "dummy" variable that appears only once. We thus obtain a decision tree of rank n/t whose leaves each contain a read-once formula of depth $k-1$ over these dummy variables. By the induction hypothesis, each such formula is equivalent to a polynomial threshold function of degree $O(n^{1-\frac{1}{3\cdot 2^{k-4}}} \log^{\frac{1}{3\cdot 2^{k-4}}} n)$ which is defined over the dummy variables described above.

In each such polynomial threshold function, we now replace each dummy variable with a real-valued polynomial over the original variables which interpolates precisely the Boolean function computed by the original term. Since each term was of size at most t, each such polynomial is of degree at most t. Consequently the function computed at each leaf of the decision tree is a polynomial threshold function of degree $O(tn^{1-\frac{1}{3\cdot 2^{k-4}}} \log^{\frac{1}{3\cdot 2^{k-4}}} n)$.

As in the proof of Theorem 3.1.3, our original function f can now be expressed as a polynomial threshold function of degree

$$\frac{n}{t} + O(tn^{1-\frac{1}{3\cdot 2^{k-4}}} \log^{\frac{1}{3\cdot 2^{k-4}}} n).$$

Taking $t = n^{\frac{1}{3\cdot 2^{k-3}}} / \log^{\frac{1}{3\cdot 2^{k-3}}} n$ proves the theorem. ■

Corollary 3.6.1 *The class of read-once constant depth formulae can be learned in subexponential time.*

Chapter 4

Boosting and Hard-Core Set Construction

4.1 Introduction

4.1.1 Boosting and Hard-Core Sets

This chapter connects two fundamental ideas from theoretical computer science: *hard-core set construction,* a type of hardness amplification from computational complexity, and *boosting*, a technique from computational learning theory.

We refer to a *hardness amplification* as a result of the following form: given a Boolean function f that is mildly inapproximable by circuits of some bounded size g, construct from f a new function f' that is highly inapproximable by all circuits of size closely related to g. Here "mildly inapproximable" means roughly that no circuit can agree with f on a fraction of inputs very close to 1, while "highly inapproximable" means that no circuit can agree with f on a fraction of inputs significantly greater than 1/2. Hardness amplification results are a crucial component of recent attempts to derandomize the complexity class BPP [7, 59, 92]. Perhaps the most famous hardness amplification result is Yao's XOR-lemma [52], which states that if a Boolean function f is mildly inapproximable by circuits of size g then the XOR of several independent copies of f is highly inapproximable for circuits of size closely related to g.

Reference:	Set size parameter:	Circuit size parameter:
Impagliazzo [58]	$\Omega(\epsilon)$	$O(\gamma^2\epsilon^2)\cdot g$
Nisan [58]	$\Omega(\epsilon)$	$O(\gamma^2(\log(1/\gamma\epsilon))^{-1})\cdot g$
This chapter	$\Omega(\epsilon)$	$O(\gamma^2(\log(1/\epsilon))^{-1})\cdot g$

Table 4.1: Comparison of known hard-core set constructions.

While the goal of hardness amplification is to amplify some small initial "hardness" of a boolean function, the goal of boosting is to "boost" some small initial advantage over random guessing that a learner can achieve in Valiant's PAC (Probabilistically Approximately Correct) model of learning. Roughly speaking, a *strong* learning algorithm in this model is an algorithm which, given access to random labelled examples $\langle x, f(x)\rangle$ drawn from any distribution $\mathcal{D}$, can generate a hypothesis h such that $\Pr_{x\in\mathcal{D}}[f(x) = h(x)] \geqslant 1-\epsilon$ for any $\epsilon > 0$, while a *weak* learning algorithm [67] can only do this for some $1/2 > \epsilon > 0$. Schapire [100] and then Freund [43, 44] gave boosting algorithms which convert weak learners into strong learners, thus proving the equivalence of weak and strong learnability. Since then, boosting has been applied in a wide variety of contexts and continues to be an active area of research, see e.g. [37, 38, 40, 39, 46, 63, 101]. All known boosting algorithms work by using the weak learning algorithm several times on a sequence of carefully constructed distributions.

Superficially, boosting and hardness amplification seem to have opposite goals—boosting constructs a hypothesis which closely approximates a function f while hardness amplification results prove that certain functions are hard to approximate. The proof techniques employed in both areas, however, have a similar structure. All known hardness amplification results go by contradiction: assuming there exists a circuit C capable of mildly approximating f', one proves the existence of a slightly larger circuit which closely approximates f. From this perspective, a hardness amplification proof resembles a type of boosting procedure: circuits which mildly approximate a function f' (these correspond to the hypotheses output by the weak learner) are combined to form a new circuit computing f on a large fraction of inputs.

In an important paper, Impagliazzo [58] reduces the problem of amplifying the hardness of a function f to the problem of constructing a distribution $\mathcal{D}$ such that f is highly inapproximable by small circuits for inputs chosen according to $\mathcal{D}$. He then constructs such a distribution and uses it to prove an XOR lemma. Impagliazzo also shows that the existence of such a distribution implies the existence of a "hard-core set" as defined in Section 4.2; we thus refer to Impagliazzo's method of constructing such a distribution as a *hard-core set construction*. Schapire [100] was the first to point out that the existence of a boosting algorithm implies the existence of such a distribution.

4.1.2 Our Results

We give an explicit correspondence between the distributions that arise in Impagliazzo's hard-core set construction and the distributions constructed by boosting algorithms. This observation allows us to prove that the hard-core set construction of Impagliazzo *is* a boosting algorithm when the initial distribution is uniform. As we will show, there are two important parameters which boosting and hard-core set constructions share: the number of "stages" required and the "boundedness" of the distributions which are constructed. Interestingly, the procedures which have been used for hard-core set construction have better "boundedness" and can be used to improve algorithms in computational learning theory, while boosting algorithms require fewer "stages" and can be used to improve hard-core set construction.

We first show how to use known boosting algorithms to obtain new hard-core set constructions. In [58] Impagliazzo proves the following theorem: given a function f such that no circuit of size less than g correctly computes f on more than $(1-\epsilon)2^n$ inputs, then for any $\gamma < 1/2$ there exists a set S of size $\epsilon 2^n$ such that no circuit of size $O(\gamma^2\epsilon^2)g$ can correctly compute f on more than a $(1/2+\gamma)$ fraction of the inputs in S. By letting known boosting algorithms dictate the construction of the distributions in Impagliazzo's proof, we improve on previous results with respect to the circuit size parameter with only a small constant factor loss in the set size parameter. As explained in Section 4.4.5, we believe our parameters to be optimal up to constant factors with respect to this class of techniques. Table 1 summarizes our hard-core set construction results.

We then show how to use Impagliazzo's hard-core set construction to obtain a more efficient version of Jackson's Harmonic Sieve algorithm [62] for learning DNF formulae under the uniform distribution using membership queries. Jackson's original algorithm learns using the hypothesis class of threshold-of-parity functions and runs in time essentially $\tilde{O}(ns^8/\epsilon^{12})$, where n is the number of variables in the DNF formula, s is the number of terms, and ϵ is the accuracy parameter. Our variant uses the same hypothesis class and runs in time $\tilde{O}(ns^8/\epsilon^8)$. We can further improve the running time to $\tilde{O}(ns^8/\epsilon^6)$ at the cost of learning using a more complex class of hypotheses.

In recent work Bshouty, Jackson and Tamon [30] have improved the running time of the Harmonic Sieve to $\tilde{O}(ns^4/\epsilon^4)$, where r is the number of distinct variables which appear in the minimal DNF representation of the target formula. Our results improve the running time of their new algorithm to $\tilde{O}(ns^4/\epsilon^2)$ time steps, which is the fastest known algorithm for PAC learning DNF with membership queries under the uniform distribution.

Our main technical contribution is a careful analysis of the distributions constructed during the boosting process. We show that boosting procedures which construct distributions with high minimum entropy are desirable for good hard-core set constructions.

4.1.3 Related Work

Boneh and Lipton [26] have applied Yao's XOR-lemma to prove the equivalence of weak and strong learnability for certain types of concept classes under the uniform distribution. Their result applies to concept classes closed under a polynomial number of XOR operations.

4.1.4 Organization

In Section 2 we give an overview of the hard-core set construction found in [58]. In Section 3 we outline the structure of all known boosting algorithms. In Section 4 we give an explicit connection between the constructions detailed in Sections 2 and 3 and show how to apply boosting techniques to obtain new hard-core set constructions. In Section 5 we show how the techniques described in section 2 can be used to improve the running time of Jackson's algorithm for learning DNF formulae. We also describe related algorithms in learning theory where our techniques can be applied.

4.2 Hard-Core Set Construction Overview

4.2.1 Definitions

Our first definition, taken from [58], formalizes the notion of a function which is hard to approximate. (Readers who are familiar with the notation of [58] will notice that we are using different variables; the reasons for this will become clear in Section 4.4.)

Definition 4.2.1 *Let f be a boolean function on $\{0,1\}^n$ and $\mathcal{D}$ a distribution on $\{0,1\}^n$. Let $0 < \epsilon < 1/2$ and let $n \leqslant g \leqslant 2^n/n$. We say that f is* ϵ-hard for size g under *$\mathcal{D}$ if for any boolean circuit C with at most g gates, we have* $\Pr_{\mathcal{D}}[f(x) = C(x)] \leqslant 1-\epsilon$.

In other words, any circuit of size at most g must disagree with f with probability at least ϵ for x drawn according to $\mathcal{D}$. Throughout the chapter we use $\mathcal{U}$ to denote the uniform distribution on $\{0,1\}^n$.

Definition 4.2.2 *A* measure *on $\{0,1\}^n$ is a function $M : \{0,1\}^n \to [0,1]$. The* absolute size *of a measure M is denoted by $|M|$ and equals $\sum_x M(x)$; the* relative size *of M is denoted $\mu(M)$ and equals $|M|/2^n$.*

Definition 4.2.3 *For any real valued function ξ, $L_\infty(\xi)$ denotes* $\max_x |\xi(x)|$.

The quantity $\log(L_\infty(\mathcal{D})^{-1})$ is often referred to as the *minimum entropy* of $\mathcal{D}$. There is a natural correspondence between measures and distributions: the distribution $\mathcal{D}_M$ induced by a measure M is defined by $\mathcal{D}_M(x) = M(x)/|M|$. Conversely, if $\mathcal{D}$ is a distribution then the measure $M_{\mathcal{D}}$ induced by $\mathcal{D}$ is defined by $M_{\mathcal{D}}(x) = \mathcal{D}(x)/L_\infty(\mathcal{D})$. Thus $M_{\mathcal{D}}$ is the largest measure which is a constant-multiple

Input: real numbers $0 < \epsilon < 1$, $0 < \gamma < 1/2$
Boolean function f

Output: circuit h such that $\Pr_{\mathcal{U}}[h(x) = f(x)] > 1 - \epsilon$

set $i \leftarrow 0$
set $M_0(x) \equiv 1$
until $\mu(M_i) < \epsilon$ **do**
    let C_i be a circuit of size at most g' with $\Pr_{\mathcal{D}_{M_i}}[C(x) = f(x)] \geqslant 1/2 + \gamma$
    set $R_{C_i}(x) \equiv 1$ if $f(x) = C_i(x)$, $R_{C_i}(x) \equiv -1$ otherwise
    set $N_i(x) \equiv \sum_{0 \leqslant j \leqslant i} R_{C_j}(x)$
    set $M_{i+1}(x) \equiv \begin{cases} 1 & \text{if } N_i(x) < 0 \\ 1 - \epsilon\gamma N_i(x) & \text{if } 0 \leqslant N_i(x) \leqslant 1/\epsilon\gamma \\ 0 & \text{if } N_i(x) > 1/\epsilon\gamma \end{cases}$
    set $i \leftarrow i + 1$
return $h \equiv MAJ(C_0, C_1, \ldots, C_{i-1})$

Figure 4-1: The `IHA` algorithm.

rescaling of $\mathcal{D}$ (note that $\mathcal{D}$ itself is a measure, though typically one which has much smaller size than $M_{\mathcal{D}}$). It is clear that $|M_{\mathcal{D}}| = 1/L_\infty(\mathcal{D})$ and $\mu(M_{\mathcal{D}}) = 1/L_\infty(2^n\mathcal{D})$. Thus, large measures correspond to distributions which do not assign large weight to any point (i.e., have high minimum entropy).

The next definition is also from [58]:

Definition 4.2.4 *We say that f is* γ-hard-core on M for size g *if* $\Pr_{\mathcal{D}_M}[f(x) = C(x)] \leqslant 1/2 + \gamma$ *for every circuit C of size at most g. For $S \subseteq \{0,1\}^n$ we say that* f is γ-hard-core on S for size g *if f is γ-hard-core on M_S for size g, where $M_S(x)$ is the characteristic function of S.*

4.2.2 Existence of Hard-Core Measures

The following theorem due to Impagliazzo is the starting point of all our results:

Theorem 4.2.5 [58] *Let f be ϵ-hard for circuits of size g under $\mathcal{U}$ and let $0 < \gamma < 1/2$. Then there is a measure M with $\mu(M) \geqslant \epsilon$ such that f is γ-hard-core on M for* *size* $g' = O(\epsilon^2\gamma^2)g$.

Proof Sketch: Assume by way of contradiction that for every measure M with $\mu(M) \geqslant \epsilon$ there is a circuit C_M of size at most g' such that $\Pr_{\mathcal{D}_M}[f(x) = C_M(x)] > 1/2+\gamma$. Now consider the algorithm `IHA` which is given in Figure 4.2.1. This algorithm iteratively modifies M until its relative size is less than ϵ. After each modification we obtain a circuit C_M as above. Once the relative size of M becomes less than ϵ we combine the circuits obtained during the process to contradict the original assumption. The following easily verifiable claims are useful for understanding how `IHA` works:

- $N_i(x)$ is the margin by which the majority vote of $C_0, \ldots, C_i$ correctly predicts the value of $f(x)$.

- The measure M_{i+1} assigns weight 0 to points where the margin of correctness is large, weight 1 to points where the margin is negative, and intermediate weight to points where the margin is positive but small.

Impagliazzo proves that after at most $i_0 = O(1/(\epsilon^2\gamma^2))$ cycles through the loop, $\mu(M_i)$ must be less than ϵ. Once this happens and we exit the loop, it is easy to see that $h \equiv MAJ(C_0, \ldots, C_{i-1})$ agrees with f on all inputs except those which have $N_i(x) \leqslant 0$ and hence $M_i(x) = 1$. Since $\mu(M_i) < \epsilon$, this implies that $\Pr_{\mathcal{U}}[f(x) = h(x)] \geqslant 1 - \mu(M_i) > 1 - \epsilon$. But h is a majority circuit over at most i_0 circuits each of size at most g', and majority over i_0 inputs can be computed by a circuit of size $O(i_0)$. It follows that h has at most $g'i_0 + O(i_0) \leqslant g$ gates, which contradicts the original assumption that f is ϵ-hard for circuits of size g under $\mathcal{U}$. ■

Using a non-constructive proof technique, Nisan has established a similar result which is reported in [58]. In Nisan's theorem the circuit size parameter is slightly worse as a function of γ but substantially better as a function of ϵ :

Theorem 4.2.6 **[58]** *Let f be ϵ-hard for circuits of size g under $\mathcal{U}$ and let $0 < \gamma < 1/2$. Then there is a measure M with $\mu(M) \geqslant \epsilon$ such that f is γ-hard-core on M for size* $g' = O(\gamma^2(\log(2/\gamma\epsilon))^{-1})g$.

In Section 4.4.2 we will establish results of this type which have a better circuit size parameter than either Theorem 4.2.5 or Theorem 4.2.6.

We note that Theorems 4.2.5 and 4.2.6 assert the existence of a large measure, not a large set as was promised in Section 1. Using a probabilistic argument which we give in Section 4.4.4 Impagliazzo has shown that the existence of a large measure M on which f is hard-core implies the existence of a large set S on which f is also hard-core (with slightly different parameters).

4.3 Boosting Overview

In this section we define the learning model, weak and strong learning, and *boosting*, which converts a weak learner to a strong one.

4.3.1 Definitions

We take as our learning framework Valiant's widely studied PAC (Probably Approximately Correct) model of concept learning [111]. In this model a *concept class* is a collection $C = \cup_{n \geqslant 1} C_n$ of boolean functions where each $f \in C_n$ is a boolean function on $\{0,1\}^n$. For example, we might have C_n as the class of all boolean conjunctions on n variables. If f and h are two boolean functions on $\{0,1\}^n$ and $\mathcal{D}$ is a distribution on $\{0,1\}^n$, we say that h is an *ϵ-approximator for f under $\mathcal{D}$* if $\Pr_{\mathcal{D}}[f(x) = h(x)] \geqslant 1 - \epsilon$. The learner has access to an *example oracle* $\mathrm{EX}(f, \mathcal{D})$ which, when queried, provides a labelled example $\langle x, f(x) \rangle$ where x is drawn from $\{0,1\}^n$ according to the distribution $\mathcal{D}$ and $f \in C_n$ is the unknown target concept which the algorithm is trying to learn. The goal of the learner is to generate an ϵ-approximator for f under $\mathcal{D}$. We thus have the following definition:

Definition 4.3.1 *An algorithm* `A` *is a* strong PAC learning algorithm for a concept class C *if the following condition holds: for any $n \geqslant 1$, any $f \in C_n$, any distribution $\mathcal{D}$ on $\{0,1\}^n$, and any $0 < \epsilon, \delta < 1$, if* `A` *is given access to n, ϵ, δ and $EX(f, \mathcal{D})$, then* `A` *runs in time polynomial in n, ϵ^{-1}, δ^{-1}, and $size(f)$, and with probability at least*

$1 - \delta$ *algorithm* `A` *outputs an* ϵ*-approximator for* f *under* $\mathcal{D}$.

In the above definition $size(f)$ measures the complexity of the function f under some fixed reasonable encoding scheme. For the concept class DNF which we will consider in Section 4.5, $size(f)$ is the minimum number of terms in any disjunctive normal form representation of f.

If the algorithm `A` is only guaranteed to find a $(1/2 - \gamma)$-approximator for some $\gamma > 0$, then we say that `A` is a $(1/2 - \gamma)$-*approximate* learning algorithm; if $\gamma = \Omega(1/p(n, size(f)))$ for some polynomial p, we say that `A` is a *weak* learning algorithm (The notion of weak learning was introduced by Kearns and Valiant in [67]). We will abuse notation and say that `A` is a $(1/2 - \gamma)$-approximate learning algorithm for f if `A` is a $(1/2 - \gamma)$-approximate learning algorithm for the concept class C which consists of the single function f. In a series of important results, Schapire [100] and subsequently Freund [43, 44] have shown that if `A` is a weak learning algorithm for a concept class C, then there exists a strong learning algorithm for C. Their proofs are highly constructive in that they give explicit *boosting* algorithms which transform weak learning algorithms into strong ones. We now formally define boosting algorithms (a related definition can be found in [45]):

Definition 4.3.2 *An algorithm* `B` *is said to be a* boosting algorithm *if it satisfies the following condition: for any boolean function* f *and any distribution* $\mathcal{D}$, *if* `B` *is given* $0 < \epsilon, \delta < 1$, $0 < \gamma \leqslant 1/2$, *an example oracle* $EX(f, \mathcal{D})$, *and a* $(1/2 - \gamma)$*-approximate learning algorithm* `WL` *for* f, *then algorithm* `B` *runs in time polynomial in* $n, size(f), \gamma^{-1}, \epsilon^{-1}$, *and* δ^{-1}, *and with probability at least* $1 - \delta$ *algorithm* `B` *outputs an* ϵ*-approximator for* f *under* $\mathcal{D}$.

4.3.2 Structure of Boosting Algorithms

All known boosting algorithms rely crucially on the fact that the weak learning algorithm `WL` can find a $(1/2 - \gamma)$-approximator for f under $\mathcal{D}'$ for *any* distribution $\mathcal{D}'$, as long as `WL` is given access to the example oracle $\mathrm{EX}(f, \mathcal{D}')$. We give the following high-level definition:

Definition 4.3.3 *A* canonical booster *is a boosting algorithm which has the following iterative structure:*

- *At stage 0 the algorithm starts with* $\mathcal{D}_0 = \mathcal{D}$ *and uses* `WL` *to generate a* $(1/2-\gamma)$*-approximator* h_0 *for* f *under* $\mathcal{D}_0$*.*
- *At stage* i *the boosting algorithm does two things: (1) constructs a distribution* $\mathcal{D}_i$ *which favors points where the previous hypotheses* $h_0, \ldots, h_{i-1}$ *do poorly at predicting the value of* f*, and (2) simulates the example oracle* $EX(f, \mathcal{D}_i)$ *and lets* `WL` *access this simulated example oracle to produce a hypothesis* h_i *which is a* $(1/2-\gamma)$*-approximator for* f *under* $\mathcal{D}_i$*.*
- *Finally, after doing this repeatedly for several stages, the boosting algorithm combines the hypotheses* $h_0, \ldots, h_{i-1}$ *in some way to obtain a final hypothesis* h *which is an* ϵ*-approximator for* f *under* $\mathcal{D}$*.*

We feel that this definition captures the essence of known boosting algorithms.

4.4 Hard-Core Set Construction from Boosting

4.4.1 A Structural Similarity

From the descriptions of the hard-core set construction of Section 4.2 and the canonical boosting algorithm of Section 4.3, one can see a close structural resemblance between the `IHA` algorithm and the canonical boosting algorithm outlined above. To be more specific, just as `IHA` assumes that at each stage there is a circuit C_i for which $\Pr_{\mathcal{D}_{M_i}}[f(x) \neq C_i(x)] \leqslant 1/2 - \gamma$, the canonical boosting algorithm assumes that `WL` can generate at each stage a hypothesis h_i for which $\Pr_{\mathcal{D}_i}[f(x) \neq h_i(x)] \leqslant 1/2 - \gamma$. The induced distributions $\mathcal{D}_{M_i}$ of `IHA` correspond precisely to the distributions $\mathcal{D}_i$ of the canonical boosting algorithm (note that `IHA` starts off with the measure $M_0 = 1$ which corresponds to the uniform distribution $\mathcal{U} = \mathcal{D}_0$). Finally, just as the canonical boosting algorithm combines the hypotheses $h_0, \ldots, h_{i-1}$ in some fashion to obtain a final hypothesis h which has $\Pr_{\mathcal{U}}[f(x) = h(x)] \geqslant 1 - \epsilon$, the `IHA` algorithm combines the circuits $C_0, \ldots, C_{i-1}$ by taking majority to obtain a circuit h such that $\Pr_{\mathcal{U}}[f(x) = h(x)] \geqslant 1 - \epsilon$.

We conclude that `IHA` is an algorithm which succeeds in boosting *provided that the starting distribution is the uniform distribution* $\mathcal{U}$. Since boosting algorithms from computational learning theory will work for *any* starting distribution, *a priori* it seems as if it should be possible to use any boosting algorithm in place of `IHA` and obtain a hard-core set construction. In the next section we prove a theorem which formalizes this idea and emphasizes the parameters which are important to obtain a good hard-core set construction.

4.4.2 A General Hard-Core Set Construction

Definition 4.4.1 *Let $\mathcal{D}$ be a distribution over $\{0,1\}^n$. For $d \geqslant 1$ we say that $\mathcal{D}$ is* d-smooth *if $L_\infty(2^n\mathcal{D}) \leqslant d$.*

As an immediate consequence of Definitions 4.2.2 and 4.4.1, we have

Observation 4.4.2 *If distribution $\mathcal{D}$ is d-smooth then $\mu(M_{\mathcal{D}}) \geqslant 1/d$.*

Definition 4.4.3 *Let* `B` *be a canonical boosting algorithm which takes as input ϵ, δ, γ, an example oracle $EX(f, \mathcal{D})$, and a $(1/2 - \gamma)$-approximate learning algorithm* `WL` *for f.*

1. *We say that* `B` *is a $k(\epsilon, \gamma)$-*stage *boosting algorithm if the following holds: For all example oracles $EX(f, \mathcal{D})$ and $(1/2-\gamma)$-approximate learners* `WL` *for f, algorithm* `B` *simulates at most $k = k(\epsilon, \gamma)$ distributions $\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_{k-1}$ for* `WL` *and uses* `WL` *to generate at most k hypotheses $h_0, \ldots, h_{k-1}$.*

2. *We say that* `B` *is a $d(\epsilon, \gamma)$-*smooth *boosting algorithm if the following holds: For all functions f and $(1/2 - \gamma)$-approximate learners* `WL`*, when* `B` *is given $EX(f, \mathcal{U})$ and* `WL`*, with nonzero probability both of the following events occur: (i) the simulated distributions $\mathcal{D}_0, \ldots, \mathcal{D}_{k-1}$ are each $d(\epsilon, \gamma)$-smooth, and (ii) the hypothesis h which* `B` *outputs satisfies $\Pr_{\mathcal{U}}[f(x) = h(x)] \geqslant 1 - \epsilon$.*

The property of the distributions $\mathcal{D}_i$ described in part 2 of the above definition is similar to Levin's notion of "dominated" distributions [80].

Now we can state the following theorem which generalizes Impagliazzo's hard-core set construction.

Theorem 4.4.4 *Let* `B` *be a* $k(\epsilon, \gamma)$*-stage,* $d(\epsilon, \gamma)$*-smooth boosting algorithm which outputs as its final hypothesis a circuit of size* r *over inputs* $h_0, \ldots, h_{k-1}$*. Let* f *be* ϵ*-hard for circuits of size* g *under* $\mathcal{U}$ *and let* $0 < \gamma < 1/2$*. Then there is a measure* M *on* $\{0,1\}^n$ *with* $\mu(M) \geqslant 1/d(\epsilon, \gamma)$ *such that* f *is* γ*-hard-core on* M *for size* $g' = (g - r)/k(\epsilon, \gamma)$*.*

Proof: The proof is analogous to the proof of Theorem 4.2.5. Assume by way of contradiction that for every measure M with $\mu(M) \geqslant 1/d(\epsilon, \gamma)$ there is a circuit C_M of size at most g' such that $\Pr_{\mathcal{D}_M}[f(x) = C_M(x)] \geqslant 1/2 + \gamma$. By Observation 4.4.2, this implies that for every $d(\epsilon, \gamma)$-smooth distribution $\mathcal{D}$ there is a circuit $C_{\mathcal{D}}$ of size at most g' such that $\Pr_{\mathcal{D}}[f(x) = C_{\mathcal{D}}(x)] \geqslant 1/2 + \gamma$.

Now run the boosting algorithm `B` on inputs ϵ, δ, γ, and $\mathrm{EX}(f, \mathcal{U})$. Since `B` is $d(\epsilon, \gamma)$-smooth, with nonzero probability we have that (i) every distribution $\mathcal{D}_i$ which `B` simulates will be $d(\epsilon, \gamma)$-smooth, and (ii) the final hypothesis which `B` outputs is an ϵ-approximator to f under the original distribution $\mathcal{U}$. By (i), there must exist a circuit C_i of at most g' gates which is a $(1/2-\gamma)$-approximator for f under $\mathcal{D}_i$. Give `B` this circuit when it calls `WL` on distribution $\mathcal{D}_i$. Now by (ii), the final hypothesis which `B` outputs must be an ϵ-approximator to f under the original distribution $\mathcal{U}$. But since `B` is $k(\epsilon, \gamma)$-stage, this final hypothesis is a circuit of size at most $r + g'k(\epsilon, \gamma) \leqslant g$, which contradicts the original assumption that f is ϵ-hard for circuits of size g under $\mathcal{U}$. ■

4.4.3 New Hard-Core Set Constructions

Here we apply Theorem 4.4.4 to obtain new hard-core set constructions from known boosting algorithms. We proceed in stages. First, we show how two different boosting algorithms yield different hard-core set constructions. Next, we combine these boosting algorithms to achieve a new hard-core set construction which improves on results of Impagliazzo and Nisan in the circuit size parameter but has a set size parameter worse than their results by a logarithmic factor. In Section 4.4.5 we improve the set size parameter to within a constant factor of the Impagliazzo/Nisan results.

We first consider Freund's boost-by-majority algorithm from [43] which, following [61], we refer to as `F1`. Algorithm `F1` is a $k = O(\gamma^{-2}\log(1/\epsilon))$-stage boosting algorithm which combines its k hypotheses using the majority function. Jackson's analysis ([61], pp. 57–59) yields the following fact about `F1`:

Fact 4.4.5 *If* `F1` *is given inputs* ϵ, δ, γ, $EX(f, \mathcal{D})$ *and a* $(1/2 - \gamma)$*-approximate weak learner* `WL` *for* f*, then with high probability each distribution* $\mathcal{D}'$ *which* `F1` *simulates for* `WL` *satisfies*

$$L_\infty(\mathcal{D}') = O(1/\epsilon^3) \cdot L_\infty(\mathcal{D}).$$

This immediately implies that `F1` is $O(1/\epsilon^3)$-smooth.[1] We thus obtain the following hard-core set construction:

Theorem 4.4.6 *Let* f *be* ϵ*-hard for circuits of size* g *under* $\mathcal{U}$ *and let* $0 < \gamma < 1/2$*. Then there is a measure* M *on* $\{0,1\}^n$ *with* $\mu(M) = \Omega(\epsilon^3)$ *such that* f *is* γ*-hard-core on* M *for size* $g' = O(\gamma^2(\log(1/\epsilon))^{-1}g)$*.*

Next, we consider Freund's later $\mathtt{B}_{\mathrm{Filt}}$ algorithm from [45] (the name comes from the fact that the algorithm "filters" examples from the original distribution to simulate new distributions). Like `F1`, algorithm $\mathtt{B}_{\mathrm{Filt}}$ is a k-stage boosting algorithm for $k = O(\gamma^{-2}(\log 1/\epsilon))$. $\mathtt{B}_{\mathrm{Filt}}$ combines its $(1/2 - \gamma)$-approximators to obtain an ϵ-approximator for f by using a majority function on k inputs which may have some random inputs. A straightforward argument shows that some circuit of size $O(k)$ is a ϵ-approximator for f. To analyze the smoothness of $\mathtt{B}_{\mathrm{Filt}}$, we use the following fact which follows from Lemma 3.4 and Lemma 3.9 of [45]:

[1] In [43] Freund states that the `F1` algorithm can be shown to be $O(1/\epsilon^2)$-smooth. Jackson proves that `F1` is $O(1/\epsilon^3)$-smooth and states that variants of `F1` can be shown to be $O(1/\epsilon^{2+\rho})$-smooth for arbitrarily small values $\rho > 0$.

Fact 4.4.7 *If* $\mathtt{B}_{Filt}$ *is given inputs* ϵ, δ, γ, $EX(f, \mathcal{D})$ *and a* $(1/2-\gamma)$*-approximate weak learner* `WL` *for* f, *then with high probability each distribution* $\mathcal{D}'$ *which* $\mathtt{B}_{Filt}$ *simulates for* `WL` *satisfies*

$$L_\infty(\mathcal{D}') = O(\log(1/\epsilon)/(\epsilon\gamma)) \cdot L_\infty(\mathcal{D}).$$

Since Fact 4.4.7 implies that $\mathtt{B}_{\text{Filt}}$ is $O(\log(1/\epsilon)/(\epsilon\gamma))$-smooth, we obtain

Theorem 4.4.8 *Let* f *be* ϵ*-hard for circuits of size* g *under* $\mathcal{U}$ *and let* $0 < \gamma < 1/2$. *Then there is a measure* M *on* $\{0,1\}^n$ *with* $\mu(M) = \Omega(\epsilon\gamma(\log(1/\epsilon))^{-1})$ *such that* f *is* γ*-hard-core on* M *for size* $g' = O(\gamma^2(\log(1/\epsilon))^{-1}g)$.

Finally we establish a stronger hard-core set construction by combining the previous two approaches. In [44] Freund describes a two-level boosting algorithm which works as follows: algorithm `F1` is used to boost from accuracy $(1/2-\gamma)$ to accuracy $1/4$, and algorithm $\mathtt{B}_{\text{Filt}}$ boosts from accuracy $1/4$ to accuracy ϵ by taking `F1` as its weak learner. We call this combined algorithm $\mathtt{B}_{\text{Comb}}$.

Lemma 4.4.9 *The boosting algorithm* $\mathtt{B}_{Comb}$ *is an* $O(\gamma^{-2}\log(1/\epsilon))$*-stage boosting algorithm.*

Proof: The top level of $\mathtt{B}_{\text{Comb}}$, which uses algorithm $\mathtt{B}_{\text{Filt}}$, takes $O(\log(1/\epsilon))$ stages since the weak learner which it uses is `F1` which provides $(1/2-\gamma')$-accurate hypotheses with $\gamma' = 1/4$. The bottom level, which uses algorithm `F1`, takes $O(\gamma^{-2})$ stages since it boosts a $(1/2-\gamma)$-approximate learner to accuracy $1/4$. Consequently the combined algorithm $\mathtt{B}_{\text{Comb}}$ uses the claimed number of stages. ∎

Lemma 4.4.10 $\mathtt{B}_{Comb}$ *is an* $O(\log(1/\epsilon)/\epsilon)$*-smooth boosting algorithm.*

Proof: Since $\mathtt{B}_{\text{Filt}}$ is boosting from accuracy $1/4$ to accuracy ϵ using `F1` as its weak learner, Fact 4.4.7 implies that each distribution $\mathcal{D}'$ which $\mathtt{B}_{\text{Filt}}$ passes to `F1` satisfies

$$L_\infty(\mathcal{D}') = O(\log(1/\epsilon)/\epsilon) \cdot L_\infty(\mathcal{D}).$$

Since `F1` is boosting from accuracy $(1/2-\gamma)$ to accuracy $1/4$, Fact 4.4.5 implies that if $\mathcal{D}''$ is the distribution which `F1` passes to `WL`, then

$$L_\infty(\mathcal{D}'') = O(1) \cdot L_\infty(\mathcal{D}').$$

Combining these two equations, we find that

$$L_\infty(\mathcal{D}'') = O(\log(1/\epsilon)/\epsilon) \cdot L_\infty(\mathcal{D}).$$

■

Finally, we note that the final hypothesis which $\mathtt{B}_{\text{Comb}}$ outputs is a depth 2 majority circuit over the weak hypotheses h_i, since both `F1` and $\mathtt{B}_{\text{Filt}}$ combine their hypotheses using the majority function. A straighforward bound on the size of this majority circuit yields the following hard-core set construction:

Theorem 4.4.11 *Let f be ϵ-hard for circuits of size g under $\mathcal{U}$ and let $0 < \gamma < 1/2$. Then there is a measure M on $\{0,1\}^n$ with $\mu(M) = \Omega(\epsilon(\log(1/\epsilon))^{-1})$ such that f is γ-hard-core on M for size $g' = O(\gamma^2(\log(1/\epsilon))^{-1}g)$.*

4.4.4 From Hard-Core Measures to Hard-Core Sets

While we have been referring to our results thus far as hard-core set constructions, in fact Theorems 4.4.6, 4.4.8 and 4.4.11 establish the existence of hard-core measures rather than hard-core sets. We now show how hard-core measure constructions such as Theorems 4.4.6, 4.4.8 and 4.4.11 imply corresponding hard-core set constructions. This conversion from hard-core measures to hard-core sets is based on an argument from [58].

We will use the following crude lemma:

Lemma 4.4.12 *The number of Boolean circuits of size g is at most $((n+5)g^2)^g$.*

Proof: To specify a circuit it suffices to specify, for each of g gates, the two inputs and the label of the gate. For a given gate there are at most g^2 choices for the inputs and at most $n+5$ choices $(x_1, \ldots, x_n, \neg, \vee, \wedge, TRUE, FALSE)$ for the label. $\blacksquare$

The following easy fact follows from the definition of $|M|$.

Fact 4.4.13 *Let C be a circuit and view C and f as taking values in $\{-1, 1\}$. Then $\Pr_{\mathcal{D}_M}[C(x) = f(x)] = \frac{1}{2} + \rho$ if and only if $\sum_{x \in \{0,1\}^n} M(x)C(x)f(x) = 2\rho|M|$.*

Now we state and prove our conversion from hard-core measures to hard-core sets. Constant factors have not been optimized in the following lemma.

Lemma 4.4.14 *Let f be a Boolean function on $\{0,1\}^n$ and M a measure such that (i) $\mu(M) \geqslant \tau$ and (ii) f is γ-hard-core on M for size g where $g \leqslant \frac{1}{2n} \cdot 2^n \gamma^2 \tau^2$. Then there exists a set S with $|S| \geqslant \frac{\tau}{2} 2^n$ such that f is 4γ-hard-core on S for size g.*

Proof: Let C be any circuit of size at most g. Consider the following randomized construction of a set $S \subseteq \{0,1\}^n$: for each $x \in \{0,1\}^n$ put x in S with probability $M(x)$. Let M_S be the characteristic function of S. For each $x \in \{0,1\}^n$ the expected value of $M_S(x)$ is $M(x)$, and thus by linearity of expectation we have

$$E\left[\sum_{x \in \{0,1\}^n} M_S(x)C(x)f(x)\right] = \sum_{x \in \{0,1\}^n} M(x)C(x)f(x) \leqslant 2\gamma|M|$$

where the inequality follows from Fact 4.4.13 and f being γ-hard-core on M for size g. For each value of x the quantity $M_S(x)C(x)f(x)$ is a random variable in the interval $[-1, 1]$. Hoeffding's tail bound now implies that

$$\begin{aligned} \Pr\left[\frac{1}{2^n} \sum_{x \in \{0,1\}^n} M_S(x)C(x)f(x) \geqslant 4\gamma\mu(M)\right] &\leqslant \exp\left(\frac{-2 \cdot 2^n (2\gamma\mu(M))^2}{4}\right) \\ &\leqslant \exp(-2 \cdot 2^n \gamma^2 \tau^2) \end{aligned}$$

where the second inequality is because $\mu(M) \geqslant \tau$. By Lemma 4.4.12 and the bound on g we have that the number of circuits of size at most g is at most $((n+5)g^2)^g \ll \frac{1}{10} \exp(2 \cdot 2^n \gamma^2 \tau^2)$. Thus the probability that there exists a C with $|C| \leqslant g$ such that $\sum_{x \in \{0,1\}^n} M_S(x)C(x)f(x) \geqslant 4\gamma|M|$ is less than $\frac{1}{10}$.

Meanwhile, we also have that $E[|S|] = |M|$ and $|S|$ is a sum of 2^n independent random variables each with range $\{0, 1\}$. Thus Hoeffding's bound implies that

$$\begin{aligned} \Pr\left[\frac{|S|}{2^n} < \frac{\mu(M)}{2}\right] &\leqslant \exp(-2 \cdot 2^n(\mu(M)/2)^2) \\ &\leqslant \exp(-2^n\tau^2/2). \end{aligned}$$

If $\tau < \frac{1}{2^{n/2}}$ then the bound on g in the statement of the lemma is less than $\frac{1}{2n}$ and the lemma is trivially true. Thus we assume that $\tau \geqslant \frac{1}{2^{n/2}}$ which by the above inequality implies that $\Pr[|S| \geqslant \frac{|M|}{2}] > \frac{1}{10}$.

These two probability bounds together imply that there exists some S with $|S| \geqslant \frac{|M|}{2} \geqslant \frac{\tau}{2}2^n$ such that for every circuit C with at most g gates

$$\sum_{x \in \{0,1\}^n} M_S(x)C(x)f(x) \leqslant 4\gamma|M| \leqslant 8\gamma|S| = 8\gamma|M_S|.$$

By Fact 4.4.13 we have that $\Pr_{x\in S}[C(x) = f(x)] \leqslant \frac{1}{2} + 4\gamma$ for every such circuit C, and thus f is 4γ-hard-core on S. ∎

Combining this lemma with Theorem 4.4.11 we obtain the following hard-core set construction:

Theorem 4.4.15 *Let f be ϵ-hard for circuits of size g under $\mathcal{U}$ where $g = O\left(\frac{2^n}{n} \cdot \frac{\epsilon^2}{\log 1/\epsilon}\right)$ and let $0 < \gamma < 1/2$. Then there is a set $S \subseteq \{0, 1\}^n$ with $|S| = \Omega(\epsilon(\log 1/\epsilon)^{-1})2^n$ such that f is 4γ-hard-core on S for size $g' = O(\gamma^2(\log(1/\epsilon))^{-1}g)$.*

4.4.5 Optimal Hard-Core Set Construction

In this section we show how the set size parameter of Theorem 4.4.15 can be improved from $\Omega(\epsilon(\log 1/\epsilon)^{-1})$ to $\Omega(\epsilon)$. The basic idea is quite simple and was suggested by Avi Wigderson [116]. Let f be ϵ-hard for circuits of size g under $\mathcal{U}$ and let S be the hard-core set whose existence is asserted by Theorem 4.4.15. If $|S| = o(\epsilon 2^n)$ then we can take away the points in this hard-core set and intuitively f should still be almost ϵ-hard for circuits of size g under the uniform distribution on the remaining points.

We can then reapply Theorem 4.4.15 on the remaining points using "almost ϵ" in place of ϵ to obtain a new hard-core set. This procedure can be repeated until the total size of all the hard-core sets is $\Omega(\epsilon 2^n)$.

To make this argument precise we will need to consider measures which are defined over proper subsets of $\{0,1\}^n$. If $X \subseteq \{0,1\}^n$ and $M : X \to [0,1]$ is a measure we write $|M|_X$ to denote $\sum_{x \in X} M(x)$ and $\mu_X(M)$ to denote $\frac{|M|_X}{|X|}$. We write $\mathcal{U}_X$ to denote the uniform distribution on X.

The following generalization of Theorem 4.4.15 is easily seen to follow from the arguments of Section 4.4.3.

Theorem 4.4.16 *Fix $X \subseteq \{0,1\}^n$ such that $|X| \geqslant 2^n/2$. Let f be ϵ-hard for circuits of size g under $\mathcal{U}_X$ where $g = O\left(\frac{2^n}{n} \cdot \frac{\epsilon^2}{\log 1/\epsilon}\right)$ and let $0 < \gamma < 1/2$. Then there is a set $S \subset X$ with $|S| = \Omega(\epsilon(\log(1/\epsilon))^{-1})|X|$ such that f is 4γ-hard-core on S for size $g' = O(\gamma^2(\log(1/\epsilon))^{-1}g)$.*

Proof: Identical to the proof of Theorem 4.4.15 with X in place of $\{0,1\}^n$ and $|X|$ in place of 2^n. Note that there is enough slack in the Hoeffding tail bounds of Lemma 4.4.14 for them to still go through as long as $|X| \geqslant 2^n/2$. ■

We need the following easy lemma:

Lemma 4.4.17 *If $X \subseteq \{0,1\}^n$ satisfies $|X| \geqslant (1 - \frac{\epsilon}{2})2^n$ and f is ϵ-hard for circuits of size g under $\mathcal{U}$ then f is $\frac{\epsilon}{2}$-hard for circuits of size g under $\mathcal{U}_X$.*

Proof: For any circuit C with $|C| \leqslant g$ we have that $|\{x : C(x) = f(x)\}| \leqslant (1-\epsilon)2^n$. Thus $\Pr_{\mathcal{U}_X}[C(x) = f(x)] \leqslant \frac{1-\epsilon}{1-\epsilon/2} < 1 - \frac{\epsilon}{2}$. ■

Now we prove our strongest hard-core set construction.

Theorem 4.4.18 *Let f be ϵ-hard for circuits of size g under $\mathcal{U}$ where $g = O\left(\frac{2^n}{n} \cdot \frac{\epsilon^2}{\log 1/\epsilon}\right)$ and let $0 < \gamma < 1/2$. Then there is a set $S \subseteq \{0,1\}^n$ with $|S| \geqslant \frac{\epsilon}{2}2^n$ such that f is γ-hard-core on S for size $g' = O(\gamma^2(\log(1/\epsilon))^{-1}g)$.*

Proof: Theorem 4.4.15 implies that there is a set S_0 with $|S_0| = \Omega(\epsilon(\log 1/\epsilon)^{-1})2^n$ such that f is γ-hard-core on S_0 for size $g' = O(\gamma^2(\log 1/\epsilon)^{-1}g)$. If $|S_0| \geqslant \frac{\epsilon}{2}2^n$ we are done, so we assume that $|S_0| < \frac{\epsilon}{2}2^n$ and let $X_1 = \{0,1\}^n \setminus S_0$. Lemma 4.4.17 implies that f is $\frac{\epsilon}{2}$-hard for circuits of size g under $\mathcal{U}_{X_1}$. Theorem 4.4.16 now implies the existence of a set $S_1 \subset X_1$ with $|S_1| = \Omega(\epsilon(\log 1/\epsilon)^{-1})|X_1|$ such that f is γ-hard-core on S_1 for size $g' = O(\gamma^2(\log(1/\epsilon))^{-1}g)$. Now let $X_2 = X_1 \setminus S_1$. Continue in this fashion, obtaining S_i from X_i as above, until $\sum |S_i| \geqslant \frac{\epsilon}{2}2^n$.

To see that this works, notice that until $\sum |S_i| \geqslant \frac{\epsilon}{2}2^n$ each set X_i satisfies $|X_i| > (1-\frac{\epsilon}{2})2^n$. Thus by Lemma 4.4.17 at each stage we have that f is $\frac{\epsilon}{2}$-hard for size g under $\mathcal{U}_{X_i}$, so we can apply Theorem 4.4.16 each time with hardness parameter $\epsilon/2$. Furthermore, we have that each $|S_i| = \Omega(\epsilon(\log(1/\epsilon))^{-1})|X_i|$ where $|X_i| > (1-\frac{\epsilon}{2})2^n$, so we must achieve $\sum |S_i| > \frac{\epsilon}{2}2^n$ after at most $O(\log 1/\epsilon)$ stages.

Now let $S = \cup_i S_i$. Since the S_i are disjoint we have $|S| \geqslant \frac{\epsilon}{2}2^n$. Moreover, since f is γ-hard-core on each $|S_i|$ for size g', for any circuit C of size at most g' we have

$$\Pr_{\mathcal{U}_S}[C(x) = f(x)] \quad = \quad \sum_i \left(\frac{|S_i|}{|S|} \Pr_{\mathcal{U}_{S_i}}[C(x) = f(x)] \right) \leqslant \frac{1}{2} + \gamma.$$

Thus f is γ-hard-core on S for size g' and the theorem is proved. ■

Remark 1. The reader may have noticed that the two arguments used to go from a hard-core measure to a hard-core set and to increase the size of the hard-core set do not depend in any essential way on the fact that the initial hard-core measure was of size $\mu(M) = \Omega(\epsilon(\log 1/\epsilon)^{-1})$. We could also have obtained the same final result – a hard-core set construction with $\epsilon/2$ as the set size parameter and $O(\gamma^2(\log 1/\epsilon)^{-1})$ as the circuit size parameter – by using the boosting algorithm `F1` which implies the existence of a measure of size $\Omega(\epsilon^3)$ (Theorem 4.4.6). We introduced and analyzed the $\mathtt{B}_{\text{Comb}}$ algorithm in part because this boosting algorithm will play an important role in the results of Section 4.5.

Remark 2. Freund has shown that any algorithm which boosts a $(1/2-\gamma)$-approximate weak learner to accuracy $1-\epsilon$ must combine at least $\Omega(\gamma^{-2}\log(1/\epsilon))$ weak hypotheses in the worst case [45]. Thus, for any hard-core set construction falling within this framework our circuit size parameter is optimal up to constant factors. Furthermore, it is easy to see that any general hard-core set construction such as those we have given must have a set size parameter of at most $O(\epsilon)$, since the original ϵ-hard function f might be very easy to compute (e.g. constant; see [107]) on a $1 - O(\epsilon)$ fraction of inputs. Thus we believe that the hard-core set construction given by Theorem 4.4.18 is optimal up to constant factors with respect to both the circuit size and set size parameters.

4.4.6 A Boosting Algorithm from IHA

We have not yet described just how boosting algorithms manage to simulate the different distributions $\mathcal{D}_i$ for the example oracles $\mathrm{EX}(f, \mathcal{D}_i)$ which are required by the weak learning algorithm at each boosting stage. There are two different types of boosting algorithms, known as *boosting-by-filtering* and *boosting-by-sampling*, which handle this issue in different ways. In boosting-by-filtering the distribution $\mathcal{D}_i$ is simulated from $\mathcal{D}$ by filtering examples received from $\mathrm{EX}(f, \mathcal{D})$. If the filtering process accepts example x with probability $\alpha(x)$ and rejects x (i.e. discards x and makes another call to $\mathrm{EX}(f, \mathcal{D})$) with probability $1 - \alpha(x)$, then it is easy to see that this filtering process defines a new distribution $\mathcal{D}'$ where

$$\mathcal{D}'(x) = \frac{\alpha(x)\mathcal{D}(x)}{\sum_y \alpha(y)\mathcal{D}(y)}. \tag{4.1}$$

The boosting algorithms `F1`, $\mathtt{B}_{\mathrm{Filt}}$ and $\mathtt{B}_{\mathrm{Comb}}$ all are examples of boosting-by-filtering algorithms.

In boosting-by-sampling, on the other hand, a set S of examples is drawn from $\mathrm{EX}(f, \mathcal{D})$ once and for all at the beginning of the boosting process and the initial distribution $\mathcal{D}_0$ is taken to be the uniform distribution over S. Subsequent distributions $\mathcal{D}_i$ are nonzero only on the points of S, and the final hypothesis h generated by boosting has high accuracy with respect to the uniform distribution over S. Well-known results on generalization error [25] imply that if h belongs to a concept class with bounded Vapnik-Chervonenkis dimension, then for sufficiently large S any hypothesis h which is correct on all of S will with high probability have low error under $\mathcal{D}$. The `AdaBoost` algorithm of [46] is an example of a boosting-by-sampling algorithm.

Impagliazzo's proof shows that it is possible to use IHA directly as a $O(1/\epsilon^2\gamma^2)$-stage, $1/\epsilon$-smooth boosting-by-sampling algorithm. boosting-by-sampling Building on this observation, Servedio has given an improved boosting-by-sampling algorithm which is also $1/\epsilon$-smooth but uses only $O(1/\epsilon\gamma^2)$ stages [105]. In the next section we discuss using IHA as a $O(1/\epsilon^2\gamma^2)$-stage, $O(1/\epsilon)$-smooth boosting-by-filtering algorithm in the case where the initial distribution $\mathcal{D}$ is uniform over $\{0,1\}^n$.

4.5 Faster Algorithms for Learning DNF

We have seen that boosting algorithms can be used to improve on previous complexity-theoretic hard-core set constructions. Now we go in the opposite direction and use ideas from hard-core set construction to establish new results in learning theory. We show that the uniform distribution boosting algorithm which is implicit in IHA can be used to significantly improve the asymptotic running time of Jackson's Harmonic Sieve algorithm for learning DNF under the uniform distribution using membership queries. This algorithm is widely viewed as one of the most important results in computational learning theory. We also show how a different modification inspired by our analysis in Section 4.4.3 can improve the running time even further at the cost of learning using more complex hypotheses.

Bshouty, Jackson and Tamon [30] have recently given a variant of the Harmonic Sieve which runs substantially faster than the original algorithm. Their improvement is obtained by speeding up a weak learning algorithm which is a component of the Harmonic Sieve, and is orthogonal to our improvements of the boosting component of the Sieve. As described below, by combining our techniques with their improvements we obtain the fastest known algorithm for learning DNF under the uniform distribution with membership queries.

4.5.1 The DNF Learning Problem

A *disjunctive normal form* (DNF) expression is a disjunction of terms where each term is a conjunction of Boolean literals. Since every Boolean function can be expressed as a DNF, the concept class DNF is the class of all Boolean functions over $\{0,1\}^n$. The *DNF-size* of a function f is the minimum number of terms in any DNF expression for f. Thus an efficient learning algorithm for the concept class DNF must be able to learn any Boolean function in time polynomial in the number of terms in its smallest DNF representation.

In his seminal paper Valiant posed the question of whether there is an efficient PAC algorithm for learning DNF under an arbitrary probability distribution on labeled examples [111]. A recent result [74] shows that there is a PAC learning algorithm which learns to accuracy ϵ and runs in time $2^{O(n^{1/3}\log n \log s)}/\epsilon$ for target concepts of DNF-size s, but it is not yet known whether there is an algorithm which runs in time $\mathrm{poly}(n, s, 1/\epsilon)$. If the learning scenario is suitably modified, though, then efficient learning of DNF becomes possible. In a breakthrough result several years ago Jackson gave the Harmonic Sieve algorithm which uses *membership queries* to learn DNF to accuracy ϵ under the uniform distribution in roughly ns^8/ϵ^{12} time steps [62]. A membership query is an oracle query in which the learner specifies a point $x \in \{0,1\}^n$ and the membership oracle $\mathrm{MEM}(f)$ returns the value $f(x)$.

Although the Harmonic Sieve runs in polynomial time, it is not considered to be computationally practical due to the high degree of the polynomial time bound. We show how to substantially improve the algorithm's time dependence on the error parameter ϵ, thus making progress towards a more efficient implementation.

4.5.2 The Harmonic Sieve

The main result of [62] is the following theorem:

Theorem 4.5.1 [62] *Let f be a Boolean function on $\{0,1\}^n$ of DNF-size s and let $0 < \epsilon, \delta < 1$. For any constant $\rho > 0$, given access to a membership oracle $MEM(f)$ the Harmonic Sieve algorithm runs in time $\tilde{O}(ns^8/\epsilon^{12+\rho})$ and with probability $1-\delta$ outputs a hypothesis h such that $\Pr_{\mathcal{U}}[h(x) \neq f(x)] \leqslant \epsilon$.*

At the heart of Jackson's Harmonic Sieve algorithm is a procedure `WDNF` which was first studied in [22]. The `WDNF` algorithm takes as input an example oracle $\mathrm{EX}(f, \mathcal{D})$, a membership oracle MEM(f), a distribution oracle DIST($\mathcal{D}$) and a value $\delta > 0$. A distribution oracle DIST($\mathcal{D}$) is an oracle which, when queried with a point x in the domain of $\mathcal{D}$, returns the value of $\mathcal{D}(x)$. With probability at least $1 - \delta$ the `WDNF` algorithm outputs a parity function which is a $(1/2 - \Omega(1/s))$-approximator for f under $\mathcal{D}$, where s is the DNF-size of f.[2]

The Harmonic Sieve algorithm works by running Freund's boosting algorithm `F1` with `WDNF` as the weak learning algorithm. At the i-th stage of boosting the `F1` boosting algorithm simulates some distribution $\mathcal{D}_i$ and uses the simulated example oracle $EX(f, \mathcal{D}_i)$ for `WDNF`. Jackson shows that the `F1` boosting algorithm constructs each distribution $\mathcal{D}_i$ in such a way that after a "one-time" initial cost of $\tilde{O}(L_\infty(2^n\mathcal{D}_i)^6)$ time steps (to estimate the scaling factor in the denominator of Equation 4.1) for each $\mathcal{D}_i$, it is possible to simulate a constant-factor approximation DIST$'(\mathcal{D}_i)$ of DIST$(\mathcal{D}_i)$ in $O(n)$ time steps per call.[3] The following lemma is a direct consequence of Jackson's Lemma 9 and the analysis used in its proof:

Lemma 4.5.2 [62] *Let f be any Boolean function of DNF-size s over $\{0,1\}^n$ and let $\mathcal{D}$ be any distribution over $\{0,1\}^n$. If* `WDNF` *is run using EX($f, \mathcal{D}$), MEM(f) and DIST$'(\mathcal{D})$ as its oracles, where each call to EX($f, \mathcal{D}$) takes t time steps, each call to MEM(f) takes 1 time step, and the time requirements of DIST$'(\mathcal{D})$ are as described above, then* `WDNF` *runs in time $\tilde{O}(s^4(L_\infty(2^n\mathcal{D}))^2 t + ns^6(L_\infty(2^n\mathcal{D}))^6)$ time steps and with probability at least $1 - \delta$ outputs a parity function which is a $(1/2 - \Omega(1/s))$-approximator to f under $\mathcal{D}$.*

Jackson proves that each call to EX($f, \mathcal{D}_i$) made by `WDNF` can be simulated in time $\tilde{O}(nL_\infty(2^n\mathcal{D}_i)^3)$ with high probability. Consequently the time bound of Lemma 4.5.2 is dominated by the $ns^6(L_\infty(2^n\mathcal{D}))^6$ term, and thus the time required for the i-th execution of `WDNF` on distribution $\mathcal{D}_i$ is bounded by $\tilde{O}(ns^6(L_\infty(2^n\mathcal{D}))^6)$. As noted in Section 4.4.3, for any $\rho > 0$ algorithm `F1` can be shown to be an $O(\gamma^{-2}\log(1/\epsilon))$-stage, $O(1/\epsilon^{2+\rho})$-smooth boosting algorithm, so $L_\infty(2^n\mathcal{D}_i) = O(1/\epsilon^{2+\rho})$ for every

[2] For $A \subseteq \{1,\dots,n\}$ the parity function $\chi_A : \{0,1\}^n \to \{0,1\}$ is $\chi_A(x) = \sum_{i\in A} x_i \bmod 2$.

[3] To be more precise, the oracle DIST$'(\mathcal{D}_i)$ is such that $\frac{1}{2}\mathrm{DIST}(\mathcal{D}_i)(x) \leqslant \mathrm{DIST}'(\mathcal{D}_i)(x) \leqslant \frac{3}{2}\mathrm{DIST}(\mathcal{D}_i)(x)$.

distribution $\mathcal{D}_i$ which WDNF uses. Since $\gamma = \Omega(1/s)$ for the WDNF algorithm, all in all the Harmonic Sieve algorithm runs in time $\tilde{O}(ns^8/\epsilon^{12+\rho})$ for any $\rho > 0$. The hypotheses output by the Harmonic Sieve are majority-of-parity circuits since each weak hypothesis is a parity circuit and F1's hypothesis is a majority circuit over weak hypotheses.

4.5.3 A Faster Version of the Harmonic Sieve

As described above, the Harmonic Sieve algorithm works by boosting under the uniform distribution and its running time strongly depends on the smoothness of the boosting algorithm. The following observation follows directly from the discussion of IHA in Section 4.2:

Observation 4.5.3 *For each measure M_i constructed in the execution of* IHA *the distribution $\mathcal{D}_{M_i}$ is $1/\epsilon$-smooth.*

This is substantially better than the distributions constructed by F1 which are guaranteed only to be $(1/\epsilon^{2+\rho})$-smooth. Thus, it appears that we can use the better boundedness of the IHA algorithm to obtain a faster version of the Harmonic Sieve, and indeed this turns out to be the case; however some details need to be addressed as described below.

One detail is that since the running time of WDNF depends on the quantity $L_\infty(2^n\mathcal{D}_i)$, where $\mathcal{D}_i$ is a distribution over $\{0,1\}^n$, we need a version of IHA which works efficiently over the entire domain $\{0,1\}^n$. Another way of saying this is that we need a boost-by-filtering version of IHA. Doing an exact computation of $\mu(M_i)$ in line 3 of IHA would take exponential time for the domain $\{0,1\}^n$, so our boost-by-sampling version of IHA instead estimates the value of $\mu(M_i)$ by using a sample average. More precisely, the algorithm draws a collection of uniformly distributed x's from $\{0,1\}^n$ and uses the observed average value of $M_i(x)$ on this sample as its estimate for $\mu(M_i)$. It is easy to see that $\mu(M_i)$ is the expected value of $M_i(x)$ for uniformly chosen x; standard bounds on sampling (e.g. Corollary 2.2 of [61]) show that with very high probability an estimate $\mu'(M_i)$ satisfying $\frac{1}{2}\mu(M_i) \leqslant \mu'(M_i) \leqslant \frac{3}{2}\mu(M_i)$ can be found using $\tilde{O}(1/\mu(M_i)^2)$ samples. Thus in the boost-by-sampling version of `IHA` the test in line 3 can be approximately performed in $\tilde{O}(1/\epsilon^2)$ time steps, and the resulting algorithm will be $2/\epsilon$-smooth with high probability. We note that the $\tilde{O}(1/\mu(M_i)^2)$ time steps required to perform this estimation for each measure M_i corresponds exactly to the "one-time cost" of $\tilde{O}(L_\infty(2^n\mathcal{D}_i)^6)$ for estimating the denominator of Equation 4.1 which was required for each distribution $\mathcal{D}_i$ simulated by `F1`.

Another detail which must be taken into account is the slowdown incurred in the process of filtering examples from EX$(f,\mathcal{U})$ to simulate draws from $EX(f,\mathcal{D}_i)$. It is not difficult to see that if a measure M_i satisfies $\mu(M_i) \geqslant \epsilon/2$, then on average at most $2/\epsilon$ draws from$EX(f,\mathcal{U})$ are required to simulate a draw from EX$(f,\mathcal{D}_{M_i})$, and with very high probability no more than $\tilde{O}(1/\epsilon)$ draws are required. Thus this slowdown, which is precisely the factor t in Lemma 4.5.2, is at most $\tilde{O}(1/\epsilon)$.

Thus `IHA` can be translated into a $O(1/\gamma^2\epsilon^2)$-stage, $O(1/\epsilon)$-smooth boost-by-filtering algorithm under the uniform distribution. This boosting algorithm can be used instead of `F1` in the top layer of the Harmonic Sieve; we call this modified algorithm `HS'`. Moreover, as discussed above Lemma 4.5.2 implies that each execution of `WDNF` in `HS'`will take at most $\tilde{O}(ns^6/\epsilon^6)$ time steps. Thus, although `HS'` requires a factor of $\tilde{\Omega}(1/\epsilon^2)$ more boosting stages than the original Sieve, this disadvantage is more than offset by the improved runtime of `WDNF`. We obtain the following:

Theorem 4.5.4 *There is a membership-query algorithm* `HS'` *for learning DNF under the uniform distribution which runs in time $\tilde{O}(ns^8/\epsilon^8)$. The algorithm outputs as its final hypothesis a majority-of-parity circuit.*

We can achieve an even faster variant of the Harmonic Sieve, at the price of using more complex hypotheses, by using the $\mathtt{B}_{\text{comb}}$ boosting algorithm instead of the boost-by-filtering `IHA` algorithm. As noted in Section 4.4.2, $\mathtt{B}_{\text{Comb}}$ is an $O(\gamma^{-2}\log(1/\epsilon))$-stage, $O(\log(1/\epsilon)/\epsilon)$-smooth boost-by-filtering algorithm. The smoothness of $\mathtt{B}_{\text{Comb}}$ implies that the "one-time cost" of computing the scaling factor for each distribution $\mathcal{D}_i$ is now $\tilde{O}(L_\infty(2^n\mathcal{D}_i)^2) = \tilde{O}(1/\epsilon^2)$. Furthermore, the time overhead factor for simulating each call to $EX(f,\mathcal{D}_i)$ is now $t = \tilde{O}(L_\infty(2^n\mathcal{D}_i)) = \tilde{O}(1/\epsilon)$. From

Lemma 4.5.2 we find that if $\mathtt{B}_{\text{Comb}}$ is used as the boosting algorithm in the Sieve, then the running time of each boosting stage will be at most $\tilde{O}(ns^6/\epsilon^6)$. Since we boost for at most $O(s^2 \log(1/\epsilon))$ stages under $\mathtt{B}_{\text{comb}}$, we have the following theorem:

Theorem 4.5.5 *There is a membership-query algorithm for learning s-term DNF formulae under the uniform distribution on $\{0,1\}^n$ which runs in time $\tilde{O}(ns^8/\epsilon^6)$. The algorithm outputs as its final hypothesis a majority-of-majority-of-parity circuit.*

The additional circuit complexity comes from the fact that the hypothesis output by $\mathtt{B}_{\text{Comb}}$ is a depth 2 majority circuit over its inputs.

4.5.4 Extensions

Throughout this section we have only discussed using the Harmonic Sieve to learn DNF formulae under the uniform distribution. Jackson generalizes the algorithm to several other concept classes including TOP (polynomial-size depth-2 circuits where the top gate computes majority and the bottom gates compute parities) and unions of axis-parallel rectangles over $\{0, 1, \ldots, b\}^n$ for constant b. In each case our approach can be used to improve the running time of Jackson's algorithms.

In recent work Bshouty, Jackson and Tamon [30] have given a new version of the Harmonic Sieve for learning DNF under the uniform distribution. The new algorithm differs from the original Harmonic Sieve in that it uses a faster version of the `WDNF` algorithm. Implicit in their analysis is the following lemma:

Lemma 4.5.6 [30] *Let f be any Boolean function of DNF-size s over $\{0,1\}^n$ and let $\mathcal{D}$ be any distribution over $\{0,1\}^n$. There is an algorithm* `WDNF′` *which takes as input an example oracle $EX(f,\mathcal{D})$, a membership oracle $MEM(f)$, a distribution oracle $DIST'(\mathcal{D})$, and a value $\delta > 0$. In the context of the Harmonic Sieve, each invocation of* `WDNF′` *with the i-th distribution $\mathcal{D}_i$ generated by the boosting algorithm takes $\tilde{O}(ns^2(L_\infty(2^n\mathcal{D}_i))^2)$ time steps and and with probability at least $1-\delta$ outputs a parity function which is a $(1/2 - \Omega(1/s))$-approximator to f under $\mathcal{D}_i$.*

The Harmonic Sieve variant described in [30] runs the original $O(1/\epsilon^{2+\rho})$-smooth `F1` boosting algorithm for $\tilde{O}(s^2)$ stages, using the new `WDNF'` algorithm as the weak learner, to obtain an overall running time of essentially $\tilde{O}(ns^4/\epsilon^4)$. By instead using the $O(\log(1/\epsilon)/\gamma^2)$-stage, $O(\log(1/\epsilon)/\epsilon)$-smooth boosting algorithm $\mathtt{B}_{\mathrm{comb}}$ as in Section 4.5.3, we obtain the following result, which is the fastest known algorithm for learning DNF under the uniform distribution using membership queries:

Theorem 4.5.7 *There is a membership-query algorithm for learning s-term DNF formulae over* $\{0,1\}^n$ *under the uniform distribution which runs in time* $\tilde{O}(ns^4/\epsilon^2)$. *The algorithm outputs as its final hypothesis a majority-of-majority-of-parity circuit.*

Chapter 5

Learnability Beyond AC^0

5.1 Introduction

5.1.1 Motivation

In his seminal 1984 paper "A Theory of the Learnable" Valiant proposed the Probably Approximately Correct (PAC) model of learning in an attempt to understand what programs a machine can acquire without explicit instruction. A central question that arises from this paradigm is the following: what is the most expressive class of Boolean functions that can be learned by a computationally efficient algorithm? As stated by Valiant, "The results of *learnability theory* ... indicate the maximum granularity of the single concepts that can be acquired without programming" [111]. Working towards this goal, researchers in computational learning theory have developed many algorithms for learning various classes of Boolean circuits. This chapter gives a new algorithm to learn the most expressive class of Boolean circuits considered to date for a well-studied variant of Valiant's model.

5.1.2 Previous Results

In Valiant's original model, a class of functions is said to be efficiently learnable if there is a polynomial-time algorithm which succeeds given labeled examples drawn from an arbitrary probability distribution. Unfortunately, few classes are known to be efficiently learnable in this sense. Hence researchers have considered algorithms which run in superpolynomial time and/or succeed under specific distributions on examples such as the uniform distribution.

Several subexponential time algorithms are known for learning interesting classes in the distribution-free model where the learning algorithm must succeed under an arbitrary probability distribution. Ehrenfeucht and Haussler [41] and Blum [18] have given $n^{\log s}$ time algorithms for learning size-s decision trees over n Boolean variables. Improving on results of Bshouty [28] and Tarui and Tsukiji [109], Klivans and Servedio [74] have given an algorithm for learning s-term Disjunctive Normal Form (DNF) formulae in time $2^{O(n^{1/3} \log s \log n)}$.

Stronger results can be achieved by requiring the learning algorithm to succeed only when given examples drawn from the uniform distribution on the Boolean cube. One such uniform distribution learning algorithm is Jackson's Harmonic Sieve algorithm for learning s-term DNF formulae [62]. The Sieve runs in poly(n, s) time but requires that the learning algorithm be allowed to make *membership queries* for the value of the target function on specified points. Another well-known uniform distribution learning algorithm is the algorithm of Linial *et al.* [81] which learns size-s depth-d AND/OR/NOT circuits in time $n^{(\log s)^d}$ without membership queries. This result is noteworthy for several reasons: (1) The class AC^0 of constant-depth polynomial-size circuits is highly expressive relative to other circuit classes studied in learning theory – note that DNF formulae are AC^0 circuits of depth 2. (2) Kharitonov [64] gave compelling evidence that the algorithm of [81] is essentially optimal by showing that an $n^{(\log s)^{o(d)}}$-time algorithm for learning size-s depth-d circuits under uniform would contradict a plausible cryptographic assumption about the hardness of integer factorization.

5.1.3 Our Results

We give the first algorithm to learn constant-depth circuits augmented with majority gates. Although Kharitonov proved that the running time of Linial *et al.*'s algorithm is essentially optimal for constant-depth AND/OR/NOT circuits, our results show that it is possible to learn a strictly more expressive class of circuits in the same amount of time.[1] We also show that under Kharitonov's cryptographic assumption our algorithm is essentially optimal with respect to both running time and expressiveness (number of majority gates) of the functions being learned.

We believe that augmenting constant depth circuits with a limited number of majority gates is an interesting and natural extension of AC^0. Such circuits have been studied by several researchers in complexity theory [5, 11, 14] and represent a significant extension of AC^0 (recall that a single majority of n variables requires depth-d circuits of size $2^{n^{1/d}}$ [54]). Our main positive result for these circuits can be stated informally as follows (a precise statement is given in Section 5.3.3):

Theorem 5.1.1 *Quasipolynomial ($2^{poly \log n}$) size constant-depth circuits which contain a polylogarithmic number of majority gates can be learned under the uniform distribution in quasipolynomial time from random examples only.*

As we allow constant depth circuits to contain more and more majority gates we move toward the circuit complexity class TC^0 which can be viewed as AC^0 augmented with a polynomial number of majority gates. TC^0 is a highly expressive class which contains functions such as integer multiplication and division – indeed it is currently conceivable that all of $\#\mathsf{P}$ is contained in non-uniform TC^0. Naor and Reingold [90] have constructed pseudo-random functions in TC^0 (under a widely held cryptographic assumption) and thus it seems unlikely that effective learning algorithms can be given for TC^0. Using results from [90] we establish the following lower bound for learning which holds under Kharitonov's plausible cryptographic assumption (a precise statement is given in Section 5.4):

Theorem 5.1.2 *Any algorithm that even weakly learns depth-5 circuits containing more than a polylogarithmic number of majority gates under the uniform distribution must run in more than quasipolynomial time, even if membership queries are allowed.*

[1] We show in Section 5.3.4 that our algorithm runs slightly faster asymptotically than the Linial *et al.* algorithm for learning standard constant-depth AND/OR/NOT circuits.

Theorems 5.1.1 and 5.1.2 together give a fairly complete picture of the complexity of learning constant depth circuits which contain majority gates.

5.1.4 Our Approach

The learning algorithm of Linial *et al.* for AC^0 is based on the fact that any function computed by a constant-depth polynomial-size circuit can be well approximated by the low-order terms in its Fourier representation. More precisely, the main lemma of [81] shows that any AC^0 function on n variables can have at most an ϵ fraction of its Fourier spectrum weight on sets which contain more than $\text{polylog}(n/\epsilon)$ variables. This property need not hold for circuits which contain majority gates; Bshouty and Tamon [32] have shown that the majority function itself has an $\Omega(1/\sqrt{n})$ fraction of its Fourier spectrum weight on sets of size $\Omega(\sqrt{n})$. Hence a different approach seems to be required for circuits which contain majority gates.

Our algorithm learns using a stagewise approach based on hypothesis boosting. Boosting is a well known technique in computational learning theory which can be used to transform a weak learning algorithm (which constructs an approximator to the function being learned whose error is only slightly less than 1/2) into a strong learning algorithm which generates a highly accurate hypothesis. Boosting algorithms accomplish this by running the weak learning algorithm repeatedly on a sequence of carefully constructed distributions.

We show (Lemma 5.3.3) that for any function f in our class and any distribution $\mathcal{D}$ which is not too far from uniform, there exists a low-order parity function which is weakly correlated with f. Since this parity depends on only a small number of variables it can be found using exhaustive search. As we show in Section 5.3, the size bound for this parity – and hence the running time of our algorithm – depends directly on the extent of $\mathcal{D}$'s deviation from the uniform distribution. Fortunately, boosting algorithms are known which have the property that if the initial distribution over examples is uniform then the distributions constructed by the boosting algorithm will not deviate too far from uniform. Using such a boosting algorithm we transform our weak learning algorithm into a strong learning algorithm which generates a highly accurate approximator to the unknown target function.

The form of the hypothesis output by our algorithm is a majority over quasipolynomially many parity functions. Since the Harmonic Sieve is already capable of learning this class in quasipolynomial time [62], it is natural to ask whether our new algorithm is learning a class that could not be learned by the existing Sieve. Several points should be noted. First, unlike the Sieve, our algorithm does not require membership queries, which is a significant advantage. Second, while our results imply that the Sieve is capable of learning AC^0 circuits augmented with a polylogarithmic number of majority gates, this was not known prior to the analysis contained in this chapter. In particular, the most general prior results [62] showed that the Sieve was capable of learning a majority of s parity functions in time polynomial in s. However, since even the class AC^0 itself is known to contain a linear-size circuit C such that any majority of parities representing C exactly contains exponentially many distinct parity functions [77], these prior results do not imply our results.

5.1.5 Learning Other Extensions of AC^0

Another natural question raised by our work is whether it is possible to efficiently learn constant-depth circuits which contain a limited number of expressive gates of various types other than majority. In Section 5.5 we consider learning constant depth $AND/OR/NOT$ circuits augmented with a single additional gate at the root which may be either a parity gate, a MOD_p gate for prime $p > 2$, or a threshold gate which computes an arbitrary weighted majority of its inputs. We give evidence that with respect to known techniques each of these three simple extensions of AC^0 may be difficult to learn.

5.1.6 Organization

In Section 5.3 we give our algorithm for learning constant-depth circuits with majority gates. In Section 5.4 we give cryptographic hardness results for learning constant-depth circuits with majority gates. Section 5.5 discusses prospects for learning other extensions of AC^0.

5.2 Preliminaries

We write $[n]$ to denote the set $\{1, \ldots, n\}$, $\mathcal{U}$ to denote the uniform distribution over $\{-1,1\}^n$ and $\log x$ to denote $\log_2 x$. We view Boolean functions as mappings from $\{-1,1\}^n$ to $\{-1,1\}$ where -1 represents TRUE and 1 represents FALSE. Unless otherwise indicated probabilities and expectations are with respect to $\mathcal{U}$.

Recall that AC^0 is the class of constant-depth polynomial-size circuits composed of unbounded fan-in AND, OR and NOT gates. The majority function $MAJ : \{-1,1\}^n \to \{-1,1\}$ is defined by $MAJ(x) = \text{sign}(x_1 + \cdots + x_n)$ which is 1 if and only if $x_1 + \cdots + x_n \geqslant 0$. We write MAC^0 to denote the class of constant-depth polynomial-size circuits which have a single unbounded fan-in majority gate at the root, and we refer to such circuits as *majority of* AC^0.

For $A \subseteq [n]$ the parity function $\chi_A : \{-1,1\}^n \to \{-1,1\}$ is defined by $\chi_A(x) = \prod_{i \in A} x_i$. As is well known, with inner product $\langle f, g \rangle = E[fg]$ and norm $\|f\| = \sqrt{E[f^2]}$ the 2^n parity functions $\{\chi_A\}_{A \subseteq [n]}$ form an orthonormal basis for the vector space of real-valued functions on $\{-1,1\}^n$. Consequently every real-valued function $f : \{-1,1\}^n \to \{-1,1\}$ can be uniquely expressed as a linear combination $f(x) = \sum_{A \subseteq [n]} \hat{f}(A)\chi_A(x)$. The coefficients $\hat{f}(A)$ are known as the *Fourier spectrum* of f. Orthonormality implies that $\hat{f}(A) = \langle f, \chi_A \rangle$ and thus $\hat{f}(A)$ measures the correlation between f and χ_A. Orthonormality also implies *Parseval's identity*, which states that for every real-valued function f we have $E[f^2] = \sum_{A \subseteq [n]} \hat{f}(A)^2$. For Boolean functions we thus have $\sum_{A \subseteq [n]} \hat{f}(A)^2 = 1$. The following easily verified fact is from [81]:

Fact 5.2.1 *For any Boolean function f and real-valued function g,*

$$\begin{aligned}\Pr[f(x) \neq sign(g(x))] &\leqslant E[(f(x) - g(x))^2] \\ &= \sum_{A \subseteq [n]} (\hat{f}(A) - \hat{g}(A))^2.\end{aligned}$$

The equality follows from Parseval's identity and the linearity of the Fourier transform.

Finally we note the following simple but useful fact: for any distribution $\mathcal{D}$, if f, h are functions which take values ± 1 then $\Pr_{\mathcal{D}}[f = h] = 1/2 + E_{\mathcal{D}}[fh]/2$.

5.2.1 The Learning Model

The learning model we consider is a well-studied distribution specific variant of Valiant's Probably Approximately Correct (PAC) learning model [111]. Let f be a Boolean function and $\mathcal{D}$ a probability distribution on $\{-1,1\}^n$. We write $EX(f, \mathcal{D})$ to denote a random example oracle which, when invoked, outputs a labeled example $\langle x, f(x)\rangle$ where $x \in \{-1,1\}^n$ is chosen according to $\mathcal{D}$.

Let C be a class of Boolean functions on $\{-1,1\}^n$. Let $f \in C$ be an unknown member of C and let A be a learning algorithm which takes as input accuracy and confidence parameters ϵ, δ and can invoke the oracle $EX(f, \mathcal{D})$. We say that A *learns C under the uniform distribution* if A satisfies the following condition: for all $0 < \epsilon, \delta < 1$, if A is given access to $EX(f, \mathcal{U})$ then with probability at least $1-\delta$ A outputs a hypothesis $h : \{-1,1\}^n \to \{-1,1\}$ such that $\Pr[f(x) = h(x)] \geqslant 1 - \epsilon$.

In order to obtain our uniform distribution learning algorithm we will need to consider *weak* learning algorithms which work for a broader range of distributions but can only generate hypotheses which are slightly more accurate than random guessing. For $\gamma > 0$ we say that A is a *weak learning algorithm for C with advantage* γ if A satisfies the following condition: for any distribution $\mathcal{D}$, for any $0 < \delta < 1$, if A is given access to $EX(f, \mathcal{D})$ then with probability at least $1 - \delta$ A outputs a hypothesis $h : \{-1,1\}^n \to \{-1,1\}$ such that $\Pr_{\mathcal{D}}[f(x) = h(x)] \geqslant 1/2 + \gamma$. As we will see the advantage γ can depend on parameters such as n, the size of a circuit for f, and the L_∞ norm of the distribution $\mathcal{D}$.

A *boosting algorithm* `B` is an algorithm which satisfies the following condition: for any Boolean function f and any distribution $\mathcal{D}$ over $\{-1,1\}^n$, if `B` is given $0 < \epsilon, \delta < 1$, $0 < \gamma < 1/2$, an example oracle $\mathrm{EX}(f, \mathcal{D})$ and a weak learning algorithm `WL` with advantage γ, then with probability at least $1 - \delta$ algorithm `B` outputs a hypothesis h

such that $\Pr_{\mathcal{D}}[f(x) = h(x)] \geqslant 1 - \epsilon$. We say that a *canonical boosting algorithm* B is a boosting algorithm which has the following iterative structure:

- At stage 0 B starts with $\mathcal{D}_0 = \mathcal{D}$ and uses WL to generate with high probability a hypothesis h_0 satisfying $\Pr_{\mathcal{D}_0}[f(x) = h(x)] \geqslant 1/2 + \gamma$.

- At stage i B does two things: (1) constructs a new distribution $\mathcal{D}_i$ which favors points where the previous hypotheses $h_0, \ldots, h_{i-1}$ do poorly at predicting the value of f, and (2) runs WL using the simuated example oracle $\mathrm{EX}(f, \mathcal{D}_i)$ to produce with high probability a hypothesis h_i satisfying $\Pr_{\mathcal{D}_i}[f(x) = h_i)] \geqslant 1/2 + \gamma$.

- Finally, after doing this repeatedly for some number T of stages, B combines the hypotheses $h_0, \ldots, h_{T-1}$ in some way to obtain with high probability a strong final hypothesis h satisfying $\Pr_{\mathcal{D}}[f(x) = h(x)] \geqslant 1 - \epsilon$.

5.3 The algorithm

We first present a weak learning algorithm for the class of constant-depth circuits with a single majority gate at the root (Section 5.3.1). In Section 5.3.2 we show how to boost this weak learning algorithm into a strong uniform distribution learning algorithm for MAC^0. In Section 5.3.3 we observe that our algorithm can in fact be used to learn constant-depth circuits which contain several majority gates. Finally in Section 5.3.4 we show that the new algorithm for MAC^0 can be used to improve previous results of Linial *et al.* for learning and approximating AC^0.

5.3.1 Weak learning MAC^0 under smooth distributions

Let $f : \{-1,1\}^n \rightarrow \{-1,1\}$ be computed by an MAC^0 circuit of size s and depth $d+1$, so $f = MAJ(C_1, \ldots, C_t)$ where $t \leqslant s$ and each $C_i : \{-1,1\}^n \rightarrow \{-1,1\}$ is a circuit of depth at most d and size at most s. Without loss of generality we can assume that $C_1(x) + \ldots + C_t(x) \neq 0$ for all $x \in \{-1,1\}^n$. (If this sum is ever zero we can reexpress f as $MAJ(C_1, \ldots, C_t, C_1, \ldots, C_t, 1)$. This leaves the depth unchanged and at most doubles the size which makes no difference for our asymptotic results.) We recall the "discriminator lemma" of Hajnal *et al.* [53]:

Lemma 5.3.1 *Let $\mathcal{H}$ be a class of ± 1-valued functions and let $f : \{-1,1\}^n \rightarrow \{-1,1\}$ be expressible as $MAJ(h_1, \ldots, h_t)$ where each $h_i \in \mathcal{H}$ and $h_1(x)+\cdots+h_t(x) \neq 0$ for all x. Then for any distribution $\mathcal{D}$ over $\{-1,1\}^n$ there is some $h \in \mathcal{H}$ such that $|E_{\mathcal{D}}[fh]| \geqslant 1/t$.*

Hence for any distribution $\mathcal{D}$ there is a size-s depth-d circuit $C_{\mathcal{D}}$ such that $E_{\mathcal{D}}[fC_{\mathcal{D}}] \geqslant 1/s$.

Our goal is to show that as long as $\mathcal{D}$ does not assign too much probability weight to any single point then there is some parity function χ_A (where A is small as described below) which has nonnegligible correlation with f under $\mathcal{D}$. Toward this end we use the main lemma of Linial *et al.* [81]:

Lemma 5.3.2 *Let C be a circuit of size s and depth d. Then for any integer k,*

$$\sum_{|A|>k} \hat{C}(A)^2 \leqslant 2s2^{-k^{1/d}/20}.$$

Let $\mathcal{D}$ be any probability distribution on $\{-1,1\}^n$. Let $k = (20\log(8s^3 2^n L_\infty(\mathcal{D})))^d$ and let $g : \{-1,1\}^n \rightarrow \mathbb{R}$ be defined by $g(x) = \sum_{|A|\leqslant k} \hat{C}_{\mathcal{D}}(A)\chi_A(x)$. Then by Parseval's identity and Lemma 5.3.2,

$$\begin{aligned} E[(g(x) - C_{\mathcal{D}}(x))^2] &= \sum_{A\subseteq[n]} (\hat{g}(A) - \hat{C}_{\mathcal{D}}(A))^2 \\ &= \sum_{|A|>k} \hat{C}_{\mathcal{D}}(A)^2 \\ &\leqslant \frac{1}{4s^2 2^n L_\infty(\mathcal{D})}. \end{aligned}$$

Since $\mathcal{D}(x) \leqslant L_\infty(\mathcal{D})$ for all x, we have $E_{\mathcal{D}}[(g(x) - C_{\mathcal{D}}(x))^2] \leqslant 1/4s^2$. Since $E_{\mathcal{D}}[X^2] \geqslant E_{\mathcal{D}}[X]^2$ for any distribution $\mathcal{D}$ and any random variable X, we have $E_{\mathcal{D}}[|g(x) -$

$C_{\mathcal{D}}(x)|] \leqslant 1/2s$. Thus

$$\begin{aligned} E_{\mathcal{D}}[fg] &= E_{\mathcal{D}}[fC_{\mathcal{D}}] + E_{\mathcal{D}}[f(g - C_{\mathcal{D}})] \\ &\geqslant E_{\mathcal{D}}[fC_{\mathcal{D}}] - E_{\mathcal{D}}[|g - C_{\mathcal{D}}|] \\ &\geqslant 1/s - 1/2s \qquad (5.1) \\ &= 1/2s \qquad (5.2) \end{aligned}$$

where the first inequality holds because the range of f is $\{-1, 1\}$. Thus the real-valued function g has nonnegligible correlation with the MAC^0 function f under $\mathcal{D}$.

Using linearity of expectation, we have

$$\begin{aligned} \sum_{|A| \leqslant k} |\hat{C}_{\mathcal{D}}(A)||E_{\mathcal{D}}[f\chi_A]| &\geqslant \sum_{|A| \leqslant k} \hat{C}_{\mathcal{D}}(A) E_{\mathcal{D}}[f\chi_A] \\ &= E_{\mathcal{D}}[f \sum_{|A| \leqslant k} \hat{C}_{\mathcal{D}}(A)\chi_A] \\ &= E_{\mathcal{D}}[fg] \\ &\geqslant 1/(2s). \end{aligned}$$

Since each Fourier coefficient $\hat{C}_{\mathcal{D}}(A)$ has magnitude at most 1, we have that $\sum_{|A| \leqslant k} |\hat{C}_{\mathcal{D}}(A)|$ is at most n^k, and thus $|E_{\mathcal{D}}[f\chi_A]| \geqslant 1/(2sn^k)$ for some $|A| \leqslant k$. Thus we have proved the following lemma:

Lemma 5.3.3 *There is a subset $A \subset [n]$ with $|A| \leqslant k$ such that $|\Pr_{\mathcal{D}}[f(x) = \chi_A(x)] - \frac{1}{2}| \geqslant 1/(4sn^k)$ where $k = (20 \log(8s^3 2^n L_\infty(\mathcal{D})))^d$.*

We can now establish the main result of this section:

Theorem 5.3.4 *Let $C_{s,d}$ be the class of MAC^0 circuits of size s and depth $d + 1$ over $\{-1, 1\}^n$. Let $\mathcal{D}$ be any distribution and let $k = (20 \log(8s^3 2^n L_\infty(\mathcal{D})))^d$. There is a weak learning algorithm for $C_{s,d}$ with advantage $\gamma = 1/(8sn^k)$ which runs in time poly$(n^k, \log 1/\delta)$. The algorithm takes δ and γ as input and outputs a hypothesis which is either χ_A or $-\chi_A$ for some $|A| \leqslant k$.*

Proof: Lemma 5.3.3 implies that for some subset A of size at most k either χ_A or $-\chi_A$ will be the desired weak hypothesis. The weak learning algorithm does an exhaustive search over all $A \subset [n]$ in order of increasing size. For each candidate parity function χ_A the algorithm draws a sample of labeled examples from $EX(f, \mathcal{D})$ and uses this sample to compute an empirical estimate of $\Pr_{\mathcal{D}}[f(x) = \chi_A(x)]$. Suppose that the candidate parity being tested is $\chi_{A'}$. Using Chernoff bounds it can be shown that if a sample of size $\mathrm{poly}(1/\gamma, \log 1/\delta')$ is used then with probability at least $1 - \delta'$ the empirical estimate of $\Pr_{\mathcal{D}}[f(x) = \chi_{A'}(x)]$ will differ from the true value by at most $\gamma/2 = 1/(16sn^k)$. The algorithm halts and outputs $\chi_{A'}$ (or $-\chi_{A'}$) if and only if the empirical estimate p satisfies $|p - 1/2| \geqslant 3\gamma/2 = 3/(16sn^k)$. It is easily verified that if δ' is taken to be $\delta/2^n$ then with probability at least $1 - \delta$ this algorithm runs in time $\mathrm{poly}(sn^k, \log(2^n/\delta)) = \mathrm{poly}(n^k, \log 1/\delta)$ and succeeds as stated in the theorem. ■

5.3.2 Learning MAC^0 under the uniform distribution

To obtain a strong learning algorithm we will use the following theorem whose proof can be found in [73, 106]:[2]

Theorem 5.3.5 *There exists a canonical boosting algorithm* `B` *which, given input parameters* $0 < \epsilon, \delta < 1$, $0 < \gamma < 1/2$, *an example oracle* $EX(f, \mathcal{U})$ *and a weak learning algorithm* `WL` *with advantage* γ, *has the following properties:*

- `B` *runs for* $T = O(1/\gamma^2\epsilon^2)$ *stages;*
- *Each distribution* $\mathcal{D}_i$ *generated by* `B` *satisfies* $L_\infty(\mathcal{D}_i) = O((1/\epsilon)2^{-n})$*;*
- *The running time of* `B` *is polynomial in* $1/\gamma$, $1/\epsilon$, $\log 1/\delta$, n, *and* t, *where* t *is the running time of* `WL` *given access to* $EX(f, \mathcal{U})$;
- *The final hypothesis output by* `B` *is a majority* $h = MAJ(h_1, \ldots, h_{T-1})$ *of the hypotheses* h_i *generated by* `WL`.

[2]Similar results can be obtained from other boosting algorithms such as [29, 36, 43]. Each of these boosting algorithms gives slightly different bounds on T and $L_\infty(\mathcal{D}_i)$ but these differences in the bounds do not affect the running time of our overall algorithm.

We use this boosting algorithm in conjunction with the weak learning algorithm of Section 5.3.1 to construct a quasipolynomial time uniform distribution learning algorithm for MAC^0. Let $\epsilon > 0$ be the desired accuracy and confidence to which we will learn the MAC^0 circuit of size s and depth $d+1$. Every distribution $\mathcal{D}_i$ constructed by $\mathtt{B}$ satisfies $L_\infty(\mathcal{D}_i) = O((1/\epsilon)2^{-n})$. For each such distribution $\mathcal{D}_i$ Theorem 5.3.4 implies that our weak learning algorithm will run in time $\mathrm{poly}(n^k, \log(1/\delta))$ and outputs a hypothesis with advantage $\gamma = 1/(8sn^k)$ where $k = O(\log^d(s/\epsilon))$. Theorem 5.3.5 implies that after $O(1/\gamma^2\epsilon^2)$ iterations with probability $1-\delta$ $\mathtt{B}$ outputs a hypothesis h such that $\Pr_{\mathcal{U}}[f(x) = h(x)] \geqslant 1-\epsilon$. This hypothesis is a majority of $O(1/\gamma^2\epsilon^2)$ parity functions, each of size at most k.

Note that although we assumed that the value of γ is known to the boosting algorithm, this assumption can be removed by using a standard "guess-and-double" technique. The idea is to start with a constant-value guess for γ (e.g. $\gamma = 1/4$) and then run the algorithm with this value. If the algorithm fails to construct a strong hypothesis (which is easily tested), the value of γ is halved and the algorithm is run again. A standard analysis which we omit shows that this process incurs at most a polynomial increase in the run-time bound and hypothesis size. Thus the overall learning algorithm does not need to be given size or depth bounds of the MAC^0 circuit to be learned.

Summarizing the above discussion, we have the following theorem:

Theorem 5.3.6 *Let $C_{s,d}$ be the class of MAC^0 circuits of size s and depth $d+1$ over $\{-1,1\}^n$. The above algorithm learns $C_{s,d}$ to accuracy ϵ under the uniform distribution in time poly$(n^{\log^d(s/\epsilon)}, \log 1/\delta)$.*[3]

5.3.3 Learning constant-depth circuits with majority gates

Using the following result of Beigel [11] we can extend our learning result for MAC^0 to circuits which contain several majority gates:

[3] *The analysis of the running time dependence on the confidence parameter δ is standard and omitted for clarity.*

Theorem 5.3.7 *Let f be computed by a circuit of size s and depth d which contains m majority gates. Then f is computed by a circuit of size $2^{m(O(\log s))^{2d+1}}$ and depth $d+2$ which contains a single majority gate at the root.*

Combining Theorems 5.3.6 and 5.3.7 we obtain our main learning result:

Theorem 5.3.8 *The class of size-s depth-d circuits which contain m majority gates can be learned to accuracy ϵ under the uniform distribution in time $n^{O((m(\log s)^{2d+1}+\log(1/\epsilon))^{d+1})}$.*

The most interesting setting of parameters in Theorem 5.3.8 is $s = 2^{polylog\, n}$, $d = O(1)$ and $m = \text{polylog}(n)$ which gives Theorem 5.1.1 as stated in the introduction.

5.3.4 Discussion

Theorem 5.3.6 can be used to give a slight improvement on results of Linial *et al.* for learning and approximating constant-depth circuits. Since a majority gate can easily simulate an AND or OR gate by padding it with extra inputs, for any size-s depth-d AND/OR/NOT circuit C it is possible to reexpress C as an MAC^0 circuit $MAJ(C_1, \ldots, C_t)$ in which each C_i has size at most s and depth at most $d-1$. Applying Theorem 5.3.6 we see that any size-s depth-d AND/OR/NOT circuit can be learned using our algorithm in time $n^{O(\log^{d-1}(s/\epsilon))}$. This is in contrast with the Linial *et al.* algorithm which runs in time $n^{O(\log^{d}(s/\epsilon))}$.

By viewing AND/OR/NOT circuits as MAC^0 circuits in this way, our algorithm also gives the best known upper bound on the degree of polynomial threshold functions which approximate constant-depth circuits. A polynomial threshold function is a Boolean function defined by a real-valued multivariate polynomial $P(x)$. The Boolean function takes value 1 on x if and only if $P(x) \geqslant 0$.

As mentioned in Section 5.3.2, the final hypothesis of our algorithm is a majority of parities of at most k variables each. Since any Boolean function on k variables can be computed by a polynomial of degree at most k, we have the following corollary:

Corollary 5.3.9 *Let f be computed by a circuit of size s and depth d. Then there is a polynomial threshold function of degree $O(\log^{d-1}(s/\epsilon))$ which agrees with f on all but an ϵ fraction of inputs.*

The Linial *et al.* results imply the existence of a polynomial threshold function approximator of degree $O(\log^d(s/\epsilon))$.

Aspnes *et al.* [5] have given tight bounds on the accuracy with which polynomial threshold functions of a given degree can approximate the parity function. Using their results together with the fact that a depth-d circuit of size s can compute parity on $\log^{d-1} s$ variables, one can show

Observation 5.3.10 *There is a circuit C of size s and depth d with the following property: any polynomial threshold function of degree $O(\log^{d/2} s)$ must disagree with C on more than an $\epsilon = 1/s$ fraction of inputs.*

Thus the $d-1$ exponent in our upper bound of Corollary 5.3.9 is worse than the best possible $(d/2)$ exponent by less than a factor of two.

Finally, we observe that as a corollary of Theorem 5.3.8 we obtain a quasipolynomial time algorithm for learning an intersection of polylog(n) halfspaces under the uniform distribution on the Boolean cube, under the restriction that each halfspace is defined by integer coefficients of magnitude poly(n). This is in contrast with a polynomial time algorithm by Baum [9] for learning an intersection of two origin-centered halfspaces under a restricted class of distributions which includes the uniform distribution. (Vempala [112] gave a polynomial-time algorithm to learn an intersection of $\log n / \log\log n$ halfspaces under "near-uniform" distributions on the unit ball, but his algorithm does not apply to the uniform distribution on the Boolean cube.)

5.4 Cryptographic Lower Bounds

In contrast with the positive learning results of Section 5.3, in this section we give cryptographic hardness results for learning circuits with majority gates. Our results, which are based on the presumed intractability of factoring Blum integers, suggest that the time bounds of Theorem 5.3.8 are essentially optimal even for weak learning algorithms which are allowed to make membership queries.

Definition 5.4.1 *Let $N = P \cdot Q$ be chosen by uniformly selecting P and Q to be two $n/2$-bit primes such that $P \equiv Q \equiv 3 \bmod 4$ (we call such an N a* random n-bit Blum integer*). Let $\epsilon(n)$ and $t(n)$ be real-valued functions. We say that a probabilistic $t(n)$-time algorithm A* ϵ-factors *if*

$$\Pr[A(N) \in \{P, Q\}] > \epsilon(n)$$

where $N = P \cdot Q$ is a random n-bit Blum integer. The $(t(n), \epsilon(n))$-factoring assumption *states that there is no probabilistic poly$(t(n))$-time algorithm which poly$(\epsilon(n))$-factors.*

After decades of intensive research on integer factorization, the fastest factoring algorithms known for n-bit Blum integers run in time $2^{\tilde{O}(n^{1/3})}$. Thus the following assumption is viewed as being quite plausible [64]: there is some absolute constant $\alpha > 0$ such that the $(2^{n^\alpha}, 2^{-n^\alpha})$-factoring assumption holds.

Definition 5.4.2 *Let $I(n)$ be an index set and let $\{f_i : \{-1,1\}^n \to \{-1,1\}\}_{i \in I(n)}$ be a family of functions. We say that $\{f_i\}_{i \in I(n)}$ is an* $(t(n), \epsilon(n))$-secure pseudorandom function family *if the following conditions hold:*

- *there is a probabilistic poly(n)-time algorithm which selects a uniform random $i \in I(n)$;*
- *there is a deterministic poly(n)-time algorithm which given $i \in I(n)$ and $x \in \{-1,1\}^n$ outputs $f_i(x)$;*
- *for any probabilistic $t(n)$-time oracle algorithm A,*

$$\left| \Pr_{i \in I(n)}[A^{f_i} \text{ outputs } 1] - \Pr_{F \in \mathcal{F}_n}[A^F \text{ outputs } 1] \right| < \epsilon(n)$$

  where $\mathcal{F}_n$ is the family of all Boolean functions $F : \{0,1\}^n \to \{0,1\}$.

The following theorem follows from [90]:

Theorem 5.4.3 *There is an index set $I(n)$, a function family $\{f_i : \{0,1\}^n \rightarrow \{-1,1\}\}_{i \in I(n)}$, and a fixed constant $\beta > 0$ such that*

- *Each function $f_i : \{-1,1\}^n \rightarrow \{-1,1\}$ is computed by a depth-5 circuit consisting of n^β majority gates;*
- *If $\{f_i\}_{i \in I(n)}$ is not a $(t(n), \epsilon(n))$-secure pseudorandom function family, then there is a poly$(t(n)/\epsilon(n))$-time randomized algorithm which poly$(\epsilon(n)/t(n))$-factors.*

The following theorem about hardness of learning follows:

Theorem 5.4.4 *Suppose that there is a membership query learning algorithm A for the class of functions $\{f_i : \{-1,1\}^n \rightarrow \{-1,1\}\}_{i \in I(n)}$ of Theorem 5.4.3 which runs in time $t(n)$ and outputs a hypothesis h such that $\Pr[h(x) \neq f(x)] \leqslant \frac{1}{2} - \epsilon(n)$. Then the $(t'(n), \epsilon'(n))$-factoring assumption is false for some $t'(n) =$poly$(t(n)/\epsilon(n))$ and $\epsilon'(n) =$poly$(\epsilon(n)/t(n))$.*

We can apply Theorem 5.4.4 to obtain hardness results for learning constant-depth circuits with majority gates:

Theorem 5.4.5 *Suppose that the $(2^{n^\alpha}, 2^{-n^\alpha})$-factoring assumption holds for some fixed $\alpha > 0$. Then any membership query algorithm that weakly learns depth 5 circuits consisting of m majority gates to accuracy $\epsilon = 1/2 - 2^{-m^{\alpha/\beta}}$ must have time complexity $\omega(2^{m^{\alpha/\beta}})$, where β is the constant from Theorem 5.4.3.*

Proof: From Theorem 5.4.3 we know that every function $f_{i,m^{1/\beta}}$ can be computed by a depth 5 circuit consisting of m majority gates. If A runs in time $2^{m^{\alpha/\beta}}$ and weakly learns $\{f_{i,m^{1/\beta}}\}$ to accuracy $1/2 - 2^{-m^{\alpha/\beta}}$, then A can weakly learn $\{f_{i,n}\}$ to accuracy $1/2 - 2^{-n^\alpha}$ in time 2^{n^α}. The result follows by Theorem 5.4.4. ■

As a corollary we have that our algorithm for learning constant-depth circuits with polylogarithmically many majority gates is essentially optimal in terms of its time complexity and the number of majority gates which it can handle.

Corollary 5.4.6 *Fix any constants $d \geqslant 5$ and $\epsilon > 0$.*

1. *Our learning algorithm from Section 5.3 learns depth-d poly*(n)*-size circuits with* $\log^k n$ *majority gates to accuracy* ϵ *in time* $2^{(\log n)^{O(k)}}$.

2. *Assuming that the* $(2^{n^\alpha}, 2^{-n^\alpha})$*-factoring assumption holds for some fixed* $\alpha > 0$*, any algorithm (even using membership queries) which learns depth-d circuits consisting of* $\log^k n$ *majority gates (even to accuracy* $1/2 - 2^{-(\log n)^{\Theta(k)}}$*) must run in time* $2^{(\log n)^{\Omega(k)}}$.

3. *Assuming that the* $(2^{n^\alpha}, 2^{-n^\alpha})$*-factoring assumption holds for some fixed* $\alpha > 0$*, then for any* $m = \omega($*polylog*$(n))$ *there is no quasipolynomial-time algorithm (even using membership queries) which learns depth-d circuits consisting of* m *majority gates (even to accuracy* $1/2 - 2^{-m^{\Theta(1)}}$*).*

5.5 Learning other extensions of AC^0

Given our results for learning AC^0 augmented with majority gates, a natural question is whether AC^0 circuits augmented with other expressive gates can be similarly learned. A first step in this direction is to investigate the learnability of constant-depth polynomial-size circuits with a single parity gate, MOD_p gate, or threshold gate at the root. We refer to these three circuit classes as *parity of* AC^0, MOD_p *of* AC^0 and *threshold of* AC^0. While we have not been able to prove cryptographic hardness results for these three classes, as described below we believe that substantially new techniques will be required to obtain learning algorithms for these classes.

The most powerful algorithms known for learning Boolean circuits under the uniform distribution are based on identifying large Fourier coefficients. Indeed, the only classes for which uniform distribution quasipolynomial time learning algorithms are known are classes such that for each $f \in C$ there is some Fourier coefficient such that $|\hat{f}_A| \geqslant 1/n^{\mathrm{polylog}(n)}$. (Our analysis in Section 5.3 implies the existence of such a Fourier coefficient for any constant-depth circuit with polylog(n) majority gates.) For parity of AC^0 it can be shown that all Fourier coefficients may be exponentially small. This is almost true of MOD_p of AC^0 as well: the only exception is the constant coefficient, and we argue that existing methods are not powerful enough to produce anything but weak learning of such functions.[4] Thus it appears that Fourier techniques will not suffice to learn parity of AC^0 or MOD_p of AC^0.

For threshold of AC^0, we give evidence that our algorithm from Section 5.3 will not work by constructing a threshold of AC^0 function with the following property: under the uniform distribution on $\{-1,1\}^n$ each input to the threshold gate has only an exponentially small correlation with the output of the circuit. (Recall that our algorithm's proof of correctness relies on the fact that for a majority of AC^0 circuit there is always some input to the majority gate which has nonnegligible correlation with the output.)

5.6 Bounds on the Fourier coefficients of Extensions of AC^0

5.6.1 Parity of AC^0

The *inner product mod 2* function $IP_2 : \{0,1\}^{2n} \rightarrow \{0,1\}$ is defined by $IP_2(x) = x_1x_2 + x_3x_4 + \ldots + x_{2n-1}x_{2n}$ mod 2. This function is computed by a depth 2 circuit with n AND gates on the bottom level which feed into a single parity gate at the root. It is known [27, 87] that every Fourier coefficient of this function satisfies $|\widehat{IP}_2(S)| = 1/2^n$.

We note that the IP_2 function can also be used to show (without any cryptographic assumptions) that our learning algorithm from Section 5.3 will not run in quasipolynomial time on depth-4 circuits with more than a polylogarithmic number of majority gates. Using majority gates to compute parity, the IP_2 function on m variables can be computed by a depth-4 size-m^2 circuit which contains m majority gates. Let $f_m : \{0,1\}^n \rightarrow \{0,1\}$ denote the inner product mod 2 function on the first

[4] Recall that under the uniform distribution weak learning is not always equivalent to strong learning, e.g. the class of monotone Boolean functions is weakly but not strongly learnable under uniform [66].

m variables $x_1, \ldots, x_m$. For $m = \omega(\text{polylog}(n))$, the first execution of our weak learning algorithm (under the uniform distribution) will take more than quasipolynomial time to identify a parity function which is weakly correlated with the target function.

5.6.2 MOD_p of AC^0 for $p > 2$

For $x \in \{0,1\}^n$ the function $MOD_p(x)$ outputs 1 if and only if $x_1 + \ldots + x_n$ is an integer multiple of p. It has been shown by Zhang [117] that for any fixed prime $p > 2$ the function $MOD_p(x)$ has $|\widehat{MOD}_p(S)| = 1/2^{\Omega(n)}$ for all $S \neq \varnothing$. This observation taken by itself leaves open the possibility that boosting and Fourier methods could be combined to learn this class as follows: choose the constant function $-\chi_\varnothing$ as the first weak hypothesis. Then apply boosting to shift the distribution. Perhaps some other parity function will now be a weak approximator with respect to this new distribution and can be found reasonably efficiently. If so, then potentially a strong hypothesis can be produced by continuing this boosting process.

However, we argue that this approach cannot succeed given current boosting algorithms. Every known boosting algorithm that is given a constant function as the first weak hypothesis will form the distribution for the next boosting round in a very simple way: it gives more weight to all positive examples and less weight to all negative examples, or vice versa. More precisely, it will change the weight on all positive examples by the same additive or multiplicative factor, and it will similarly change the weight on all negative examples by using a single factor. If the initial distribution has at most two distinct weights, one shared by all positive examples and the other shared by all negative examples, then the new distribution will have the same form—all positives share a single weight, all negatives a different weight—but will have weights that differ from the first distribution.

Also, while the specific weight changes may vary with the boosting algorithm, in no case will the change be greater than is required to balance the weight on the positive and negative examples. Specifically, in the case of learning MOD_p with respect to the uniform distribution, the most weight put on examples in the second boosting distribution $\mathcal{D}_1$ will be $(p/2)/2^n$.

We will show that after any such change to the distribution it will still be the case that only the constant parity function has more than inverse exponential correlation with the target MOD_p function with respect to the uniform distribution.

So consider the correlation of any parity function χ_B, $B \neq \varnothing$, with the target MOD_p function f with respect to $\mathcal{D}_1$. Let $w_{-1}/2^n$ be the weight placed on each negative example by $\mathcal{D}_1$ and $w_1/2^n$ be the weight on each positive example. Then

$$E_{\mathcal{D}_1}[f\chi_B] = \frac{w_{-1}}{2^n} \sum_{x.f(x)=-1} -\chi_B(x) + \frac{w_1}{2^n} \sum_{x.f(x)=1} \chi_B(x).$$

Now let $f'(x) = (f(x)+1)/2$ (f' is the 0/1-valued function that is 1 if and only if f is 1). Then for all $A \neq \varnothing$, $\widehat{f'}(A) = \hat{f}(A)/2$. Similarly, for $f''(x) = (1 - f(x))/2$ (1 if and only if f is -1), $\widehat{f''}(A) = -\hat{f}(A)/2$ for all $A \neq \varnothing$. So

$$\begin{aligned} E_{\mathcal{D}_1}[f\chi_B] &= w_1 E_{\mathcal{U}}[f'(x)\chi_B(x)] - w_{-1}E_{\mathcal{U}}[f''(x)\chi_B(x)] \\ &= \frac{w_{-1}+w_1}{2}\hat{f}(B). \end{aligned}$$

Since the w's are at most linear in p, and since $\hat{f}(B)$ is exponentially small, we have that every parity function (except possibly the constant) is very weakly correlated with f with respect to $\mathcal{D}_1$. This implies that at the next boosting step, any known Fourier-based learning strategy can do no better than choosing the constant as the weak hypothesis. But this will simply lead to a distribution $\mathcal{D}_2$ of the same form as $\mathcal{D}_1$ and with the same constraints on the weights.

5.7 Learning Threshold of AC^0

A threshold function $t : \{-1,1\}^n \to \{-1,1\}$ is defined by real numbers $w_1, \ldots, w_n, \theta$; the value of $t(x)$ is 1 if and only if $w_1x_1+\cdots+w_nx_n \geqslant \theta$. We do not know whether every threshold of AC^0 function must have some Fourier coefficient of inverse quasipolynomial size. Thus it is conceivable that our algorithm in Section 5.3 may be able to learn any threshold of AC^0 function in quasipolynomial time. However, we can show that our analysis given in Section 5.3.1 breaks down for threshold of AC^0. More specifically we construct a threshold of AC^0 function such that even under the uniform distribution on $\{-1,1\}^n$ no input to the threshold gate is nonnegligibly correlated with its output. We use the following lemma:

Lemma 5.7.1 *Let $n \geqslant 1$ be odd and let $t : \{-1,1\}^n \to \{-1,1\}$ be the threshold function defined by $t(x) \stackrel{def}{=} (-1)^{(j+1)}$ where $1 \leqslant j \leqslant n$ is the smallest index such that $x_j = 1$ (if $x_j = -1$ for all j then $t(x) = -1$). For $1 \leqslant i \leqslant n+1$ and $1 \leqslant j \leqslant n$ let $x^i \in \{0,1\}^n$ be defined as follows:*

$$x^i_j = \begin{cases} -1 & \text{if } i > j \\ 1 & \text{if } i = j \\ (-1)^{i+j+1} & \text{if } i < j. \end{cases}$$

Let F_ℓ denote the ℓ-th Fibonacci number (so $F_1 = F_2 = 1, F_3 = 2$, etc.) and let $G_\ell \stackrel{def}{=} F_{\ell+2} + F_{\ell-1} - 1$. Let $\mathcal{D}$ be the following distribution over $\{-1,1\}^n$:

$$\mathcal{D}(x) = \begin{cases} F_i/G_n & \text{if } x = x^i \text{ for some } 1 \leqslant i \leqslant n \\ F_{n-1}/G_n & \text{if } x = x^{n+1} \\ 0 & \text{otherwise.} \end{cases}$$

Then for each $i = 1, \ldots, n$ we have $|E_{\mathcal{D}}[x_i t(x)]| = 1/G_n = 1/2^{\Omega(n)}$ and moreover $E_{\mathcal{D}}[t(x)] = 1/G_n = 1/2^{\Omega(n)}$.

As the first step of the analysis in Section 5.3.1, we observed that for any distribution $\mathcal{D}$ and any MAC^0 circuit $f = MAJ(C_1, \ldots, C_t)$ there is some AC^0 circuit C_i which is an input to the top-level majority gate such that $|E_{\mathcal{D}}[C_i f]| \geqslant 1/t$. We now show that this property does not hold for threshold of AC^0 even if the distribution is uniform. More precisely, we have

Lemma 5.7.2 *There is a threshold of AC^0 function $g : \{-1,1\}^n \to \{-1,1\}$, defined by $g(x) = t(C_1, \ldots, C_n)$ where each C_i is a polynomial-size depth-2 circuit, such that for $1 \leqslant i \leqslant n$ we have $|E[C_i g]| = O(1/G_n)$ and moreover $E[g] = O(1/G_n)$.*

Proof: We exhibit n constant-depth circuits $C_1, \ldots, C_n$ which have the following property: given a uniformly distributed input $x \in \{-1,1\}^n$, the distribution of the n-bit string $C_1(x)C_2(x)\ldots C_n(x)$ which is input to t is exponentially close to the distribution $\mathcal{D}$ described in Lemma 5.7.1. This is done as follows: let $[0, a_1], [a_1 + 1, a_2], \ldots, [a_{n-1}+1, a_n], [a_n + 1, 2^n - 1]$ be a partition of $\{0, 1, \ldots, 2^n - 1\}$ into $n + 1$ disjoint intervals where the i-th interval $[a_{i-1}+1, a_i]$ is of length approximately $2^n \mathcal{D}(x^i)$. On input $x \in \{-1,1\}^n$ the j-th circuit C_j outputs 1 if and only if the number in $\{0, 1, \ldots, 2^n - 1\}$ corresponding to x lies in an interval $[a_{k-1}+1, a_k]$ such that the i-th bit of x^k is 1.

It is easily verified that the above choice of $a_1, \ldots, a_n$ causes the distribution of $C_1(x)C_2(x)\ldots C_n(x)$ to be exponentially close to $\mathcal{D}$. Each circuit C_i as described above must output 1 if and only if its input, viewed as an integer, lies in a union of at most n intervals in $\{0, 1, \ldots, 2^n - 1\}$. Such circuits can be constructed in polynomial size and depth 2. ■

We conclude by observing that known polynomial-time algorithms for learning a single threshold gate make essential use of geometric techniques (e.g. the linear programming algorithms used by Blumer *et al.* [25] and the "outlier removal lemma" of Blum *et al.* [21]). These techniques have a very different feel from the Fourier methods which have proved useful for uniform distribution learning. It would be interesting to have a learning algorithm which combines these two approaches.

Chapter 6

New Algorithms for Learning Intersections of Halfspaces

6.1 Introduction

Let f be a hyperplane in $\mathbb{R}^n$, i.e., $f(x) = \sum_{i=1}^{n} a_i x_i + \theta$ where a_i's and θ are integers. This hyperplane induces two halfspaces and a natural classification of points: the set of points x such that $f(x) \geqslant 0$ are labelled positive and the set of points y such that $f(y) < 0$ are labelled negative. The problem of learning an unknown halfspace, also known as learning a linear threshold function, has been studied for over 40 years, dating back to the work of Block [17], Novikoff [93], and Rosenblatt[98] and continues to be an active area of research. The first algorithm for PAC learning a single halfspace uses linear programming and appears in work due to Blumer et al. [25] from 1989 (in Chapter 3 we saw how to apply this result to learn DNF formulae in time $2^{\tilde{O}(n^{1/3})}$).

While the problem of learning a single halfspace is well understood, the problem of learning an intersection of several halfspaces remains one of the central challenges in computational learning theory. Researchers have been interested in intersections of halfspaces because they form a powerful concept class. Any convex body, for example, can be expressed as an intersection of halfspaces, and several well studied concept classes, such as DNF formulae, can be viewed as special cases. Several researchers have made progress on solving restricted variants of this problem ([23, 112, 20, 78]) but very little is known with respect to the original problem of PAC learning an intersection of halfspaces from random examples only. The best result to date is due to Baum [9] who in 1989 gave an algorithm to learn the intersection of two origin-centered halfspaces with respect to any symmetric distribution.

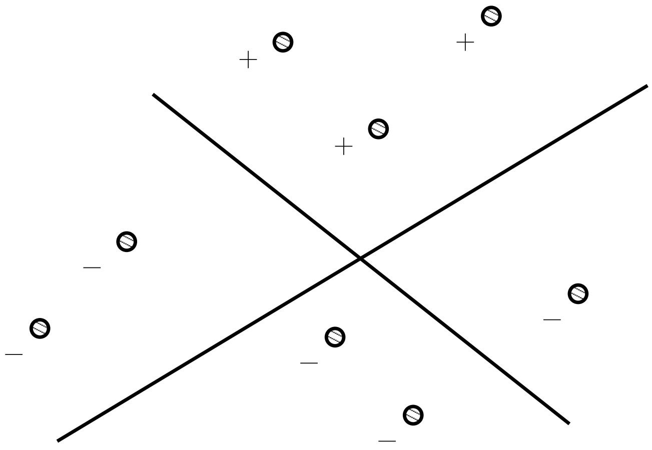

Figure 6-1: The intersection of two halfspaces

6.2 Our Results

In this chapter we give several new results for learning intersections, arbitrary functions, and thresholds of halfspaces from random examples only. We present the first polynomial time algorithm for learning the intersection of any constant number of halfspaces under the uniform distribution on the Boolean hypercube ($\{0,1\}^n$) to within any constant error parameter. More precisely we have

Theorem 6.2.1 *An arbitrary Boolean function of k halfspaces is learnable with respect to the uniform distribution to accuracy ϵ in time $n^{O(k^2/\epsilon^2)}$.*

We also give new algorithms for learning intersections of halfspaces with respect to an arbitrary distribution. First we show that the intersection of k halfspaces where the sum of coefficients of each halfspace is bounded by w can be learned in time

$n^{O(k \log k \log w)}$. This gives the first quasipolynomial time algorithm for learning the intersection of a constant number of halfspaces with polynomially bounded weights under any distribution. In fact we can learn any function of a constant number of halfspaces in quasipolynomial time:

Theorem 6.2.2 *An arbitrary Boolean function of k halfspaces where the sum of the coefficients of each halfspace is bounded by w is learnable with respect to any distribution to accuracy ϵ in time $n^{O(k^2 \log w)}$ (and time polynomial in ϵ^{-1}).*

Secondly we show a related result stating that the intersection of k halfspaces where the sum of coefficients of each halfspace is bounded by w can be learned in time $n^{O(w^{1/2} \log k)}$. This gives the first subexponential time algorithm for learning the intersection of a polynomial number of halfspaces whose coefficients are strongly bounded:

Theorem 6.2.3 *The intersection of k halfspaces where the sum of the coefficients of each halfspace is bounded by w is learnable with respect to any distribution to accuracy ϵ in time $n^{O(w^{1/2} \log k)}$ (and time polynomial in ϵ^{-1}).*

Since the first two theorems apply to any function of halfspaces we obtain the first polynomial time algorithm for learning neural networks under the uniform distribution in which there are a constant number of bottom level gates that the inputs feed into (again to within any constant error parameter):

Theorem 6.2.4 *Any neural network with k bottom level gates and n inputs can be learned to accuracy ϵ in time $n^{O(k^2/\epsilon^2)}$.*

6.3 Previous Work

The most significant progress on the problem of learning intersections of halfspaces from random hypercube examples only is due to Baum [9] who gave a polynomial time algorithm for learning an intersection of two origin-centered halfspaces under any "symmetric" distribution (which satisfies $\mathcal{D}(x) = \mathcal{D}(-x)$ for all $x \in \mathbb{R}^n$) by reducing the problem to learning the XOR of two halfspaces.

Several researchers have considered a different variant of this problem by allowing the learning algorithm to make membership queries (i.e. to query the unknown intersection of halfspaces at points of its choosing). In particular, Blum et. al. [20] have given a membership-query algorithm for learning the intersection of two halfspaces (with respect to any distribution on the hypercube) which will work even if the queries are answered incorrectly near the boundary of the unknown concept. Kwek and Pitt [78] have given a membership-query algorithm for learning the intersection of k halfspaces in n dimensions (with respect to any distribution on the hypercube) in time polynomial in n and k (their work also contains additional information on the history of the problem).

Progress has also been made on the problem of learning intersections of halfspaces when the examples are chosen according to particular distributions on the n-dimensional Euclidean ball. Building on work due to Blum and Kannan [23], Vempala [112] gave a polynomial-time algorithm which can learn an intersection of $\log n / \log \log n$ halfspaces with respect to "near-uniform" distributions on the Euclidean ball. His approach uses the technique of random sampling to hone in on the subspace spanned by the normal vectors of the unknown halfspaces.

6.4 Our Approach

We use complexity theoretic tools to give new representations of intersections of halfspaces. Our approach is twofold: (1) we give a Fourier based algorithm for learning with respect to the uniform distribution and (2) we represent intersections of halfspaces as polynomial threshold functions in order to learn with respect to an arbitrary distribution.

6.4.1 Fourier Analysis and Noise Stability

To describe our Fourier based algorithm we reconsider the classical problem of learning a single linear halfspace. As mentioned earlier, several algorithms are known for this problem including the Perceptron algorithm [1, 98, 47], Winnow [83], Boosting based methods [103], and Linear Programming [25]. Although these algorithms differ in various aspects, they all use geometry to deduce where the unknown halfspace lies.

We give an alternate, non-geometric algorithm for learning a single linear halfspace: we estimate the Fourier spectrum of the underlying linear threshold function. We use the fact that a $1-\epsilon$ fraction of the Fourier spectrum of a linear threshold function resides on basis vectors of degree $O(1/\epsilon^2)$. As a result, we obtain an $n^{O(1/\epsilon^2)}$ time algorithm for learning a halfspace to within accuracy ϵ by applying the "low degree algorithm" of Linial *et al.* [81] which estimates the Fourier coefficients of all $O(1/\epsilon^2)$-degree basis vectors.

One obvious drawback of this algorithm is its bad dependency on ϵ. On the other hand, this algorithm, unlike all previous geometric ones, generalizes easily to an arbitrary function of halfspaces. As we will see, we can almost immediately obtain an $n^{O(k^2/\epsilon^2)}$ time algorithm for learning an arbitrary function of k halfspaces.

We prove the above Fourier concentration bounds using a technique called *noise-stability*. A function is noise-stable if, for a randomly chosen input, a random perturbation of some small subset of the input bits does not, with high probability, change the output of the function. Functions that are highly noise-stable have well understood Fourier concentration properties. We will show using work work due to Benjamini *et al.* [15] and Y. Peres that a linear threshold function has good noise-stability properties and is thus amenable to this type of Fourier analysis.

Many researchers have previously used a Fourier approach to develop powerful algorithms for concept classes including AC^0 [81] and DNF formulae [61]. While the idea of learning via a Fourier estimation has been used before, the application to learning intersections of halfspaces is new.

Analyzing the noise-stability of boolean functions has previously appeared as a complexity theoretic tool for understanding percolation [15], Boolean circuits, [15] and hardness amplification [94]. We believe this to be the first application of these techniques to machine learning.

6.4.2 The Polynomial Threshold Function Approach

We give new learning algorithms for learning with respect to an arbitrary distribution by showing how to represent intersections of halfspaces as low degree polynomial threshold functions. Recall that in chapter 3 we showed how to represent a polynomial size DNF formula as a polynomial threshold function of degree $\tilde{O}(n^{1/3})$ to obtain a $2^{\tilde{O}(n^{1/3})}$ time algorithm for learning DNF under any distribution. Here we show that any function of k halfspaces where the sum of the coefficients of each halfspace is bounded by w can be represented as polynomial threshold fuction of degree $O(k^2 \log w)$. The idea is to apply the rational function approximation tools from Beigel, Reingold, and Spielman [14] (a well known result in complexity) to force the output values of a linear threshold function close to boolean ones. Having done this we can easily obtain a single polynomial threshold function computing an arbitrary function of halfspaces by using addition and multiplication to mimic the logical AND and OR operations. Our dependence on w comes directly from the degree of the rational functions constructed in [14].

We can give an alternative polynomial threshold construction useful for representing the intersection of many halfspaces where each halfspace has small coefficients. By generalizing the main theorem from Chapter 3, we obtain a polynomial threshold function of degree $O(w^{1/2} \log k)$ computing the intersection of k halfspaces where the sum of the coefficients of each linear threshold function is at most w.

6.5 Preliminaries

In this chapter we work within Valiant's *Probably Approximately Correct* (PAC) model [111] of learning. In this model a *concept class* C is a collection of Boolean functions $c : \{0,1\}^n \to \{-1,1\}$.

In the PAC model, for Boolean functions c, h on $\{0,1\}^n$ and $\mathcal{D}$ a distribution on $\{0,1\}^n$, we say that h is an *ϵ-approximator for c under $\mathcal{D}$* if $\Pr_{\mathcal{D}}[c(x) = h(x)] \geqslant 1-\epsilon$. The learning algorithm has access to an *example oracle $EX(c, \mathcal{D})$* which, when queried, provides a labeled example $\langle x, c(x)\rangle$ where x is drawn from $\{0,1\}^n$ according to the distribution $\mathcal{D}$ and $c \in C$ is the unknown target concept which the algorithm is trying to learn. The goal of the learner is to generate an ϵ-approximator for c under $\mathcal{D}$. An algorithm A is a *PAC learning algorithm for a concept class C* if the following condition holds: for any $c \in C$, any distribution $\mathcal{D}$ on $\{0,1\}^n$, and any $0 < \epsilon, \delta < 1$, if A is given ϵ and δ and has access to $EX(c, \mathcal{D})$, then with probability at least $1-\delta$ algorithm A outputs an ϵ-approximator for c under $\mathcal{D}$. We say that *A PAC learns C in time t* if A runs for at most t time steps and outputs a hypothesis h which can be evaluated on any point $x \in \{0,1\}^n$ in time t.

6.5.1 Fourier Analysis

We briefly review some facts from Fourier analysis that can also be found in Chapter 5:

For $A \subseteq [n]$ the parity function $\chi_A : \{-1,1\}^n \to \{-1,1\}$ is defined by $\chi_A(x) = \prod_{i \in A} x_i$. As is well known, with inner product $\langle f, g\rangle = E[fg]$ and norm $\|f\| = \sqrt{E[f^2]}$ the 2^n parity functions $\{\chi_A\}_{A \subseteq [n]}$ form an orthonormal basis for the vector space of real-valued functions on $\{-1,1\}^n$. Consequently every real-valued function $f : \{-1,1\}^n \to \{-1,1\}$ can be uniquely expressed as a linear combination $f(x) = \sum_{A \subseteq [n]} \hat{f}(A)\chi_A(x)$.

The coefficients $\hat{f}(A)$ are known as the *Fourier spectrum* of f. Orthonormality implies that $\hat{f}(A) = \langle f, \chi_A\rangle$ and thus $\hat{f}(A)$ measures the correlation between f and χ_A. Orthonormality also implies *Parseval's identity*, which states that for every real-valued function f we have $E[f^2] = \sum_{A \subseteq [n]} \hat{f}(A)^2$. For Boolean functions we thus have $\sum_{A \subseteq [n]} \hat{f}(A)^2 = 1$.

The following easily verified fact is from [81]:

Fact 6.5.1 *For any Boolean function f and real-valued function g,*

$$\begin{aligned}\Pr[f(x) \neq sign(g(x))] &\leqslant E[(f(x) - g(x))^2] \\ &= \sum_{A \subseteq [n]} (\hat{f}(A) - \hat{g}(A))^2.\end{aligned}$$

The equality follows from Parseval's identity and the linearity of the Fourier transform.

We define $\mathsf{S}_t(f) = \sum_{A \subset [n], |A| \leqslant t} \hat{f}(A)^2$ and often refer to $\mathsf{S}_t(f)$ as the Fourier weight of f concentrated on basis vectors of degree (or length) up to t. Sometimes we abuse notation and write $\mathsf{S}_{O(t)}(f) = \delta$ to mean that there exists a sufficiently large constant c such that $\mathsf{S}_{ct}(f) = \delta$.

If a Boolean function has a $1-\epsilon$ fraction of its Fourier weight concentrated on basis vectors of degree less than t then there exists a simple n^t algorithm for computing an approximation to f. The algorithm takes several labelled random examples and approximates the Fourier coefficient of each basis vector of degree t or less by tallying the correlation of each basis vector with the labelled examples. This fact is first stated and proved in [81]; we reference a nice presentation by Mansour [85]:

Fact 6.5.2 *Let f be a Boolean function such that $\mathsf{S}_t(f) \geqslant 1 - \epsilon$. Then there exists an algorithm running in time $O(n^t)$ which, with probability $1 - \delta$, outputs a threshold of parity functions g such that $\mathsf{E}[(f - g)^2] \leqslant \epsilon$.*

Thus, for a Boolean f, if $\mathsf{S}_t(f) \geqslant 1 - \epsilon$, then f is learnable to within accuracy ϵ under the uniform distribution in time $O(n^t)$.

For a Boolean predicate P dependent upon some random experiment we let $\mathbf{1}_P$ denote the indicator variable which takes on the value 1 if P is true and 0 otherwise.

6.5.2 Polynomial Threshold Functions

Definition 6.5.3 *A* polynomial threshold function *is defined by a real-valued multivariate polynomial $p(x_1, \ldots, x_n)$. The output of the polynomial threshold function on input $x \in \{-1, 1\}^n$ is 1 if $p(x_1, \ldots, x_n) \geqslant 0$ and is -1 otherwise. The* degree *of* *a polynomial threshold function is simply the degree of the polynomial p. A linear threshold function is a polynomial threshold function of degree one.*

Definition 6.5.4 *Let* $h : \{-1,1\}^n \to \{-1,1\}$ *be a linear threshold function on the Boolean cube. The* weight *of h is the smallest integer W for which there exist* $w_1, \ldots, w_n, \theta \in \mathbb{N}$ *with* $|\theta| + \sum_{i=1}^{n} |w_i| = w$ *such that* $h(x) = sgn(w \cdot x - \theta)$ *for all* $x \in \{-1,1\}^n$.

We restate two facts from Chapter 3 that will be useful here:

Fact 6.5.5 ([25, 86]) *In both the PAC model and the model of exact learning from equivalence queries, there are algorithms which learn the class of linear threshold functions over* $\{0,1\}^n$ *in time poly*(n).

The algorithms of Fact 6.5.5 are based on polynomial time linear programming. We will need the following extension of Fact 6.5.5:

Fact 6.5.6 *Let C be a class of functions each of which can be expressed as an degree-d polynomial threshold function over* $\{0,1\}^n$*. Then in both the PAC learning model and the model of exact learning from equivalence queries, there is a learning algorithm for C which runs in time* $n^{O(d)}$.

6.6 Learning Functions of Halfspaces Under the Uniform Distribution

The actual algorithm for learning an intersection of halfspaces under the uniform distribution is simple: to learn to within accuracy ϵ, estimate the Fourier coefficients of basis vectors of length $O(1/\epsilon^2)$. What is more complicated is the analysis, which can be broken down into two parts: (1) a proof that any function of k halfspaces has noise stability $O(k\sqrt{\epsilon})$ and (2) a proof that noise stability $O(k\sqrt{\epsilon})$ implies a Fourier concentration that requires us to estimate the coefficients of basis vectors of degree only $O(k^2/\epsilon^2)$.

6.6.1 Noise Stability of a Linear Threshold Function

We begin with the definition of the noise stability of a Boolean function:

Definition 6.6.1 *Let $f : \{0,1\}^n \to \{-1,1\}$ be a Boolean function. Let $x \in \{0,1\}^n$. Flip each bit of x with probability ϵ and let $N_\epsilon(x)$ be the resulting string. The* noise stability *of f with respect to ϵ, denoted $\mathsf{NoiseStab}_\epsilon(f)$, is equal to*

$$Pr_x[f(x) \neq f(N_\epsilon(x))]$$

where the probability is taken over both the choice of string x and subsequent perturbation (each bit of x is flipped with probability ϵ).

In other words, the noise stability of f is the probability that the value of f on a random input is different than the value of f on that input after each bit is flipped with probability ϵ. As an easy example the constant function is highly noise stable, and the parity function is not noise stable (flipping any bit alters its output). In work by Benjamini, Kalai, and Schramm [15], the authors prove that a linear threshold function (such as majority) has noise-stability $O(\epsilon^{1/4})$. What we will use here is a result due to Y. Peres [96] showing that the noise stability of any linear threshold function is actually $O(\sqrt{\epsilon})$. The proof we present here was communicated to us by Y. Peres [96]:

Theorem 6.6.2 *[96] Let f be a linear threshold function with weights $w_1, \ldots, w_n$ and threshold θ. Let $x = x_1, \ldots, x_n$ be a randomly chosen input. Then*

$$Pr[f(x) \neq f(N_\epsilon(x))] \leqslant O(\sqrt{\epsilon})$$

We will first compute, in terms of k, the probability that flipping k bits of x changes the output of $f(x)$:

Lemma 6.6.3 *Let $f : \{0,1\}^n \to \{-1,1\}$ be a linear threshold function with threshold $\theta = 0$. Then*

$$Pr[f(x) \neq f(N_k(x))] \leqslant 2\sqrt{k/n}$$

where $N_k(x)$ denotes the bit string x after exactly k random bit positions have been flipped.

Proof: Let π be a random permuation on $[n]$. Let $\ell = \lfloor n/k \rfloor$. Now define $Y_1 = \sum_{j=1}^{k} w_{\pi(j)}x_j, \ldots, Y_\ell = \sum_{j=(\ell-1)k+1}^{\ell k} w_{\pi(j)}x_j$ and $Z = \sum_{j=\ell k+1}^{n} x_j w_{\pi(j)}$. Let $\sigma_i = sgn(Y_i)$. Finally define $S_i = \sum_{j=1,\ldots,i} Y_j$ and $S_\ell = (\sum_{j=1,\ldots,\ell} Y_\ell) + Z$. Notice that S_ℓ simply equals $f(x)$. Our goal is to bound the following quantity

$$Pr[sgn(S_l) \neq f(N_k(x))] \tag{6.1}$$

The probability here is taken over a randomly chosen input x, permutation π and k randomly chosen flipped bits. Because of the random permutation we can assume without loss of generality that the flipped bits occur in the last block of inputs, $x_{(\ell-1)k+1}, \ldots, x_{\ell k}$. Notice that the output of $f(x)$ will change if and only if $|S_{\ell-1} + Z| < |Y_\ell|$. Let us set $Y_0 = Z$ and $\sigma_0 = sgn(Y_0)$. Thus we have

$$\begin{aligned}
Pr[sgn(S_l) \neq f(N_k(x))] &= Pr[|S_{\ell-1} + Z| < |Y_\ell|] && (6.2)\\
&= 1 - Pr[|S_{\ell-1} + Z| > |Y_\ell|] && (6.3)\\
&= 1 - 2Pr[\sum_{i=0,\ldots,\ell-1} \sigma_i|Y_i| - |Y_\ell| > 0] && (6.4)\\
&= 1 - 2Pr[sgn(\sum_{i=0,\ldots,\ell-1} -\sigma_i\sigma_\ell|Y_i| + \sigma_\ell|Y_\ell|) = -\sigma_\ell] && (6.5)
\end{aligned}$$

(Line 6.5 is obtained by multiplying by $-\sigma_\ell$). Note that $\sigma_0, \ldots, \sigma_{\ell-1}$ are independent of σ_ℓ, so, for every i, the value $-\sigma_i\sigma_\ell$ is distributed identically to σ_i. Hence we have

$$\begin{aligned} Pr[sgn(S_l) \neq f(N_k(x))] &= 1 - 2Pr[sgn(\sum_{i=0,\ldots,\ell-1} \sigma_i |Y_i| + \sigma_\ell |Y_\ell|) = -\sigma_\ell] & (6.6) \\ &= 1 - 2Pr[sgn(S_\ell) = -\sigma_\ell] & (6.7) \end{aligned}$$

We can rewrite line 6.7 as $2\mathsf{E}[1/2 - 1_{sgn(S_\ell)=-\sigma_\ell}]$. Notice, however, that the above argument did not rely at all on the fact that the *last* block of bits were flipped. That is to say we could also rewrite line 6.7 as $2\mathsf{E}[1/2 - 1_{sgn(S_\ell)=-\sigma_i}]$ for any i. Hence line 6.7 is equal to the average of the quantities:

$$Pr[sgn(S_l) \neq f(N_k(x))] = 2\mathsf{E}[1/2 - (1/\ell + 1)(\sum_{i=0,\ldots,\ell} 1_{sgn(S_\ell)=-\sigma_i})]$$

Using the simple fact that $\mathsf{E}[1_{sgn(S_\ell)=-\sigma_i}] \leqslant \mathsf{E}[|1_{sgn(S_\ell)=1}| + |1_{sgn(S_\ell)=-1}|]$ we have

$$\begin{aligned} Pr[sgn(S_l) \neq f(N_k(x))] &\leqslant 2\mathsf{E}[|1/2 - (1/\ell + 1) \sum_{i=0,\ldots,\ell} 1_{1=-\sigma_i}| + & (6.8) \\ &\quad |1/2 - (1/\ell + 1) \sum_{0=1,\ldots,\ell} 1_{-1=-\sigma_i}|] & (6.9) \\ &\leqslant 1/\sqrt{\ell} + 1/\sqrt{\ell} & (6.10) \\ &\leqslant 2\sqrt{k/n} & (6.11) \end{aligned}$$

Line 6.10 follows from the fact that the variance of $(1/(\ell+1)\sum_{i=0,\ldots,\ell} 1_{1=-\sigma_i})$ is $1/4(\ell+1)$, its expectation is $1/2$, and $\mathsf{E}[x]^2 \leqslant \mathsf{E}[x^2]$. ■

To prove Theorem 6.6.2 we can average this value over all choices of k.

Proof of Theorem 6.6.2:

Given that k bits are flipped, the probability that the value of f changes is less than $2\sqrt{k/n}$. To compute the noise stability of f we must now compute $\mathsf{E}[2\sqrt{k/n}]$ where the expectation is taken over the number of flips. This is at most $2\sqrt{\mathsf{E}[k/n]}$ which is $2\sqrt{\epsilon}$, as k is a Bernoulli random variable with parameter ϵ. We must now handle the case when the threshold is not equal to 0 as was assumed in Lemma

6.6.3. Assume that $f = sgn(\sum_{i=1}^{n} w_i x_i - \theta)$. Introduce a new weight $w_{n+1} = \theta$ and consider the functon $f' = sgn(\sum_{i=1}^{n+1} w_i x_i)$. Notice that $Pr[f'(x) \neq f'(N_\epsilon(x))]$ is less than $2\sqrt{\epsilon}$ since f' has threshold 0. Let $p = Pr[f'(x) \neq f'(N_\epsilon(x)) \mid x_{n+1} = -1$ and x_{n+1} does not flip$] = Pr[f(x) \neq f(N_\epsilon(x))]$. Then $2\sqrt{\epsilon} \geqslant (1/2)(1-\epsilon)p$ and so $p \leqslant 5\sqrt{\epsilon}$. ■

We can now prove a bound on the noise stability of an arbitrary function of k halfspaces:

Theorem 6.6.4 *Let $f : \{0,1\}^n \to \{-1,1\}$ be any function of k halfspaces. Then* $\mathsf{NoiseStab}_\epsilon(f) \leqslant 5k\sqrt{\epsilon}$.

Proof: Let $f(h_1, \ldots, h_k)$ be an arbitrary Boolean function of k halfspaces. Theorem 6.6.2 implies that for every i, $Pr[h_i(x) \neq h_i(N_\epsilon(x))]$ is $5\sqrt{\epsilon}$. The probability that some h_i changes value is at most $5k\sqrt{\epsilon}$. Thus, the probability the $f(h_1, \ldots, h_k)$ changes value is at most $5k\sqrt{\epsilon}$, as at least one argument to f must change for f's value to change. ■

6.6.2 From Noise Stability to Fourier Concentration

Now we need to establish the connection between the noise stability of a function and its Fourier concentration properties. The following proposition first appears in [15] who examined the noise stability of threshold functions and its application to percolation and circuit complexity. Recently O'Donnell [94] has used this fact as a complexity theoretic tool for showing how to amplify the hardness of a Boolean function within NP. The following proposition is taken from [94]:

Proposition 6.6.5 *[15, 94] Let f be a Boolean function on n inputs. Then*

$$\mathsf{NoiseStab}_\epsilon(f) = 1/2 - (1/2) \sum_{S \subset [n]} (1-2\epsilon)^{|S|} \hat{h}(S)^2$$

Bshouty *et al.* [31] establish a related but somewhat looser Fourier-based relationship than the one above that applies to arbitrary noise distributions.

Using Proposition 6.6.5 we can show that a $1-\epsilon$ fraction of a threshold function's Fourier weight lies on basis vectors of degree $O(1/\epsilon^2)$:

Theorem 6.6.6 *Let f be a function such that* $\mathsf{NoiseStab}_\epsilon(f) \leqslant \sqrt{\epsilon}$. *Then, for* $0 < \epsilon < 1/2$, $\mathsf{S}_{16/\epsilon^2}(f) \geqslant 1-\epsilon$.

Proof: From Proposition 6.6.5 we know $-\sum_{S\subset[n]}(1-2\epsilon)^{|S|}\hat{f}(S)^2 = 2\sqrt{\epsilon}-1$. Continuing we see:

$$\sum_{S\subset[n]}(1-2\epsilon)^{|S|}\hat{f}(S)^2 \geqslant 1-2\sqrt{\epsilon} \tag{6.12}$$

$$\sum_{S\subset[n]}(1-(1-2\epsilon)^{|S|})\hat{f}(S)^2 \leqslant 2\sqrt{\epsilon} \tag{6.13}$$

$$\sum_{S\subset[n],|S|\geqslant\frac{1}{\epsilon}}(1-(1-2\epsilon)^{|S|})\hat{f}(S)^2 \leqslant 2\sqrt{\epsilon} \tag{6.14}$$

Notice that when $|S| \geqslant \frac{1}{\epsilon}$, and $\epsilon < 1/2$, $1-(1-2\epsilon)^{|S|} \geqslant 1/2$. Since every coefficient of in the sum from line 6.14 is greater than $1/2$, we can conclude

$$\begin{aligned}\sum_{S\subset[n],|S|\geqslant 1/\epsilon}(1/2)\hat{f}(S)^2 &\leqslant 2\sqrt{\epsilon}\\ \sum_{S\subset[n],|S|\geqslant 1/\epsilon}\hat{f}(S)^2 &\leqslant 4\sqrt{\epsilon}\end{aligned}$$

To achieve a $1-\epsilon$ fraction of the spectrum we need to set $4\sqrt{\delta}=\epsilon$ and notice that less than an ϵ fraction of the Fourier spectrum resides on basis vectors of degree greater than $1/\delta = 16/\epsilon^2$:

Thus

$$\sum_{S\subset[n],|S|\leqslant 16/\epsilon^2}\hat{f}(S)^2 \geqslant (1-\epsilon)$$

■

Corollary 6.6.7 *Let $f : \{0,1\}^n \rightarrow \{-1,+1\}$ be a linear threshold function. Then for $0 < \epsilon < 1/2$, $\mathsf{S}_{O(1/\epsilon^2)}(f) \geqslant 1 - \epsilon$.*

We point out here that the bound from Corollary 6.6.7 is essentially tight. In [72], it is shown that for the case of the majority function, it is necessary to look at basis vectors of degree $O(1/\epsilon^2)$ to capture a $1 - \epsilon$ fraction of the Fourier spectrum.

We can also use Theorem 6.6.6 to characterize the Fourier spectrum of an arbitrary Boolean function of halfspaces:

Corollary 6.6.8 *Let $f : \{0,1\}^n \rightarrow \{-1,1\}^n$ compute an arbitrary Boolean function of k halfspaces. Then for $0 < \epsilon < 1/10k^2$, $\mathsf{S}_{O(k^2/\epsilon^2)}(f) \geqslant 1 - \epsilon$.*

Proof: From Theorem 6.6.4, $\mathsf{NoiseStab}_\epsilon(f) \leqslant 5k\sqrt{\epsilon}$. Thus, as indicated by the proof of Theorem 6.6.6, to obtain a $1 - \epsilon$ fraction of the Fourier weight we need only look at the coefficients of basis vectors of degree $O(k^2/\epsilon^2)$. ■

Combining Corollary 6.6.8 with Fact 6.5.2 we obtain our main learning result for intersections of halfspaces under the uniform distribution:

Theorem 6.6.9 *An arbitrary Boolean function of k halfspaces can be learned under the uniform distribution to accuracy $0 < \epsilon < 1/10k^2$ in time $n^{O(k^2/\epsilon^2)}$.*

Theorem 6.6.9 implies the existence of a polynomial time algorithm for learning any neural network where there are a constant number of "hidden" nodes to within any constant error parameter. Previous work (see for example [6, 76]) works in time exponential in the fan-in of the hidden nodes.

6.7 Learning Intersections of Halfspaces under Arbitrary Distributions

For our distribution independent results we return to the polynomial threshold function approach from Chapter 3. Using techniques from rational function approximation, we show that the intersection of any constant number of halfspaces with polynomially bounded weights can be computed by a relatively small degree polynomial threshold function. By generalizing the results from Chapter 3 we show that Chebyshev polynomials can be used to construct polynomial threshold functions which compute the intersection of many small weight halfspaces.

All of the learning results in this section also hold for the model of *exact learning from equivalence queries* [2] as it is known how to learn linear threshold functions in this model [86].

6.7.1 Rational Function Approximation

We use the rational function approximation to the sign function introduced by Beigel *et al.* [14], building on work of Newman [91]. The motivation of Beigel *et al.* was the study of the complexity class PP. The main technical theorem we need is:

Theorem 6.7.1 *[14] For every $\ell, t \geqslant 1$ there exists a rational function P_t^ℓ in one variable with the following properties:*

- *For any $x \in [1, 2^t]$ the value $P_t^\ell(x) \in [1, 1 + 1/\ell]$.*
- *For any $x \in [-2^t, -1]$ the value $P_t^\ell(x) \in [-1 - 1/\ell, -1]$.*
- *$P_t^\ell(x)$ has degree $O(t \log \ell)$.*
- *The coefficients of $P_t^\ell(x)$ are at most $2^{O(t^2 \log \ell)}$.*

We begin by showing how to express the intersection of two weight-w halfspaces as a polynomial threshold function of degree $O(\log w)$. Let $F(x_1, \ldots, x_n)$ and $G(x_1, \ldots, x_n)$ be two weight-w halfspaces and let $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$ be their underlying linear forms. Without loss of generality assume that f and g never output 0 (multiply each linear form by 2 and add 1). Set $\ell = 1/3$ and $t = \log w$ and consider the following sum of rational functions:

$$I = P_{\log w}^{(1/3)}(f(x_1, \ldots, x_n)) + P_{\log w}^{(1/3)}(g(x_1, \ldots, x_n))$$

Notice that if both of the summands are positive then by Theorem 6.7.1, I is at least 2. Similarly, if either one of the summands is negative then I is at most

1/3. Hence the function $(I - 1)$ sign represents the intersection of the two halfspaces on all inputs. By clearing denominators (multiply by the square of each denominator), we obtain a polynomial threshold function I' of degree $O(\log w)$ computing the intersection of F and G on all inputs.

Now we can extend this construction to the intersection of k halfspaces:

Theorem 6.7.2 *Let $H_1, \ldots, H_k$ be linear threshold functions mapping $\{-1,1\}^n \to \{-1,1\}$ each of weight at most w. The function $H_1(x) \wedge \cdots \wedge H_k(x)$ can be expressed as a polynomial threshold function of degree $O(k \log k \log w)$.*

Proof: Let $\ell = 2k$ and $t = \log w$ and consider the sum of rational functions

$$I = P^{2k}_{\log w}(h_1) + \cdots + P^{2k}_{\log w}(h_k)$$

where h_i is the linear form underlying H_i. On input x, if $h_i(x) \geqslant 0$ for every i, then $I \geqslant k$. On the other hand if for some i, $h_i(x) < 0$ then $I \leqslant k - 1$. Thus $sgn(I - k)$ computes $H_1(x) \wedge \cdots \wedge H_k(x)$ on all inputs. Clearing denominators results in a polynomial threshold function of degree $O(k \log k \log w)$. ■

As a simple generalization to this theorem we obtain

Theorem 6.7.3 *Let $H_1, \ldots, H_m$ be weight-w halfspaces on $\{-1,1\}^n$ and let G be a weight-k halfspace on $\{-1,1\}^m$. Then $\{G(H_1(x), \ldots, H_m(x))\}$ of weight-k thresholds of weight-w halfspaces can be expressed as a polynomial threshold function of degree $O(k \log k \log w)$.*

Applying Fact 6.5.6 we have the following learning result:

Theorem 6.7.4 *Let $H_1, \ldots, H_m$ be weight-w halfspaces on $\{-1,1\}^n$ and let G be a weight-k halfspace on $\{-1,1\}^m$. The class $\{G(H_1(x), \ldots, H_m(x))\}$ of weight-k thresholds of weight-w halfspaces can be learned to accuracy ϵ under any distribution in time $n^{O(k \log k \log w)}/\epsilon$.*

Our results can be extended to give strong learning algorithms for arbitrary functions of halfspaces:

Theorem 6.7.5 *An arbitrary Boolean function of k halfspaces each of weight w can be be learned in time $n^{O(k^2 \log w)}$ with respect to an arbitrary distribution.*

Proof: Let $M : \{-1,1\}^k \rightarrow \{-1,1\}$ be any Boolean function of k inputs. We wish to express $M(H_1, \ldots, H_k)$ (where each H_i is a weight w halfspace) as a polynomial threshold function of degree $O(k^2 \log w)$. To do so we first apply the above rational function approximation to each underlying linear form h_i where $t = \log w$ and $\ell = 2^{-3k}$. Since M is a boolean function on k inputs, there exists a multilinear polynomial in k variables $\tilde{M}$ with at most 2^k terms (and the coefficient of each term has magnitude at most 2^k) interpolating M on all $\{-1,1\}^k$ inputs. Consider the function $I = \tilde{M}(P^{2^{-3k}}_{\log w}(h_1), \ldots, P^{2^{-3k}}_{\log w}(h_k))$. I equals the sum of at most 2^k rational functions each having degree at most $O(k^2 \log w)$. Notice that due to our choice of ℓ, for every input x, each term of the polynomial I differs in value by at most 2^{-2k} from the value of its corresponding term in the polynomial $M(H_1, \ldots, H_k)$. Therefore, the output of $I(x)$ differs from the output of $M(x)$ by at most 2^{-k}, and the function $(I - 1/2)$ sign represents the function M on all inputs. After multiplying through by the denominator of each $P^{2^{-2k}}_{\log w}(h_i)$, we obtain a polynomial threshold function of degree $O(k^2 \log w)$ computing M. Applying Fact 6.5.6 finishes the proof. ■

Siu *et al.* [108] have proved a similar result to Theorem 6.7.5. They showed that any depth 3 threshold circuit with t_1 gates in the first level and t_2 gates in the second level can be written as the sign of a rational function of degree $O(t_1 t_2 log^4 n)$ and use this fact to prove a lower bound on the size of threshold circuits computing the parity function.

6.7.2 Learning the AND of many small-weight halfspaces in subexponential time

We can give an alternate construction of a polynomial threshold function computing an intersection of halfspaces by giving a simple generalization of the main theorem from Chapter 3:

Theorem 6.7.6 *5.1.1 Any s-term t-DNF can be expressed as a polynomial threshold function of degree $O(t^{1/2} \log s)$.*

Our generalization is the following:

Theorem 6.7.7 *Let $h_1, \ldots, h_k$ be linear threshold functions mapping $\{-1,1\}^n \to \{-1,1\}$ each of weight at most w. The function $h_1(x) \wedge \cdots \wedge h_k(x)$ can be expressed as a polynomial threshold function of degree $O(\sqrt{w} \log k)$.*

Notice that in contrast with Theorem 6.7.3, Theorem 6.7.7 gives interesting bounds when considering a polynomial number of halfspaces where each halfspace has small weight (e.g. weight $O(1)$).

The main tools used here as in Chapter 3 are the Chebyshev polynomials of the first kind. These polynomials play an important role in numerical analysis and approximation theory; we need only a few simple properties. The d-th Chebyshev polynomial of the first kind, $T_d(x)$, is a univariate degree-d polynomial which satisfies the following conditions [34]:

Lemma 6.7.8 *The polynomial $T_d(x)$ satisfies*

- $|T_d(x)| \leqslant 1$ *for* $|x| \leqslant 1$ *with* $T_d(1) = 1$;
- $T'_d(x) \geqslant d^2$ *for* $x > 1$ *with* $T'_d(1) = d^2$.

Proof of Theorem 6.7.7: The proof is a generalization of the proof of Theorem 5.1.1. Let $f = H_1 \wedge H_2 \wedge \cdots \wedge H_k$ be an intersection of k halfspaces, each of weight at most w. We let 1 represent TRUE so $f(x) = 1$ iff each $H_i(x) = 1$. Let

$$p(y) = T_d\left(y\left(1 + \frac{1}{w}\right)\right)$$

where $d = \lceil \sqrt{w} \rceil$. Lemma 6.7.8 implies that for $y \geqslant 1$ we have $p(y) \geqslant 2$ but $|p(y)| \leqslant 1$ for $y \in [-1 + \frac{1}{w}, 1 - \frac{1}{w}]$.

Let $w_i \leqslant w$ denote the weight of the i-th halfspace H_i, so $H_i(x) = \mathrm{sgn}(w^i \cdot x - \theta_i)$ where $|\theta_i| + \sum_{j=1}^n |w^i_j| = w_i$. (Without loss of generality we may assume that each w_i

and w are at least 2.) Let

$$Q_i(x) = p\left(1 - \frac{w^i \cdot x - \theta_i + 1}{w_i}\right).$$

Consider the polynomial threshold function "$P(x) \geqslant -k - \frac{1}{2}$" where

$$P(x) = \sum_{i=1}^{k} -Q_i(x)^{\lceil \log 2k \rceil}.$$

Since p is a polynomial of degree $d = \lceil \sqrt{w} \rceil$ and $w^i \cdot x$ is a polynomial of degree 1, this polynomial threshold function has degree $\lceil \sqrt{w} \rceil \lceil \log 2k \rceil$. We now show that this polynomial threshold function computes f exactly on $\{-1,1\}^n$.

Fix any element $x \in \{-1,1\}^n$.

- If $f(x) = 1$ then for each i we have $0 \leqslant w^i \cdot x - \theta_i \leqslant w_i$ and hence $-1 + \frac{1}{w} \leqslant \frac{-1}{w_i} \leqslant 1 - \frac{w^i \cdot x - \theta_i + 1}{w_i} \leqslant 1 - \frac{1}{w}$. Consequently for each i we have $|Q_i(x)| \leqslant 1$ and hence $-Q_i(x)^{\lceil \log 2k \rceil} \geqslant -1$ so $P(x) \geqslant -k$.

- If $f(x) = 0$ then for some i we have $w_i \cdot x - \theta_i \leqslant -1$ and hence $1 - \frac{w^i \cdot x - \theta_i + 1}{w_i} \geqslant 1$. Consequently $Q_i(x) \geqslant 2$ and hence $-Q_i(x)^{\lceil \log 2k \rceil} \leqslant -2k$. Since $-Q_i(x)^{\lceil \log 2k \rceil} \leqslant 1$ for all i, we have $P(x) \leqslant -2k + (k-1) = -k - 1$. □

Theorem 6.7.7 immediately implies the following learning result:

Theorem 6.7.9 *The intersection of k weight-w halfspaces is learnable with respect to an arbitrary distribution in time $n^{O(w^{1/2} \log k)}$.*

6.7.3 Lower bounds on the degree of polynomial threshold functions

For certain ranges of k and w the bounds of Theorem 6.7.7 and Theorem 6.7.2 are nearly optimal. Results of Minsky and Papert [88, 74] imply that for $w \leqslant n^{2/3}$ and $k \geqslant \sqrt{w}$ there is an intersection of k halfspaces of weight w such that any polynomial threshold function must have degree $\Omega(\sqrt{w})$. In the case that $w = k^2$, a polynomial threshold function of degree $\Omega(k)$ is required, and the bounds from Theorem 6.7.7 and Theorem 6.7.2 are $O(k \log k)$ and $O(k \log^2 k)$ respectively. Hence it is not possible to achieve say a polylog(k, w) degree bound for an arbitrary intersection of k halfspaces of weight w.

The degree bound we prove in Theorem 6.7.3 is also nearly optimal. Hajnal *et al.* [53] show how to compute the parity function on k variables as a weight-$O(k)$ threshold of weight-$O(k)$ halfspaces, but any polynomial threshold function computing parity on k variables requires degree k (see [5]). Hence our upper bound from Theorem 6.7.3 of $O(k \log^2 k)$ is off by at most a factor of $O(\log^2 k)$.

Chapter 7

Conclusions and Open Problems

In this thesis we have demonstrated the usefulness of re-examining well studied concept classes in machine learning with tools traditionally reserved for complexity theoretic problems. Strictly speaking the algorithms we give in this thesis have been known to the learning community for years. Our algorithms for learning DNF formulae and intersections of halfspaces under an arbitrary distribution use linear programming methods [25] to learn a single linear threshold function. The algorithms for learning DNF formulae, constant depth circuits with majority gates, and intersections of halfspaces under the uniform distribution (from Chapters 4, 5, and 6 respectively) rely heavily on a Fourier sampling technique introduced over ten years ago by Linial et al. [81].

While the algorithms themselves were known for years, the representations of concept classes we have given are new and are a direct result of our complexity theoretic approach. We believe that further application of complexity theoretic machinery will result in improved structural characterizations of concept classes. We state a few open problems along these lines:

- What is the lowest degree polynomial threshold function computing an AC^0 circuit? We can only prove bounds for read once AC^0 (see Chapter 3). Even for the case of depth-3 circuits no upper bound less than n is known.

- What is the lowest degree polynomial threshold function computing the intersection of two halfspaces? Again nothing is known. A bound of $o(n)$ would immediately imply the first subexponential time algorithm for learning the intersection of two halfspaces with respect to an arbitrary distribution. Can one prove a lower bound of $\Omega(\log n)$?

- What is the noise stability of the intersection of k halfspaces? We give an upper bound of $O(k\sqrt{\epsilon})$, but this seems naive, given that we only make use of the union bound. For example in the case of a read once intersection of halfspaces, one can prove a significantly better bound (see [72]). An improved bound would imply better uniform distribution learning algorithms for intersections of halfspaces.

- What is the lowest degree polynomial threshold function computing the intersection of k, weight w halfspaces? For an arbitrary function of halfspaces our bounds are tight to within a polylogarithmic factor (see Chapter 6); is it possible to do better when we are only considering an intersection of weight bounded halfspaces?

More generally, it seems that the most expressive uniform distribution learning algorithms all make use of Fourier analysis while distribution independent algorithms rely on linear programming. Is it possible to combine these two techniques? What does a structural characterization of a concept class via polynomial threshold functions say about its Fourier spectrum?

7.1 Complexity Theoretic Applications

Techniques from computational learning theory may be of use in proving new results in complexity theory. In Chapter 4 we saw how to use boosting algorithms to develop new hard core set constructions. In [71], the author uses boosting to give a proof of the inapproximability of parity by AC^0 without using the Hastad Switching Lemma [54]. Is it possible to derive further results along these lines? What complexity theoretic implications does our boosting based learning algorithm from Chapter 5 have for constant depth circuits with majority gates (for example, can one build a hitting-set generator for such a circuit class?). What other derandomization implications are there for learning techniques aside from hard core set construction? Impagliazzo and Wigderson [60] rephrase certain pseudo-random generator constructions in terms of learning problems. Can learning theory be used to further our knowledge in derandomization?

Bibliography

[1] S. Agmon. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6(3):382-392,1954.

[2] D. Angluin. Queries and concept learning. *Machine Learning* **2** (1988), 319-342.

[3] M. Anthony. Classification by polynomial surfaces. *Discrete Applied Mathematics* **61** (1995), 91-103.

[4] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 1998.

[5] J. Aspnes, R. Beigel, M. Furst and S. Rudich. The expressive power of voting polynomials. *Combinatorica* **14**:2 (1994), 1-14. Earlier version in "Proc. 23rd ACM Symposium on Theory of Computation" (1991), 402-409.

[6] P. Auer, S. Kwek, W. Maass, and M. Warmuth. Learning of Depth Two Neural Networks with Constant Fan-in at the Hidden Nodes. In *Proceedings of the 9th Annual Conference on Computational Learning Theory*, New York, 1996.

[7] L. Babai, L. Fortnow, N. Nisan, and A.Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[8] L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36 (1988), 254-276.

[9] E. Baum. A polynomial time algorithm that learns two hidden unit nets. *Neural Computation*, 2:510–522, 1991.

[10] R. Beigel. The polynomial method in circuit complexity, in "Proc. 8th Conf. on Structure in Complexity Theory" (1993), 82-95.

[11] R. Beigel. When do extra majority gates help? polylog(n) majority gates are equivalent to one. *Computational Complexity*, 4:314–324, 1994.

[12] R. Beigel, Personal communication, 2000.

[13] R. Beigel, N. Reingold and D. Spielman. The perceptron strikes back, *in* "Proc. 6th Conf. on Structure in Complexity Theory" (1991), 286-291.

[14] R. Beigel, N. Reingold, and D. Spielman. PP is closed under intersection. *Journal of Computer and System Sciences*, 50(2):191–202, 1995.

[15] I. Benjamini, G. Kalai, and O. Schramm. Noise sensitivity of Boolean functions and applications to percolation. *Institute des Haute Etudes Scientifique*, vol 90, pp5–43, 1999.

[16] C. Berg and S. Ulfberg. A lower bound for perceptrons and an oracle separation of the PP hierarchy. *J. Comput. Syst. Sci.* **56**:3 (1998), 263-271.

[17] H. Block. The Perceptron: a model for brain functioning. *Reviews of Modern Physics*, 34: 123–135, 1962.

[18] A. Blum. Rank-r decision trees are a subclass of r-decision lists. *Information Processing Letters* **42**:4 (1992), 183-185.

[19] A. Blum, C. Burch, and J. Langford. On Learning Monotone Boolean Functions. In *Proceedings of the 39th Annual Foundations of Computer Science*, 1998.

[20] A. Blum, P. Chalasani, S. Goldman, and D. Slonim. Learning with Unreliable Boundary Queries. *Journal of Computer and System Sciences* **56**(2) (1998), 209-222.

[21] A. Blum, A. Frieze, R. Kannan, and S. Vempala. A polynomial time algorithm for learning noisy linear threshold functions, *in* "Proc. 37th Symp. on Found. of Comp. Sci." (1996), 330-338.

[22] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. Rudich. Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In *Proceedings of the Twenty-Sixth Annual Symposium on Theory of Computing*, pages 253–262, 1994.

[23] A. Blum and R. Kannan. Learning an intersection of k halfspaces over a uniform distribution. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, 1993.

[24] A. Blum and S. Rudich. Fast learning of k-term DNF formulas with queries. *J. Comp. Syst. Sci.* **51**(3) (1995), 367-373.

[25] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[26] D. Boneh and R. Lipton. Amplification of weak learning over the uniform distribution. In *Proceedings of the Sixth Annual Workshop on Computational Learning Theory*, pages 347–351, 1993.

[27] J. Bruck. Harmonic analysis of polynomial threshold functions. *SIAM Journal on Discrete Mathematics*, 3(2):168–177, 1990.

[28] N. Bshouty. A subexponential exact learning algorithm for DNF using equivalence queries. *Information Processing Letters* **59** (1996), 37-39.

[29] N. Bshouty and D. Gavinsky. On boosting with optimal poly-bounded distributions. In *Proceedings of the Fourteenth Annual Conference on Computational Learning Theory / Fifth European Conference on Computational Learning Theory*, pages 490–506, 2001.

[30] N. Bshouty, J. Jackson, and C. Tamon. More efficient pac learning of dnf with membership queries under the uniform distribution. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 286–295, 1999.

[31] N. Bshouty, J. Jackson, and C. Tamon. Uniform-distribution attribute noise learnability. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 75–80, 1999.

[32] N. Bshouty and C. Tamon. On the Fourier spectrum of monotone functions. *Journal of the ACM*, 43(4):747–770, 1996.

[33] T. Bylander. Worst-case analysis of the perceptron and exponentiated update algorithms. *Artificial Intelligence* **106** (1998).

[34] E. W. Cheney. *Introduction to approximation theory.* McGraw-Hill, 1966.

[35] E. Cohen. Learning noisy perceptrons by a perceptron in polynomial time, *in* "Proc. 38th Symp. on Found. of Comp. Sci." (1997), 514-523.

[36] C. Domingo and O. Watanabe. Madaboost: a modified version of adaboost. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, pages 180–189, 2000.

[37] H. Drucker and C. Cortes. Boosting decision trees. In *Advances in Neural Information Processing Systems 8*, pages 479–485, 1996.

[38] H. Drucker, C. Cortes, L. D. Jackel, Y. Lecun, and V. Vapnik. Boosting and other ensemble methods. *Neural Computation*, 6(6):1289–1301, 1994.

[39] H. Drucker, R. Schapire, and P. Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Machine Intelligence*, 7(4):705–719, 1993.

[40] H. Drucker, R. Schapire, and P. Simard. Improving performance in neural networks using a boosting algorithm. In *Advances in Neural Information Processing Systems 5*, pages 42–49, 1993.

[41] A. Ehrenfeucht and D. Haussler. Learning decision trees from random examples. *Information and Computation* **82**:3 (1989), 231-246.

[42] A. Ehrenfeucht, D. Haussler, M. Kearns and L. Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation* **82**:3 (1989), 247-251.

[43] Y. Freund. Boosting a weak learning algorithm by majority. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 202–216, 1990.

[44] Y. Freund. An improved boosting algorithm and its implications on learning complexity. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 391–398, 1992.

[45] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

[46] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[47] Y. Freund and R. Schapire. Large margin classification using the perceptron algorithm, *in* "Proc. Eleventh Ann. Conf. on Comp. Learning Theory" (1998), 209-217.

[48] B. Fu. Separating PH from PP by relativization. *Acta Math. Sinica* **8**:3 (1992), 329-336.

[49] M. Furst, J. Saxe, and M. Sipser. Parity, Circuits, and the Polynomial-Time Hierarchy. *Mathematical Systems Theory*, 17(1), 1984.

[50] M. Goldmann, J. Haastad and A. Razborov. Majority gates vs. general weighted threshold gates. *Computational Complexity* **2** (1992), 277-300.

[51] S. Goldwasser,S. Micali, and C. Rackoff. The knowledge complexity of interactive proofs. SIAM Jounral on Computing, 18(1) 186–208, 1989.

[52] O. Goldreich, N. Nisan, and A. Wigderson. On Yao's XOR-lemma. Electronic Colloquium on Computational Complexity TR95-050, 1995.

[53] A. Hajnal, W. Maass, P. Pudlak, M. Szegedy, and G. Turan. Threshold circuits of bounded depth. *Journal of Computer and System Sciences*, 46:129–154, 1993.

[54] J. Hastad. *Computational Limitations for Small Depth Circuits.* MIT Press, Cambridge, MA, 1986.

[55] J. Hastad. Some optimal inapproximability results. *Journal of the ACM*, 48:798–859, 2001.

[56] T. Hancock and Y. Mansour. Learning monotone k-μ DNF formulas on product distributions, *in* "Proc. 4th Ann. Workshop on Comp. Learning Theory" (1991), 179-183.

[57] D. Helmbold, R. Sloan and M. Warmuth. Learning nested differences of intersection-closed concept classes. *Machine Learning* **5** (1990), 165-196.

[58] R. Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proceedings of the Thirty-Sixth Annual Symposium on Foundations of Computer Science*, pages 538–545, 1995.

[59] R. Impagliazzo and A. Widgerson. P = BPP unless E has subexponential circuits: derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual Symposium on Theory of Computing*, pages 220–229, 1997.

[60] R. Impagliazzo and A. Widgerson. Randomness vs. time: de-randomization under a uniform assumption. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, 1998.

[61] J. Jackson. *The Harmonic sieve: a novel application of Fourier analysis to machine learning theory and practice.* PhD thesis, Carnegie Mellon University, August 1995.

[62] J. Jackson. An efficient membership-query algorithm for learning dnf with respect to the uniform distribution. *Journal of Computer and System Sciences*, 55:414–440, 1997.

[63] J. Jackson and M. Craven. Learning sparse perceptrons. In *Advances in Neural Information Processing Systems 8*, pages 654–660, 1996.

[64] M. Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proceedings of the Twenty-Fifth Annual Symposium on Theory of Computing*, pages 372–381, 1993.

[65] M. Kearns *The Computational Complexity of Machine Learning.* The MIT Press, 1990.

[66] M. Kearns, M. Li, and L. Valiant. Learning Boolean Formulas. *Journal of the ACM*, 41(6), 1298–1328, 1994.

[67] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, 1994.

[68] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory.* The MIT Press, 1994.

[69] R. Khardon. On using the Fourier transform to learn disjoint DNF. *Inf. Proc. Lett.* **49** (1994), 219-222.

[70] J. Kivinen, M. Warmuth, and P. Auer. The perceptron algorithm vs. winnow: linear vs. logarithmic mistake bounds when few input variables are relevant, *in* "Proc. 8th Conf. on Computational Learning Theory," (1995), 289-296.

[71] A. Klivans. On the Derandomization of Constant Depth Circuits. In *Proceedings of the 5th International Workshop on Randomization and Approximation Techniques in Computer Science* (RANDOM), 2001.

[72] A. Klivans, R. O'Donnell, and R. Servedio. New Algorithms for Learning Intersections and Thresholds of Halfspaces. In preparation, 2002.

[73] A. Klivans and R. Servedio. Boosting and hard-core sets. In *Proceedings of the Fortieth Annual Symposium on Foundations of Computer Science*, pages 624–633, 1999.

[74] A. Klivans and R. Servedio. Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. In *Proceedings of the Twenty-Sixth Annual Symposium on Theory of Computing*, 2001.

[75] L. Kucera, A. Marchetti-Spaccamela and M. Protassi. On learning monotone DNF formulae under uniform distributions. *Inf. and Comput.* **110** (1994), 84-95.

[76] P. Koiran and J. Shawe-Taylor. Techniques in neural learning. NeuroCOLT Technical Report 95-36.

[77] M. Krause and P. Pudlak. On the computational power of depth 2 circuits with threshold and modulo gates. In *Proceedings of the Twenty-Sixth Annual Symposium on Theory of Computing*, pages 48–57, 1994.

[78] S. Kwek and L. Pitt. PAC Learning Intersections of Halfspaces with Membership Queries. *Algorithmica* **22**(1/2), pages 53–75, 1998.

[79] E. Kushilevitz and D. Roth. On learning visual concepts and DNF formulae, *in* "Proc. Sixth Ann. ACM Conference on Computational Learning Theory" (1993), 317-326.

[80] L. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

[81] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform and learnability. *Journal of the ACM*, 40(3):607–620, 1993.

[82] N. Linial and N. Nisan. Approximate inclusion-exclusion. *Combinatorica* **10**:4 (1990), 349-365.

[83] N. Littlestone. Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning* **2** (1988), 285-318.

[84] S. Lokam, Personal communication (2001).

[85] Y. Mansour. Learning Boolean functions via the Fourier transform. In *Theoretical Advances in Neural Computation and Learning* (V.P. Roychodhury and K-Y. Siu and A. Orlitsky, ed.), 1994.

[86] W. Maass and G. Turan. How fast can a threshold gate learn? *in* "Computational Learning Theory and Natural Learning Systems: Volume I: Constraints and Prospects," S. J. Hanson, G. Drastal, & R. Rivest, eds., MIT Press (1994), 381-414.

[87] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes.* North-Holland, New York, 1977.

[88] M. Minsky and S. Papert. *Perceptrons.* MIT Press, 1968 (expanded edition 1988).

[89] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[90] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *Proceedings of the Thirty-Eighth Annual Symposium on Foundations of Computer Science*, pages 458–467, 1997.

[91] D.J. Newman. Rational approximation to $|x|$. *Michigan Mathematical Journal*, 11:11-14,1964.

[92] N. Nisan and A. Wigderson. Hardness versus randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[93] A. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on Mathematical Theory of Automata*, Vol. 12, 1962.

[94] R. O'Donnell. Hardness amplification within NP. In *Proceedings of the 34th Symposium on Theory of Computing*, 2002.

[95] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[96] Y. Peres. Personal communication, 2002.

[97] L. Pitt and L. Valiant. Computational Limitations on Learning from Examples. *Journal of the ACM*, 35(4), 1988.

[98] F. Rosenblatt. The Perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65, 386–407, 1958.

[99] Y. Sakai and A. Maruoka. Learning monotone log-term DNF formulas under the uniform distribution. *Theory Comput. Systems* **33** (2000), 17-33.

[100] R. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[101] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 80–91, 1998.

[102] M. Schmitt. Identification criteria and lower bounds for Perceptron-like learning rules. *Neural Computation* **10** (1998), 235-250.

[103] R. Servedio. On PAC learning using Winnow, Perceptron, and a Perceptron-like algorithm, *in* "Proc. Twelfth Ann. Conf. on Comp. Learning Theory" (1999), 296-307.

[104] R. Servedio. On Learning Monotone DNF under Product Distributions. In *Proceedings of the Fourteenth Annual Conference on Computational Learning Theory*, pages 558–573, 2001.

[105] R. Servedio. Smooth boosting and learning with malicious noise. In *Proceedings of the Fourteenth Annual Conference on Computational Learning Theory*, pages 473–489, 2001.

[106] R. Servedio. *Efficient Algorithms in Computational Learning Theory.* PhD thesis, Harvard University, 2001.

[107] R. Shaltiel. Towards proving strong direct product theorems. In *Proceedings of the Sixteenth Conference on Computational Complexity*, pages 107–117, 2001.

[108] K. Siu, V. Roychowdhury, and T. Kailath. Rational Approximation Techniques for Analysis of Neural Networks. In *IEEE Transactions on Information Theory*, Vol. 40(2), 467–474, 1994.

[109] J. Tarui and T. Tsukiji. Learning DNF by approximating inclusion-exclusion formulae. In *Proc. IEEE Conference on Computational Complexity*, 215-220, 1999.

[110] L. Trevisan. Construction of extractors using pseudo-random generators. In *Proceedings of the 31st ACM Symposium on the Theory of Computing*, pages 141–148, 1999.

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

[112] S. Vempala. A random sampling based algorithm for learning the intersection of halfspaces. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 508–513, 1997.

[113] K. Verbeurgt. Learning DNF under the uniform distribution in quasi-polynomial time, *in* "Proc. 3rd Ann. Workshop on Comp. Learning Theory" (1990), 314-326.

[114] K. Verbeurgt. Learning sub-classes of monotone DNF on the uniform distribution, *in* "Proc. 9th Conf. on Algorithmic Learning Theory" (1998), 385-399.

[115] N. Vereshchagin. On the power of PP, *in* "Proc. 7th Conf. on Structure in Complexity Theory" (1992), 138-143.

[116] A. Wigderson. Personal communication, 1999.

[117] Z. L. Zhang. Complexity of symmetric functions in perceptron-like models. Master's thesis, University of Massachusetts at Amherst, 1992.