On the Learnability of Monotone Functions

Homin K. Lee

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2009

ABSTRACT

On the Learnability of Monotone Functions

Homin K. Lee

A longstanding lacuna in the field of computational learning theory is the learnability of succinctly representable monotone Boolean functions, *i.e.,* functions that preserve the given order of the input. This thesis makes significant progress towards understanding both the possibilities and the limitations of learning various classes of monotone functions by carefully considering the complexity measures used to evaluate them.

We show that Boolean functions computed by polynomial-size monotone circuits are hard to learn assuming the existence of one-way functions. Having shown the hardness of learning general polynomial-size monotone circuits, we show that the class of Boolean functions computed by polynomial-size depth-3 monotone circuits are hard to learn using statistical queries. As a counterpoint, we give a statistical query learning algorithm that can learn *random* polynomial-size depth-2 monotone circuits (*i.e.,* monotone DNF formulas).

As a preliminary step towards a fully polynomial-time, proper learning algorithm for learning polynomial-size monotone decision trees, we also show the relationship between the average depth of a monotone decision tree, its average sensitivity, and its variance.

Finally, we return to monotone DNF formulas, and we show that they are *teachable* (a different model of learning) in the average case. We also show that non-monotone DNF formulas, juntas, and sparse GF_2 formulas are teachable in the average case.

Table of Contents

List of Figures

Acknowledgments

I would like to express my gratitude [Hyl04] to everybody who has had a hand in helping this dissertation come to fruition. I am deeply indebted to my advisors Rocco Servedio and Tal Malkin. Their tutelage, patience, and encouragement were invaluable to me in this research. In particular, I would like to thank them for the insightful, *"Wake up Mr. Sleepy! Your unconscious mind is dead!"* I would also like to thank Moti Yung for his guidance and support, especially during the early years of my graduate career.

I wish to single out Andrew Wan, who collaborated with me in much of my research, for special acknowledgement. He has been a constant source of inspiration for me. I would like to thank the Columbia University Faculty House for many fine lunches, as well as Emily Ford, Carey Kasten, Jonathan Rick, Erica Siegel, and Paul Vernon for eating these lunches with me.

I would also like to thank my colleagues and friends — including Spyridon Antonakopoulos, Seung Geol Choi, Justin Cranshaw, Dana Dachman-Soled, Ilias Diakonikolas, Ariel Elbaz, Ragesh Jaiswal, Troy Lee, Kevin Matulef, Li-Yang Tan, Emanuele Viola, and Hoeteck Wee — for many inspiring conversations, rousing procrastinations, and the continued success of Dixon's Book Club.

I would like to thank my co-authors — Dana Dachman-Soled, Vitaly Feldman, Jeffrey Jackson, Tal Malkin, Rocco Servedio, Andrew Wan, and Hoeteck Wee — for their contributions to this thesis. Thanks to Adam Klivans, Tal Malkin, Rocco Servedio, Mihalis Yannakakis, and Moti Yung for serving on my thesis committee.

Finally, I would like to thank Christelle Palpacuer for her love. *C'est fin c'est très fin ça se mange sans faim.*

This thesis is dedicated to all the sad boys and party girls.

Chapter 1

Introduction

...the problem of distinguishing between projectible and non-projectible hypotheses, is as important as it is exasperating

– Nelson Goodman, *Fact, Fiction, and Forecast*

The commonplace phenomenon of humans learning new concepts has been extensively studied by learning theorists in fields as disparate as psychology, philosophy, statistics, education, and computer science. Concepts are considered to be the constituents of thoughts and are essential to most accounts of human cognition, yet we know very little about the types of concepts humans can learn. It is intuitively clear that classes of simple concepts (*e.g.,* the rules of board games) are easier to learn than more complex ones (*e.g.,* the ideas of hermeneutical phenomenology), but to formalize this intuition we need a notion of complexity and a clear model of learning. These are precisely the tools that computational learning theory provides.

One of the difficulties in creating a coherent theory of learnability is that the same concept can be expressed in many different ways. For instance, the concept of [city] can be expressed by the words 'city', 'metropolis', '*urbs*', or 'an inhabited place of greater size, population, or importance than a town or village'. If we measured the complexity of the concept [city] by the number of letters it took to express it, the last expression would imply that [city] is a complex concept, whereas the first expression would imply that it is a simple one. This phenomenon should be familiar to readers of academic writing where it is often unclear if an idea being discussed is inherently complex or simply poorly expressed.

For a learner to have any hope of learning a class of concepts, the concepts should be expressible reasonably succinctly under *some* representation scheme. The major goal in computational learning theory is to identify classes of concepts with succinct representations that can be learned efficiently. In particular, a lot of effort has been made to identify efficiently learnable classes of binary classification concepts, which we model as Boolean functions. (*E.g.,* [animal?] is the concept that classifies all the objects in the world as being an animal or not an animal.)

The most influential and widely used model of learning comes from Valiant's seminal paper "A Theory of the Learnable" [Val84] where he suggested the concrete and robust Probably Approximately Correct, or PAC, model of concept learning. In the PAC model, a learning algorithm for a class of concepts C is given examples drawn from a fixed distribution labeled according to a fixed but unknown target concept $c \in C$. The goal of the algorithm is to efficiently generate a hypothesis under *any* representation scheme, which, with high probability, accurately approximates the target concept relative to the distribution over the domain. Ideally, the learning algorithm will work for all concepts $c \in C$ and all distributions over the examples.

The information-theoretic learnability of classes under the PAC model is well-understood [BEHW89, EHKV89]. However, the complexity measures (*e.g.,* the Vapnik-Chervonenkis dimension) used to characterize the information-theoretic learnability of concept classes are too coarse to differentiate between classes that are easy to learn efficiently and those that are provably impossible to do so. For many basic concept classes polynomially-many samples are enough to learn the concept information-theoretically, but the information-theoretic bounds give us little, if any, insight into the computational hardness of the problem.

Moreover, despite intensive research, no algorithm is currently known that learns arbitrary Boolean functions efficiently with respect to any reasonable representation scheme even though there is no information-theoretic impediment to learning from a polynomial number of examples. Due to this lack of success, researchers have tried to learn various restricted classes of Boolean functions. The most natural and closely studdied class is the class of all monotone functions. Monotone functions are functions that preserve the given order of the input. For $f : \{0,1\}^n \rightarrow \{0,1\}$ these are functions that satisfy $f(x) \geq f(y)$ whenever $x \geq y$ in the partial order on $\{0,1\}^n$. Most "natural" classification concepts that humans use (such as [animal?]) are in some sense monotone, and monotone functions correspond to many natural representation schemes (*e.g.,* circuits including no negations).

Showing that classes of monotone functions with succinct representations are easy or hard to learn efficiently would have significant practical implications as well as theoretical interest. Efficient algorithms would have immediate machine learning applications, whereas impossibility results would show the limits of what we can expect from learning systems in the real world. Studying the learnability of these concept classes will also gives us insight into computational complexity in general. These learning problems, like the famous problems of graph isomorphism and factoring, have resisted being classified as being in P (*i.e.,* solvable efficiently) or being NP-hard (*i.e.,* as hard as the hardest problems that can be efficiently verified).

1.1 Overview

This doctoral thesis seeks to enhance our understanding of efficiently learnable concept classes by carefully considering the complexity measures used to evaluate them. The main contribution of this thesis is the significant progress made on understanding both the possibilities and the limitations of learning various classes of monotone functions.

- **Chapter 2 – Background**

  In this chapter we define the PAC learning model, present some mathematical background material, and give a brief history of important results in learning monotone functions.

- **Chapter 3 – The Cryptographic Hardness of Learning Monotone Functions**

  Over the years a range of positive algorithmic results have been obtained for learning various classes of monotone Boolean functions from uniformly distributed random examples. To date, the only negative result for learning monotone functions in this model is a non-explicit lower bound showing that certain superpolynomial-size monotone circuits cannot be learned to accuracy $1/2 + \omega(\log n)/\sqrt{n}$ [BBL98]. This is in contrast with the situation for non-monotone functions, where a wide range of cryptographic hardness results are known establishing that various "simple" classes of polynomial-size circuits are not learnable by polynomial-time algorithms.

In this chapter we establish cryptographic hardness results for learning various classes of "simple" monotone circuits, giving a computational analogue of the information-theoretic hardness results mentioned above. Lower bounds of the type we establish have previously only been known for non-monotone functions. Some of our results show the cryptographic hardness of learning polynomial-size monotone circuits to an accuracy bound that is close to optimal by known positive results. Our main tool is a complexity-theoretic approach to hardness amplification via noise sensitivity of monotone functions that was pioneered by O'Donnell [O'D04].

This chapter is based on the paper "Optimal Cryptographic Hardness of Learning Monotone Functions", which is joint work with Dana Dachman-Soled, Tal Malkin, Rocco Servedio, Andrew Wan, and Hoeteck Wee, and appeared in the Proceedings of the 35th International Colloquium on Automata, Languages and Programming [DSLM$^+$08].

- **Chapter 4 – The Statistical Hardness of Learning Monotone Functions**

  Having shown the hardness of learning general polynomial-size monotone circuits, we focus on low-depth polynomial-size monotone circuits. This chapter complements the previous one by proving lower bounds that are *unconditional*, but which are only true for a restriction of the PAC learning model. In particular, we will be using the statistical query model of Kearns [Kea98] where instead of being given random examples, the learner is allowed queries of the form: "What is the approximate expected value of the function g on a random example labeled by the concept?"

  Unlike the conditional hardness results in PAC learning, there are unconditional information-theoretic barriers to statistical query learning. We show that the class of Boolean functions computed by polynomial-size depth-3 monotone circuits are hard to learn using statistical queries. The statistical query model encompasses almost all known learning algorithms, and our result implies that we will need new algorithmic techniques to be able to learn concept classes that are computable by polynomial-size depth-3 monotone circuits.

This chapter is based on a manuscript, which is joint work with Vitaly Feldman and Rocco Servedio.

- **Chapter 5 – Learning Random Monotone DNF**

  We move to even simpler circuits, polynomial-size depth-2 monotone circuits, otherwise known as monotone DNF formulas. The learnability of polynomial-size monotone DNF formulas is still unresolved, and the best algorithm to date only works for monotone DNF of sublinear size.

  We give an algorithm that with high probability properly learns random monotone $t(n)$-term DNF under the uniform distribution on the Boolean cube $\{0,1\}^n$. For any polynomially bounded function $t(n) \leq \text{poly}(n)$ the algorithm runs in time $\text{poly}(n, 1/\epsilon)$ and with high probability outputs an ϵ-accurate monotone DNF hypothesis. This is the first algorithm that can learn monotone DNF of arbitrary polynomial size in a reasonable average-case model of learning from random examples only.

  This chapter is based on the paper "Learning random monotone DNF", which is joint work with Jeffrey Jackson, Rocco Servedio, and Andrew Wan, and appeared in the Proceedings of the 11th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 12th International Workshop on Randomization and Computation [JLSW08].

- **Chapter 6 – The Structure of Monotone Decision Trees**

  Decision trees are one of the most important concept classes in the field of computational learning theory. They are perhaps the smallest concept class (the class of polynomial-size decision trees is a subset of the class of polynomial-size DNF formulas) for which no completely efficient algorithms exist, yet they are also widely used in experimental and applied machine learning. The heuristics used in machine learning (*e.g.,* the C4.5 and CART software packages [BFSO84, Qui93]) "grow" a tree from the root to its leaves by repeatedly replacing an existing leaf with a node labeled with a variable that minimizes the empirical error with respect to the given data sample.

Unfortunately, these heuristics have been difficult to analyze in the PAC model of learning. The best algorithm for PAC-learning decision trees is still the one by Ehrenfeucht and Haussler from 1989 [EH89], which learns size-s decision trees in time $O(n^{\log s})$. Even when the inputs are assumed to be distributed uniformly, finding an efficient algorithm for polynomial-size decision trees remains an open problem.

When considering polynomial-size decision trees that compute *monotone* functions (also called "monotone decision trees"), there is an efficient algorithm that can learn them to constant accuracy under the uniform distribution [OS07]. The resulting hypothesis, however, is a real-valued polynomial threshold function. Ideally, one would hope for an algorithm that learns polynomial-size monotone decision trees to arbitrary accuracy, under any distribution, and outputs a polynomial-size monotone decision tree as a hypothesis. As a first step to such a result, we show the relationship between the average depth of a monotone decision tree, its average sensitivity and its variance.

- **Chapter 7 – Teaching DNF in the Average Case**

  Finally, we return to monotone DNF formulas, and we show that they are *teachable* in the average case. *I.e.,* we study the average number of well-chosen labeled examples that are required for a helpful teacher to uniquely specify a target function within the concept class. This "average teaching dimension" has been studied in learning theory and combinatorics and is an attractive alternative to the "worst-case" teaching dimension of Goldman and Kearns [GK92] which is exponential for many interesting concept classes. Balbach [Bal05] showed that the classes of 1-decision lists and 2-term DNF each have linear average teaching dimension.

  As our main result, we extend Balbach's teaching result for 2-term DNF by showing that for any $1 \leq s \leq 2^{\Theta(n)}$, the well-studied concept classes of at-most-s-term DNF and at-most-s-term monotone DNF each have average teaching dimension $O(ns)$. The proofs use detailed analyses of the combinatorial structure of "most" DNF formulas and monotone DNF formulas. We also establish asymptotic separations between the worst-case and average teaching dimension for various other interesting Boolean concept classes such as juntas and sparse GF_2 polynomials.

This chapter is based on the paper "DNF are Teachable in the Average Case", which is joint work with Rocco Servedio, and Andrew Wan, and appeared in *Machine Learning* [LSW07]; a preliminary version of the paper appeared in the Proceedings of the 19th Annual Conference on Computational Learning Theory.

- **Chapter 8 – Conclusions and Future Directions**

  We close by proposing directions for future research.

Chapter 2

Background

In this chapter we go over the necessary background for the subsequent chapters. We introduce the main concept classes that we discuss in this thesis in Section 2.1. In Section 2.2 we recall Valiant's PAC model of learning. We provide some mathematical background in Section 2.3, and we conclude by highlighting some results on learning monotone Boolean functions in Section 2.4.

Throughout this proposal, we will write $\mathbf{1}_p$ for the indicator function that outputs 1 when the statement p is true, and 0 otherwise. As is standard in the literature, we write $[n]$ for the set $\{1, \ldots, n\}$, log to denote $\log_2$ and ln to denote the natural logarithm.

2.1 Concepts and Representations

Let X represent the domain under consideration. A *binary-classification concept* is any Boolean function $c : X \to \{0, 1\}$ over this domain. A *concept class* is a set $\mathcal{C}$ of concepts. As an example, X could be the set of all organisms, and $\mathcal{C}$ could be all the Linnaean taxonomic classes. (*E.g.,* [mammalia?] and [zygomycota?] are concepts in this class). Typically, however, X will be $\{0, 1\}^n$ for the concept classes we consider. A simple concept class over this domain is the class of *k-juntas*, which are the class of functions that depend on an unknown k out of the n variables.

We will also be restricting our attention to concept classes consisting of succinctly representable functions. As is usual in the theory of computation, succinctly will mean that the function has a representation of polynomial size (in n), and we will use the notation poly() to mean a fixed polynomial in the input variables. Thus, we will usually specify concept classes by their representations. Two that have received a lot of attention in the learning community are decision trees and disjunctive normal form formulas.

Let the input variables for the Boolean functions be $x_1, \ldots, x_n$, and let $\bar{x}_i$ denote the negation of x_i. We will call variables or their negations *literals*. We will often refer to a logical assignment of the variables as a string and vice-versa; thus, a string $x \in \{0,1\}^n$ corresponds to a truth-value assignment to the variables $x_1, \ldots, x_n$.

Decision trees are rooted binary trees where each internal node is labeled with a variable and each leaf is labeled with a bit in $\{0,1\}$. A decision tree computes a Boolean function in the obvious way: on input $x \in \{0,1\}^n$, if the variable at the root x_i is 0, the left child is evaluated, otherwise the right child is. This process is repeated until a leaf is reached, and the leaf's value is the function output (see Figure 2.1). A *disjunctive normal form (DNF)* formula is a disjunction of conjunctions of Boolean literals. One can view a DNF formula as a depth-2 unbounded-fan-in circuit that computes an OR of ANDS. We refer to the conjunctions as *terms* (see Figure 2.2). A DNF formula is said to be a read-k DNF formula if each literal occurs in at most k terms.

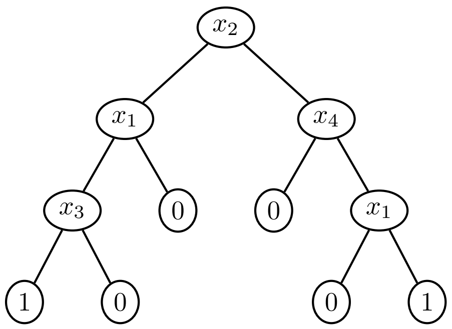

Figure 2.1: A decision tree of size 6.

Both representation schemes can represent any Boolean function. They also have natural complexity measures associated with them: the number of leaves for decision trees; and the number of terms for DNF formulas. Part of their attraction is that they seem to capture how humans naturally represent classifications (*e.g.,* [mammalia?] can be represented as [live birth?] and [sweat glands?] or [echidna?] or [platypus?]).

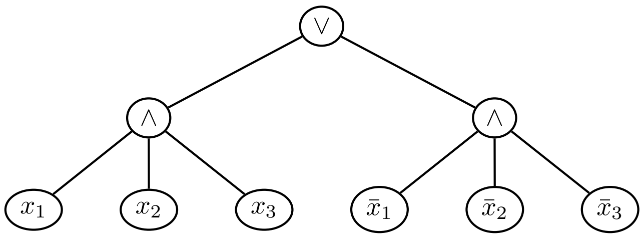

Figure 2.2: A DNF formula of size 2.

Now we can consider *representation classes* which are just concept classes specified by a representation scheme. A natural representation class is the class of functions representable by polynomial-sized decision trees. Another is the class of functions representable by polynomial-sized DNF formulas. It is important to be careful about distinguishing between the syntactic representations of these functions and the functions themselves. For instance, a function representable by a polynomial-sized decision tree has several equivalent polynomial-size decision tree representations. Thus, the number of polynomial-sized decision trees is much greater than the number of functions *representable* by polynomial-sized decision trees. When we refer to the *decision tree size* (or the *DNF formula size*) of a function we will always be referring to the size of the minimum size decision tree (or DNF formula) that computes the function.

One particular DNF formula we will encounter is the "tribes" function on $c \cdot d$ variables.

Definition 1. *The tribes function,* $\mathrm{Tribes}_d^c : \{0,1\}^{cd} \to \{0,1\}$*, is a size-*$c$*, read-once, DNF formula with terms of size* d*:*

$$\mathrm{Tribes}_d^c(x_1, \ldots, x_{cd}) \overset{def}{=} (x_1 \wedge \cdots \wedge x_d) \vee (x_{d+1} \wedge \cdots \wedge x_{2d}) \vee \cdots \vee (x_{(c-1)d+1} \wedge \cdots \wedge x_{cd}).$$

When $c = 2^d$*, we simply write* Tribes_d*.*

Circuits are another general representation class. A $\mathcal{G}$*-Boolean circuit* with n inputs is a directed acyclic graph, where the inputs to the circuit are the source vertices, and the other vertices are gates. The gates are labeled by functions $g \in \mathcal{G}$, and one of the gates is the output gate. The *size* of the circuit is the number of gates, not counting the input nodes, and the *depth* of the circuit is the length of the longest path from an input node to the output gate.

The class AC^0 consists of functions computable by polynomial-size constant-depth circuits comprised of AND, OR, and NOT gates of arbitrary fan-in. We can view DNF formulas as being depth-2 AC^0 circuits. Sometimes we will further refine AC^0 by specifying the depth of the circuits, and we write AC^0_d to denote the class of functions computable by polynomial-size, depth-d AND/OR/NOT-circuits. AC^1 is the class of functions computable by polynomial-size log-depth AND/OR/NOT-circuits, and in general AC^i is the class of functions computable by polynomial-size $\log^i$-depth AND/OR/NOT-circuits. We also define the general class $\mathsf{AC} := \cup_i \mathsf{AC}^i$.

The class NC refers to circuits with fan-in 2, and all the AC classes have corresponding NC versions. The class TC^0 consists of functions computable by polynomial-size, constant-depth circuits comprised of AND, OR, and NOT gates of arbitrary fan-in as well as threshold gates. A threshold gate returns 1 if at least half of its inputs are 1, and 0 otherwise. The following inclusions are known:

$$\mathsf{NC}^0 \subseteq \mathsf{AC}^0 \subseteq \mathsf{TC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{AC}^1.$$

Finally, we end with the concept class of parity functions over the domain $\{0,1\}^n$. There are 2^n different parity functions χ_S, for $S \subseteq [n]$. If the range is $\{0,1\}$, then $\chi_S(x) = \sum_{i \in S} x_i \bmod 2$. If the range is $\{+1,-1\}$, then $\chi_S(x) = \prod_{i \in S} x_i$.

2.2 PAC Learning

We begin this section by defining Valiant's PAC model of learning and we will follow with some of the variants that we will be considering in this thesis. For a more comprehensive introduction to computational learning theory, we refer the reader to the textbook by Kearns and Vazirani [KV94].

In Valiant's Probably Approximately Correct learning model [Val84] a learning algorithm for a concept class $\mathcal{C}$ tries to approximate an unknown *target concept* $c \in \mathcal{C}$ given *examples* of input/output pairs. The examples are generated by an *example oracle* $EX(c, \mathcal{D})$. When the oracle is invoked, it draws x from an unknown distribution $\mathcal{D}$, and gives the learner $(x, c(x))$. The value $c(x)$ is called a *label* and the pair $(x, c(x))$ is called a *labeled* *example.* The goal of the learner is to generate a hypothesis h under *any* representation scheme, and we say the *error rate* of h is $\Pr_{x \in \mathcal{D}}[h(x) \neq c(x)]$.

The learner is allowed to fail with probability δ and output a hypothesis with error rate up to ϵ. We allow the learner to fail since the algorithm could receive highly unrepresentative examples from the oracle, and we call δ the *confidence* parameter. On the other hand, even with a representative sample, the target concept might be impossible to learn exactly, and thus the hypothesis is allowed to be approximately correct.

We have the following formal definition of the PAC model in its full generality:

Definition 2. *Let $\mathcal{C}$ be a concept class over X. An algorithm A is said to be a* PAC learning algorithm for $\mathcal{C}$ *if for all $0 < \epsilon, \delta < 1$, for all $c \in \mathcal{C}$, and all distributions $\mathcal{D}$ over X, A on input ϵ, δ and access to the example oracle $EX(c, \mathcal{D})$ with probability at least $1 - \delta$ outputs a hypothesis h such that $\Pr_{x \in \mathcal{D}}[h(x) \neq c(x)] \leq \epsilon$.*

If A runs in time $\operatorname{poly}(n, \operatorname{size}(c), \epsilon^{-1}, \delta^{-1})$, we say that A is a *polynomial-time PAC learning algorithm.* A concept class is said to be *efficiently PAC learnable* if there exists a polynomial-time PAC learning algorithm for the class.

We note two relaxations of the PAC model now, and we will encounter others later on. The first is a relaxation known as *weak learning.* Definition 2 is referred to as *strong learning* since the learner is required to output a hypothesis of accuracy ϵ for any ϵ that it receives as an input. A weak-learning algorithm only has to output a hypothesis that does noticeably better than random, *i.e.*, $\Pr_{x \in \mathcal{D}}[h(x) \neq c(x)] \leq 1/2 - \gamma$, for $\gamma = 1/\operatorname{poly}(n, \operatorname{size}(c))$. We call γ the *advantage* or the *correlation.*

The second relaxation is the *uniform-distribution PAC-learning model.* Our original definition, Definition 2, is sometimes referred to as the *distribution-free PAC-learning model* since the learner is required to output a valid hypothesis for any unknown distribution. In the uniform-distribution PAC-learning model, however, the learning algorithm is given independent random examples $(x, c(x))$ where each x is drawn uniformly from the n-dimensional Boolean cube. Algorithms and hardness results in this framework have interesting connections with topics such as discrete Fourier analysis [Man94], circuit complexity [LMN93], noise sensitivity and the influence of variables in Boolean functions [KKL88, BKS99, KOS04, OS07], coding theory [FGKP07], privacy [BLR08, KLN+08], and cryptography [BFKL93, Kha95]. For these reasons, and because the model is natural and elegant in its own right, the uniform distribution learning model has been intensively studied for almost two decades.

2.3 Mathematical Background

We will mostly be dealing with the behavior of Boolean functions of the form $f : \{0,1\}^n \rightarrow \{0,1\}$, but sometimes it will be more convenient to use the range $\{+1,-1\}$, in which case -1 signifies TRUE and $+1$ signifies FALSE. We can also do an explicit conversion by setting $g : \{0,1\}^n \rightarrow \{+1,-1\}$ to be $g(x) = 1 - 2f(x)$.

Let $\mathcal{D}$ be a distribution on the input space. The functions on the domain $\{0,1\}^n$ form a 2^n-dimensional vector space with the following inner product:

$$\langle f_1, f_2 \rangle_{\mathcal{D}} := \sum_{x \in \{0,1\}^n} \mathcal{D}(x) f_1(x) f_2(x) = \mathop{\mathbb{E}}_{\mathcal{D}}[f_1(x) f_2(x)]$$

The *norm* of a function is then $||f|| := \sqrt{\langle f, f \rangle_{\mathcal{D}}}$. This is also called the L_2-norm, and more generally, for $p > 0$, $||f||_p = \mathbb{E}_{\mathcal{D}}[|f(x)|^p]^{1/p}$. The L_∞-norm is $||f||_\infty = \max_x |f(x)|$.

Note that $||f||_p$ is always 1 when the range is $\{+1,-1\}$. Also note that

$$\mathop{\mathbb{E}}_{\mathcal{D}}[f_1(x) f_2(x)] = \Pr_{\mathcal{D}}[f_1(x) = f_2(x)] - \Pr_{\mathcal{D}}[f_1(x) \neq f_2(x)].$$

Let $\epsilon = \Pr_{\mathcal{D}}[f_1(x) \neq f_2(x)]$, then $\langle f_1, f_2 \rangle_{\mathcal{D}} = 1 - 2\epsilon$ measures the correlation between f_1 and f_2.

For $x \in \{0,1\}^n$, x_i will denote the i-th bit of x. The *Hamming weight* of $x \in \{0,1\}^n$, *i.e.,* the number of 1's, will be denoted $|x|$. Let e_i denote the vector of Hamming weight 1 that is all 0's except at x_i. The inner product between two vectors $a, b \in \{0,1\}^n$ is defined as $\langle a, b \rangle := \sum_{i=1}^n a_i b_i \bmod 2 \in \{0,1\}$, and the inner product of a vector with itself is $||a|| := \langle a, a \rangle$. The bit-wise exclusive-or between them is denoted $a \oplus b \in \{0,1\}^n$. We will also often identify subsets of $[n]$ with vectors in $\{0,1\}^n$ and vice-versa ($a \in \{0,1\}^n \leftrightarrow a = \{i \in [n] : a_i = 1\}$). Thus we will denote the symmetric difference between two subsets $a, b \subseteq [n]$ as $a \oplus b$.

The *distance* between two vectors $a, b \in \{0,1\}^n$ is the number of bits that need to be flipped to go from one vector to the other, $\Delta(a,b) := |a \oplus b|$. We say that two vectors are *neighbors* if their distance is 1. We view $\{0,1\}^n$ as endowed with the natural partial order. For $x, y \in \{0,1\}^n$, we write $x \leq y$ if and only if $x_i \leq y_i$ for all $i = 1, \ldots, n$, and $x < y$ if $x \leq y$ and $x \neq y$. We will often consider the case where the distribution on inputs is uniform, in which case we will often leave $\mathcal{D}$ out of the notation.

2.3.1 The Fourier Transform

For a set $s \subseteq [n]$, we define $\chi_s : \{0,1\}^n \to \{0,1\}$ to be $\chi_s(x) = \sum_{i \in s} x_i \bmod 2$ (or $\chi_s(x) = \langle s, x\rangle$). If the range is $\{+1,-1\}$, then $\chi_s(x) = \prod_{i \in s}(-1)^{x_i}$ (or $\chi_s(x) = (-1)^{\langle s,x\rangle}$). These 2^n functions are the *parity* functions.

Note that $\mathbb{E}[\chi_\emptyset] = 1$, $\mathbb{E}[\chi_a] = 0$ for $a \neq \emptyset$, and $\chi_a \chi_b = \chi_{a \oplus b}$. It follows that the parity functions form an orthonormal basis for the space of functions $f : \{0,1\}^n \to \mathbb{R}$. The parity functions are thus also the *characters* of $\{0,1\}^n$, and they form a group $\mathbb{P}$. Since $\chi_a \chi_b = \chi_{a \oplus b}$, there is a natural isomorphism between $\{0,1\}^n$ and $\mathbb{P}$, and thus $\chi_a(b) = \chi_b(a)$.

If $f : \{0,1\}^n \to \{+1,-1\}$ is any function, we define its *Fourier transform* $\hat{f} : \mathbb{P} \to \mathbb{R}$ by

$$\hat{f}(\chi_a) := \langle f, \chi_a \rangle = \mathop{\mathbb{E}}_{x}[f(x)\chi_a(x)],$$

and due to the isomorphism with $\{0,1\}^n$ we can also view it as a function that maps $\{0,1\}^n$ to $[-1,+1]$ and define $\hat{f}(a) := \hat{f}(\chi_a)$. When working on $\mathbb{P}$, we will use the *counting measure* which assigns weight 1 to each $a \in \mathbb{P}$. Thus expectations and norms are taken with respect to the appropriate measure:

$$\langle \hat{f}, \hat{g} \rangle = \mathop{\mathbb{E}}_{a \in \{0,1\}^n}[\hat{f}(a)\hat{g}(a)] := \sum_{a \in \{0,1\}^n} \hat{f}(a)\hat{g}(a),$$

$$\|\hat{f}\|_p := \left(\sum_{a \in \{0,1\}^n} |\hat{f}(a)|^p \right)^{1/p}.$$

However, we will favor the sum notation when taking the expectation over $\mathbb{P}$ to prevent any confusion due to the isomorphism with $\{0,1\}^n$. We will also use lower case, script letters for the norms to emphasize the difference (ℓ_p instead of L_p).

The Fourier inversion theorem says that every real function f over $\{0,1\}^n$ can also be expressed as a linear combination of characters:

$$f(x) = \sum_{a \in \{0,1\}^n} \hat{f}(a)\chi_a(x)$$

where $\hat{f}(a)$, the coefficient of χ_a, is called a *Fourier coefficient*. Note that $\hat{f}(\emptyset) = \mathbb{E}[f]$.

Theorem 3 (Plancherel's Theorem)**.** *For any $f, g : \{0,1\}^n \to \{+1,-1\}$, $\langle f, g\rangle = \langle \hat{f}, \hat{g}\rangle$. In particular, if $g = f$, then $\langle f, f\rangle = \langle \hat{f}, \hat{f}\rangle$. (This is Parseval's equality.)*

Proof.

$$\begin{aligned}\langle f, g\rangle &= \left\langle \sum_{a\in\{0,1\}^n} \hat{f}(a)\chi_a, \sum_{b\in\{0,1\}^n} \hat{g}(b)\chi_b \right\rangle = \sum_{a,b\in\{0,1\}^n} \hat{f}(a)\hat{g}(b)\langle \chi_a, \chi_b\rangle \\ &= \sum_{a\in\{0,1\}^n} \hat{f}(a)\hat{g}(a) = \langle \hat{f}, \hat{g}\rangle.\end{aligned}$$

We can also write this as $\mathbb{E}_x[f(x)g(x)] = \sum_a \hat{f}(a)\hat{g}(a)$. ∎

Parseval's equality has a nice consequence for Boolean functions: $\mathbb{E}_{a\in\{0,1\}^n} \hat{f}(a)^2 = \langle \hat{f}, \hat{f}\rangle = \langle f, f\rangle = 1$. It should be noted that the Fourier transform of Boolean functions is sometimes called the Walsh-Hadamard transform.

2.3.2 Tail Bounds

We will often need to bound the probability that a random variable will deviate from its expectation.

Fact 4 (Markov's Inequality)**.** *Let X be a non-negative random variable. Then,*

$$\Pr[X > \lambda] < \frac{\mathbb{E}[x]}{\lambda},$$

for any $\lambda > 0$.

When bounding the probability that a sum of random variables will deviate from its expected value, we have the following two bounds due to Chernoff and Hoeffding.

Fact 5 ([Che52])**.** *Let $X_1, \ldots, X_t$ be binary 0/1 independent random variables with $0 < \mathbb{E}[X_i] < 1$ for all i. Let $X = \frac{1}{t}\sum_{i=1}^t X_i$ be the empirical average. Then for any $0 < \epsilon < 1$,*

$$\Pr[X \geq (1+\epsilon)\,\mathbb{E}[X]] < e^{-\frac{\mathbb{E}[X]\epsilon^2}{3}},$$

and

$$\Pr[X \leq (1-\epsilon)\,\mathbb{E}[X]] < e^{-\frac{\mathbb{E}[X]\epsilon^2}{2}}.$$

Fact 6 ([Hoe63]). *Let $X_1, \dots, X_t$ be independent random variables such that $a \leq X_i \leq b$ for all i. Let $X = \frac{1}{t}\sum_{i=1}^{t} X_i$ be the empirical average. Then for any $\epsilon > 0$,*

$$\Pr[X \geq \mathbb{E}[X] + \epsilon] \leq e^{-\frac{2t\epsilon^2}{(b-a)^2}},$$

and

$$\Pr[X \leq \mathbb{E}[X] - \epsilon] \leq e^{-\frac{2t\epsilon^2}{(b-a)^2}}.$$

2.4 Learning Monotone Functions

Valiant introduced the Probably Approximately Correct (PAC) model of learning Boolean functions from random examples more than two decades ago [Val84]. Since that time a great deal of research effort has been expended on trying to understand the inherent abilities and limitations of computationally efficient learning algorithms. For many classes of functions, uniform distribution learning algorithms have been devised which substantially improve on a naive exponential-time approach to learning via brute-force search. However, despite intense efforts, researchers have not yet been able to obtain polynomial-time learning algorithms in this model for various simple classes of functions. Interestingly, in many of these cases restricting the class of functions to the corresponding class of *monotone* functions has led to more efficient – sometimes polynomial-time – algorithms.

Monotone functions are non-decreasing functions. A Boolean function f is *monotone* if $x \leq y$ implies $f(x) \leq f(y)$. Monotone functions seem to be much easier to learn than general functions. If the concept class is the class of all Boolean functions, a polynomial-time learner cannot do any better than random guessing, but if the concept class is the class of all *monotone* Boolean functions, then a fairly simple learner can output a hypothesis that weakly learns the target concept [BBL98, OW09].

If we view the domain of $\{0,1\}^n$ as a hypercube, monotone functions are intuitively easier since they split the hypercube into two connected halves, and the learner only needs to figure out where the separation is. The slice functions are an extreme example of this.

For $g : \{0,1\}^n \to \{0,1\}$, we write slice_g to denote the "middle slice" function:

$$\text{slice}_g(x) = \begin{cases} 1 & \text{if } |x| > \lfloor k/2 \rfloor \\ g(x) & \text{if } |x| = \lfloor k/2 \rfloor \\ 0 & \text{if } |x| < \lfloor k/2 \rfloor. \end{cases}$$

It is immediate that slice_g is a monotone Boolean function for any function g.

There have been several cases where restricting our attention to monotone functions has lead to more efficient algorithms. Some examples include:

1. To date, the best algorithms for learning arbitrary k-juntas, runs in time $n^{.704k}$ [MOS03]. However, a simple algorithm learns monotone $O(\log n)$-juntas to perfect accuracy in $\text{poly}(n)$ time, and a more complex algorithm [BT06] learns monotone $\tilde{O}(\log^2(n))$-juntas to any constant accuracy in $\text{poly}(n)$ time.

2. The fastest known algorithm for learning $\text{poly}(n)$-size general decision trees to constant accuracy takes $n^{O(\log n)}$ time (this follows from [EH89, Ver90]), but $\text{poly}(n)$-size decision trees that compute monotone functions (called "monotone decision trees") can be learned to accuracy ϵ in $O(n^{1/\epsilon^2})$ time [OS07] (which is polynomial-time for constant ϵ).

3. The fastest known uniform distribution learning algorithm for the general class of s-term DNF formulas, due to Verbeurgt [Ver90], runs in time $n^{O(\log s)}$ to learn to any constant accuracy. In contrast, for s-term monotone DNF formulas, Servedio [Ser04] gives an algorithm that runs in $s^{O(\log s)}$ time. Thus the class of $2^{O(\sqrt{\log n})}$-term monotone DNF formulas can be learned to any constant accuracy in $\text{poly}(n)$ time, but no such result is known for $2^{O(\sqrt{\log n})}$-term general DNF formulas.

4. No polynomial-time algorithm can learn the general class of all Boolean functions on $\{0,1\}^n$ to accuracy better than $\frac{1}{2} + \frac{\text{poly}(n)}{2^n}$, but a simple polynomial-time algorithm can learn the class of all monotone Boolean functions to accuracy $\frac{1}{2} + \frac{\Omega(\log n)}{\sqrt{n}}$ [OW09]. We note also that the result of [BT06] mentioned above follows from a $2^{\tilde{O}(\sqrt{n})}$-time algorithm for learning arbitrary monotone functions on n variables to constant accu-

racy (it is easy to see that no comparable algorithm can exist for learning arbitrary Boolean functions to constant accuracy).

Even given these results, our knowledge of the learnability of succinctly represented monotone Boolean functions remains woefully incomplete. The most outstanding question is considered to be the learnability of polynomial-size monotone DNF formulas [BT06, HM91, KMSP94, Ver98]. This thesis presents the best algorithm for learning polynomial-size monotone DNF formulas in the average-case to date, and it helps complete the picture by showing that several classes of succinctly representable monotone functions are hard to learn.

Chapter 3

The Cryptographic Hardness of Learning Monotone Functions

In this chapter we addresses the discrepancy between known positive and negative results for uniform distribution learning by establishing strong computational hardness results for learning various classes of *monotone* functions.

3.1 Introduction

Essentially all known representation-independent hardness of learning results (*i.e.,* results that apply to learning algorithms that do not have any restrictions on the syntactic form of the hypotheses they output) rely on some cryptographic assumption, or an assumption that easily implies a cryptographic primitive. For example, under the assumption that certain subset sum problems are hard on average, Kharitonov [Kha95] showed that the class AC^1 is hard to learn under the uniform distribution. Subsequently Kharitonov [Kha93] showed that if factoring Blum integers is 2^{n^ϵ}-hard for some fixed $\epsilon > 0$, then even the class AC^0 similarly cannot be learned in polynomial time under the uniform distribution. In later work, Naor and Reingold [NR04] gave constructions of pseudorandom functions with very low circuit complexity; their results imply that if factoring Blum integers is super-polynomially hard, then even depth-5 TC^0 circuits cannot be learned in polynomial time under the uniform distribution. We note that all of these hardness results apply even to algorithms which may make black-box "membership queries" to obtain the value $f(x)$ for inputs x of their choosing.

Monotonicity versus cryptography. Given that cryptography precludes efficient learning while monotonicity seems to make efficient learning easier, it is natural to investigate how these phenomena interact. One could argue that prior to the current work there was something of a mismatch between known positive and negative results for uniform-distribution learning: as described in Section 2.4 a fairly broad range of polynomial-time learning results had been obtained for various classes of monotone functions, but there were no corresponding computational hardness results for monotone functions. Can all monotone Boolean functions computed by polynomial-size circuits be learned to 99% accuracy in polynomial time from uniform random examples? Prior to our work answers were not known even to such seemingly basic questions about learning monotone functions as this one. This gap in understanding motivated the research presented in this chapter (which, as we describe below, lets us answer "no" to the above question in a strong sense).

3.1.1 Our Results and Techniques: Cryptography Trumps Monotonicity.

We present several different constructions of "simple" (polynomial-time computable) monotone Boolean functions and prove that these functions are hard to learn under the uniform distribution, even if membership queries are allowed. We now describe our main results, followed by a high-level description of how we obtain them.

Blum, Burch, and Langford (henceforth referred to as BBL) [BBL98] showed that arbitrary monotone functions cannot be learned to accuracy better than $\frac{1}{2} + \frac{O(\log n)}{\sqrt{n}}$ by any algorithm which makes poly(n) many membership queries. This is a non-explicit bound which is proved using randomly generated monotone DNF formulas of size (roughly) $n^{\log n}$. A natural goal is to obtain *explicit* lower bounds for learning polynomial-time-computable monotone functions which match, or nearly match, this level of hardness (which is optimal by the $(\frac{1}{2} + \frac{\log n}{\sqrt{n}})$-accuracy algorithm of O'Donnell and Wimmer [OW09] as stated in Section 2.4). We prove near-optimal hardness for learning polynomial-size monotone circuits:

Theorem 7 (informal)**.** *If one-way functions exist, then there is a class of* poly(n)*-size monotone circuits that cannot be learned to accuracy* $\frac{1}{2} + \frac{1}{n^{1/2-\epsilon}}$ *for any fixed* $\epsilon > 0$.

Our approach yields even stronger lower bounds if we make stronger assumptions:

- Assuming the existence of sub-exponential one-way functions, we improve the bound on the accuracy to $1/2 + \mathrm{polylog}(n)/n^{1/2}$.
- Assuming the hardness of factoring Blum integers, our hard-to-learn functions may be computed in monotone NC^1.
- Assuming that Blum integers are 2^{n^ϵ}-hard to factor on average (the same hardness assumption used in Kharitonov's work [Kha93]), we obtain a lower bound for learning constant-depth circuits of sub-polynomial size that almost matches the positive result by Servedio [Ser04]. More precisely, we show that for any (sufficiently large) constant d, the class of monotone functions computed by depth-d AND/OR/NOT circuits of size $2^{(\log n)^{O(1)/(d+1)}}$ cannot be learned to accuracy 51% under the uniform distribution in poly(n) time. In contrast, the positive result by Servedio shows that monotone functions computed by depth-d AND/OR/NOT circuits of size $2^{O((\log n)^{1/(d+1)})}$ can be learned to any constant accuracy in poly(n) time.

These results are summarized in Figure 3.1.1.

Proof techniques. A natural first approach is to try to "pseudorandomize" BBL's construction of random $n^{\log n}$-term monotone DNF formulas. We were not able to do this directly; indeed, as we discuss in Chapter 8, pseudorandomizing the BBL construction seems closely related to an open problem of Goldreich *et al.* [GGN03]. However, it turns out that a closely related approach does yield some results along the desired lines; in Section 3.4 we present and analyze a simple variant of the BBL information-theoretic construction and then show how to "pseudorandomize" the variant. Since our variant gives a weaker quantitative bound on the information-theoretic hardness of learning than BBL, this gives a construction of polynomial-time-computable monotone functions which, assuming the existence of one-way functions, cannot be learned to accuracy $\frac{1}{2} + \frac{1}{\mathrm{polylog}(n)}$ under the uniform distribution. While this answers the question posed above (even with "51%" in place of

Hardness assumption	**Complexity of f**	**Accuracy bound**	**Ref.**
none	random $n^{\log n}$-term monotone DNF	$\frac{1}{2} + \frac{\omega(\log n)}{n^{1/2}}$	[BBL98]
OWF (poly)	poly(n)-size monotone circuits	$\frac{1}{2} + \frac{1}{n^{1/2-\epsilon}}$	Thm. 7
OWF $(2^{n^{\alpha}})$	poly(n)-size monotone circuits	$\frac{1}{2} + \frac{\text{poly}(\log n)}{n^{1/2}}$	Thm. 17
factoring BI (poly)	monotone NC^1-circuits	$\frac{1}{2} + \frac{1}{n^{1/2-\epsilon}}$	Thm. 19
factoring BI $(2^{n^{\alpha}})$	depth-d, $2^{(\log n)^{O(1)/(d+1)}}$-size AND/OR/NOT circuits	$\frac{1}{2} + o(1)$	Thm. 22

Figure 3.1: Summary of known hardness results for learning monotone Boolean functions. The meaning of each row is as follows: under the stated hardness assumption, there is a class of monotone functions computed by circuits of the stated complexity which no poly(n)-time membership query algorithm can learn to the stated accuracy. In the first column, OWF and BI denote one-way functions and Blum Integers respectively, and "poly" and "$2^{n^{\alpha}}$" means that the problems are intractable for poly(n)- and $2^{n^{\alpha}}$-time algorithms respectively (for some fixed $\alpha > 0$). Recall that the poly(n)-time algorithm of O'Donnell and Wimmer [OW09] for learning monotone functions implies that the best possible accuracy bound for monotone functions is $\frac{1}{2} + \frac{\Omega(\log n)}{n^{1/2}}$.

"99%"), the $\frac{1}{\text{polylog}(n)}$ factor is rather far from the $\frac{O(\log n)}{\sqrt{n}}$ factor that one might hope for as described above.

In Section 3.2 we use a different construction to obtain much stronger quantitative results; another advantage of this second construction is that it enables us to show hardness of learning *monotone circuits* rather than just circuits computing monotone functions. We start with the simple observation that using standard tools it is easy to construct polynomial-size monotone circuits computing slice functions that are pseudorandom on the middle layer of the Boolean cube $\{0,1\}^n$. Such functions are easily seen to be mildly hard to learn, *i.e.*, hard to learn to accuracy $1-\frac{\Omega(1)}{\sqrt{n}}$. We then use the elegant machinery of hardness amplification of monotone functions which was pioneered by O'Donnell [O'D04] to amplify the hardness of this construction to near-optimal levels (rows 2–4 of Figure 3.1.1). We obtain our result for constant depth, sub-polynomial-size circuits (row 5 of Figure 3.1.1) by augmenting this approach with an argument which at a high level is similar to one used by Allender *et al.* [AHM$^+$06], by "scaling down" and modifying our hard-to-learn functions using a variant of Nepomnjaščiĭ's theorem [Nep70].

3.1.2 Preliminaries

As described earlier, all of our hardness results apply even to learning algorithms which may make *membership queries*, *i.e.,* black-box queries to an oracle which gives the label $f(x)$ of any example $x \in \{0,1\}^n$ on which it is queried. It is clear that for learning with respect to the uniform distribution, having membership query access to the target function f is at least as powerful as being given uniform random examples labeled according to x since the learner can simply generate uniform random strings for herself and query the oracle to simulate a random example oracle.

The goal of the learning algorithm is to construct a hypothesis h so that $\Pr_x[h(x) \neq f(x)]$ is small, where the probability is taken over the uniform distribution. We shall only consider learning algorithms that are allowed to run in poly(n) time, so the learning algorithm L may be viewed as an oracle probabilistic polynomial-time (p.p.t.) machine which is given black-box access to the function f and attempts to output a hypothesis h with small error relative to f.

We establish that a class $\mathcal{C}$ of functions is hard to learn by showing that for a uniform random $f \in \mathcal{C}$, the expected error of any poly(n)-time learning algorithm L is close to 1/2 when run with f as the target function. Thus we bound the quantity

$$\Pr_{f\in\mathcal{C},x\in\{0,1\}^n}[L^f(1^n)\to h; h(x) = f(x)] \tag{3.1}$$

where the probability is also taken over any internal randomization of the learning algorithm L. We say that *class $\mathcal{C}$ is hard to learn to accuracy* $\frac{1}{2} + \epsilon(n)$ if for every poly(n)-time membership query learning algorithm L (*i.e.,* p.p.t. oracle algorithm), we have (3.1) $<$ $\frac{1}{2} + \epsilon(n)$ for all sufficiently large n. As noted by BBL, it is straightforward to transform a lower bound of this sort into a lower bound for the usual ϵ, δ formulation of PAC learning.

Circuit complexity. A circuit is said to be monotone if it is composed entirely of AND/OR gates with no negations. Every monotone circuit computes a monotone Boolean function, but of course non-monotone circuits may compute monotone functions as well. The famous result of Razborov [Raz85] shows that there are natural monotone Boolean functions (such as the perfect matching function) which can be computed by polynomial-size circuits but cannot be computed by polynomial-size monotone circuits. Thus, in general, it is a stronger result to show that a function can be computed by a small monotone circuit than to show that it is monotone and can be computed by a small circuit.

Pseudorandom functions. Pseudorandom functions [GGM86] are the main cryptographic primitive that underlie our constructions. For a fixed $k(n) \leq n$, let $\mathcal{G}$ be a family of functions $\{g : \{0,1\}^{k(n)} \rightarrow \{0,1\}\}$ each of which is computable by a circuit of size $\mathrm{poly}(k(n))$. We say that $\mathcal{G}$ is a $t(n)$*-secure pseudorandom function family* if the following condition holds: for any probabilistic $t(n)$-time oracle algorithm A, we have

$$\left| \Pr_{g \in \mathcal{G}}[A^{g}(1^n) \text{ outputs } 1] - \Pr_{g' \in \mathcal{G}'}[A^{g'}(1^n) \text{ outputs } 1] \right| \leq 1/t(n)$$

where $\mathcal{G}'$ is the class of all $2^{2^{k(n)}}$ functions from $\{0,1\}^{k(n)}$ to $\{0,1\}$ (so the second probability above is taken over the choice of a truly random function g'). Note that the purported distinguisher A has oracle access to a function on $k(n)$ bits but is allowed to run in time $t(n)$.

It is well known that a pseudorandom function family that is $t(n)$-secure for all polynomials $t(n)$ can be constructed from any one-way function [GGM86, HILL99]. We shall use the following folklore quantitative variant which relates the hardness of the one-way function to the security of the resulting pseudorandom function:

Proposition 8. *Fix $t(n) \geq \mathrm{poly}(n)$ and suppose there exist one-way functions that are hard to invert on average for $t(n)$-time adversaries. Then there exists a constant, $0 < c < 1$, such that for any $k(n) \leq n$, there is a pseudorandom family $\mathcal{G}$ of functions $\{g : \{0,1\}^{k(n)} \rightarrow \{0,1\}\}$ that is $(t(k(n)))^c$-secure.*

3.2 Lower Bounds via Hardness Amplification of Monotone Functions

In this section we prove our main hardness results, summarized in Figure 3.1.1, for learning various classes of monotone functions under the uniform distribution with membership queries.

Let us start with a high-level explanation of the overall idea. Inspired by the work on hardness amplification within NP initiated by O'Donnell [O'D04, Tre03, HVV04], we study constructions of the form

$$f(x_1, \ldots, x_m) = C(f'(x_1), \ldots, f'(x_m))$$

where C is a Boolean "combining function" with low noise stability (we give precise definitions later) which is both *efficiently computable* and *monotone*. Recall that O'Donnell showed that if f' is weakly hard to compute and the combining function C has low noise stability, then f is very hard to compute. This result holds for general (not necessarily monotone) functions C, and thus generalizes Yao's XOR lemma, which addresses the case where C is the XOR of m bits (and hence has the lowest noise stability of all Boolean functions, see [O'D04]).

Roughly speaking, we establish an analogue of O'Donnell's result for learning. Our analogue, essentially states that for certain well-structured[1] functions f' that are hard to learn to high accuracy, if C has low noise stability then f is hard to learn to accuracy even slightly better than $1/2$. Since our ultimate goal is to establish that "simple" classes of monotone functions are hard to learn, we shall use this result with combining functions C that are computed by "simple" monotone Boolean circuits. In order for the overall function f to be monotone and efficiently computable, we need the initial f' to be well-structured, monotone, efficiently computable, and hard to learn to high accuracy. Such functions are easily obtained by a slight extension of an observation of Kearns *et al.* [KLV94]. They noted that the middle slice f' of a random Boolean function on $\{0,1\}^k$ is hard to learn to accuracy greater than $1 - \Theta(1/\sqrt{k})$ [BBL98, KLV94]; by taking the middle slice of a

[1]As will be clear from the proof, we require that f' be balanced and have a "hard-core set."

pseudorandom function instead, we obtain an f' with the desired properties. In fact, by a result of Berkowitz [Ber82] this slice function is computable by a polynomial-size monotone circuit, so the overall hard-to-learn functions we construct are computed by polynomial-size monotone circuits.

Organization. First we adapt the analysis in [O'D04, Tre03, HVV04] to reduce the problem of constructing hard-to-learn monotone Boolean functions to constructing monotone combining functions C with low noise stability. Then we show how constructions and analyses from [O'D04, MO03] can be used to obtain a "simple" monotone combining function with low noise stability. We establish Theorems 16 and 17 (lines 2 and 3 of Figure 3.1.1) by making different assumptions about the hardness of the initial pseudorandom functions. Finally, in Section 3.3 we establish Theorems 19 and 22 by making specific number theoretic assumptions (namely, the hardness of factoring Blum integers) to obtain hard-to-learn monotone Boolean functions that can be computed by very simple circuits.

3.2.1 Hardness Amplification for Learning

Let $C : \{0,1\}^m \to \{0,1\}$ and $f' : \{0,1\}^k \to \{0,1\}$ be Boolean functions. We write $C \circ f'^{\otimes m}$ to denote the Boolean function over $(\{0,1\}^k)^m$ given by:

$$C \circ f'^{\otimes m}(x) = C(f'(x_1), \ldots, f'(x_m)), \qquad \text{where } x = (x_1, \ldots, x_m).$$

Following the analysis in [O'D04, Tre03, HVV04], we shall study the bias and noise stability of various Boolean functions. Specifically, we adopt the following notations and definitions from [HVV04]. The *bias* of a Boolean random variable X is defined to be

$$\mathrm{Bias}[X] := |\Pr[X = \textsc{true}] - \Pr[X = \textsc{false}]|.$$

We can view the output of a probabilistic Boolean function h over $\{0,1\}^n$ as a probability distribution over Boolean functions on $\{0,1\}^n$. The *expected bias* of a probabilistic Boolean function h is:

$$\mathrm{ExpBias}[h] := \mathop{\mathbb{E}}_{x}[\mathrm{Bias}[h(x)]].$$

Let C be a Boolean function and $0 \leq \delta \leq \frac{1}{2}$. The *noise stability of f at noise rate δ*, denoted $\mathrm{NoiseStab}_\delta[f]$, is defined to be

$$\mathrm{NoiseStab}_\delta[C] \stackrel{\mathrm{def}}{=} \mathrm{E}_{x,\eta}[C(x) \oplus C(x \oplus \eta)] = 2 \cdot \Pr_{x,\eta}[C(x) = C(x \oplus \eta)] - 1$$

where $x \in \{0,1\}^n$ is uniform random, $\eta \in \{0,1\}^n$ is a vector whose bits are each independently 1 with probability δ.

Throughout this subsection we write m for $m(n)$ and k for $k(n)$. We shall establish the following:

Lemma 9. *Let $C : \{0,1\}^m \rightarrow \{0,1\}$ be a polynomial-time computable function. Let $\mathcal{G}'$ be the family of all 2^{2^k} functions from $\{0,1\}^k$ to $\{0,1\}$, where $n = mk$ and $k = \omega(\log n)$. Then the class $\mathcal{C} = \{f = C \circ \mathrm{slice}_g^{\otimes m} \mid g \in \mathcal{G}'\}$ of Boolean functions over $\{0,1\}^n$ is hard to learn to accuracy*

$$\frac{1}{2} + \frac{1}{2}\sqrt{\mathrm{NoiseStab}_{\Theta(1/\sqrt{k})}[C]} + \frac{1}{n^{\omega(1)}}.$$

This easily yields Corollary 11, which is an analogue of Lemma 9 with pseudorandom rather than truly random functions, and which we use to obtain our main hardness of learning results.

Proof of Lemma 9: Let k, m be such that $mk = n$, and let $C : \{0,1\}^m \rightarrow \{0,1\}$ be a Boolean combining function. We prove the lemma by upper bounding

$$\Pr_{g \in \mathcal{G}', x \in \{0,1\}^n}\left[L^f(1^n) \rightarrow h;\ h(x) = f(x)\right] \tag{3.2}$$

where L is an arbitrary p.p.t. oracle machine (running in time $\mathrm{poly}(n)$ on input 1^n) that is given oracle access to $f \stackrel{\mathrm{def}}{=} C \circ \mathrm{slice}_g^{\otimes m}$ and outputs some hypothesis $h : \{0,1\}^n \rightarrow \{0,1\}$.

We first observe that since C is computed by a uniform family of circuits of size $\mathrm{poly}(m) \leq \mathrm{poly}(n)$, it is easy for a $\mathrm{poly}(n)$-time machine to simulate oracle access to f if it is given oracle access to g. So (3.2) is at most

$$\Pr_{g \in \mathcal{G}',\ x \in \{0,1\}^n}\left[L^g(1^n) \rightarrow h;\ h(x) = (C \circ \mathrm{slice}_g^{\otimes m})(x)\right]. \tag{3.3}$$

To analyze the above probability, suppose that in the course of its execution L never queries g on any of the inputs $x_1, \ldots, x_m \in \{0,1\}^k$, where $x = (x_1, \ldots, x_m)$. Then the *a posteriori* distribution of $g(x_1), \ldots, g(x_m)$ (for uniform random $g \in \mathcal{G}'$) given the responses to L's queries that it received from g is identical to the distribution of $g'(x_1), \ldots, g'(x_m)$, where g' is an independent uniform draw from $\mathcal{G}'$: both distributions are uniform random over $\{0,1\}^m$. (Intuitively, this just means that if L never queries the random function g on any of $x_1, \ldots, x_m$, then giving L oracle access to g does not help it predict the value of f on $x = (x_1, \ldots, x_m)$.) Since L runs in poly(n) time, for any fixed $x_1, \ldots, x_m$ the probability that L queried g on any of $x_1, \ldots, x_m$ is at most $\frac{m \cdot \mathrm{poly}(n)}{2^k}$. Hence (3.3) is bounded by

$$\Pr_{g,g' \in \mathcal{G}',\, x \in \{0,1\}^n}\left[L^g(1^n) \to h;\ h(x) = (C \circ \mathrm{slice}_{g'}^{\otimes m})(x)\right] + \frac{m \cdot \mathrm{poly}(n)}{2^k}. \tag{3.4}$$

The first summand in (3.4) is the probability that L correctly predicts the value $C \circ \mathrm{slice}_{g'}^{\otimes m}(x)$, given oracle access to g, where g and g' are independently random functions and x is uniform over $\{0,1\}^n$. It is clear that the best possible strategy for L is to use a maximum likelihood algorithm, *i.e.*, predict according to the function h which, for any fixed input x, outputs 1 if and only if the random variable $(C \circ \mathrm{slice}_{g'}^{\otimes m})(x)$ (we emphasize that the randomness here is over the choice of g') is biased towards 1. The expected accuracy of this h is precisely

$$\frac{1}{2} + \frac{1}{2}\,\mathrm{ExpBias}[C \circ \mathrm{slice}_{g'}^{\otimes m}]. \tag{3.5}$$

Now fix $\delta \stackrel{\text{def}}{=} \binom{k}{\lfloor k/2 \rfloor}/2^k = \Theta(1/\sqrt{k})$ to be the fraction of inputs in the "middle slice" of $\{0,1\}^k$. We observe that the probabilistic function $\mathrm{slice}_{g'}$ (where g' is truly random) is "δ-random" in the sense of ([HVV04], Definition 3.1), meaning that it is balanced, truly random on inputs in the middle slice, and deterministic on all other inputs. This means that we may apply the following technical lemma (Lemma 3.7 from [HVV04], see also [O'D04]):

Lemma 10. *Let $h : \{0,1\}^n \to \{0,1\}$ be a function that is δ-random. Then*

$$\mathrm{ExpBias}[C \circ h^{\otimes m}] \leq \sqrt{\mathrm{NoiseStab}_\delta[C]}.$$

Applying this lemma to the function $\mathrm{slice}_{g'}$ we obtain:

$$\mathrm{ExpBias}[C \circ \mathrm{slice}_{g'}^{\otimes m}] \leq \sqrt{\mathrm{NoiseStab}_\delta[C]}. \tag{3.6}$$

Combining (3.4), (3.5) and (3.6) and recalling that $k = \omega(\log n)$, we obtain Lemma 9.

Corollary 11. *Let $C : \{0,1\}^m \to \{0,1\}$ be a polynomial-time computable function. Let $\mathcal{G}$ be a pseudorandom family of functions from $\{0,1\}^k$ to $\{0,1\}$ that are secure against $\operatorname{poly}(n)$-time adversaries, where $n = mk$ and $k = \omega(\log n)$. Then the class $\mathcal{C} = \{f = C \circ \operatorname{slice}_g^{\otimes m} \mid g \in \mathcal{G}\}$ of Boolean functions over $\{0,1\}^n$ is hard to learn to accuracy*

$$\frac{1}{2} + \frac{1}{2}\sqrt{\operatorname{NoiseStab}_{\Theta(1/\sqrt{k})}[C]} + \frac{1}{n^{\omega(1)}}.$$

Proof. The corollary follows from the fact that (3.3) must differ from its pseudorandom counterpart,

$$\Pr_{g \in \mathcal{G},\, x \in \{0,1\}^n}\Big[L^g(1^n) \to h;\ h(x) = (C \circ \operatorname{slice}_g^{\otimes m})(x)\Big], \tag{3.7}$$

by less than any fixed $1/\operatorname{poly}(n)$. Otherwise, we would easily obtain a $\operatorname{poly}(n)$-time distinguisher that, given oracle access to g, runs L to obtain a hypothesis h and checks whether $h(x) = (C \circ \operatorname{slice}_g^{\otimes m})(x)$ for a random x to determine whether g is drawn from $\mathcal{G}$ or $\mathcal{G}'$. ■

By instantiating Corollary 11 with a "simple" monotone function C having low noise stability, we obtain strong hardness results for learning simple monotone functions. We exhibit such a function C in the next section.

3.2.2 A Simple Monotone Combining Function

In this section we combine known results of [O'D04, MO03] to obtain:

Proposition 12. *Given a value k, let $m = 3^\ell d 2^d$ for ℓ, d satisfying $3^\ell \le k^6 < 3^{\ell+1}$ and $d \le O(k^{.35})$. Then there exists a monotone function $C : \{0,1\}^m \to \{0,1\}$ computed by a uniform family of $\operatorname{poly}(m)$-size, $\log(m)$-depth monotone circuits such that*

$$\operatorname{NoiseStab}_{\Theta(1/\sqrt{k})}[C] = O\Big(\frac{k^6 \log m}{m}\Big). \tag{3.8}$$

Note that in this proposition we may have m as large as $2^{\Theta(k^{.35})}$ but not larger. O'Donnell [O'D04] gave a lower bound of $\Omega(\frac{\log^2 m}{m})$ on $\operatorname{NoiseStab}_{\Theta(1/\sqrt{k})}[C]$ for every monotone m-variable function C, so the above upper bound is fairly close to the best possible (within a $\operatorname{polylog}(m)$ factor if $m = 2^{k^{\Theta(1)}}$).

Following [O'D04, HVV04], we use the "recursive majority of 3" function and the tribes function (see Definition 1) in our construction. We require the following results on noise stability:

Lemma 13 ([O'D04])**.** *Let* Rec-Maj-3$_\ell$: $\{0,1\}^{3^\ell} \to \{0,1\}$ *be defined as follows: for* $x = (x^1, x^2, x^3)$ *where each* $x^i \in \{0,1\}^{3^{\ell-1}}$,

$$\text{Rec-Maj-3}_\ell(x) \stackrel{def}{=} \text{Maj}(\text{Rec-Maj-3}_{\ell-1}(x^1), \text{Rec-Maj-3}_{\ell-1}(x^2), \text{Rec-Maj-3}_{\ell-1}(x^3)).$$

Then for $\ell \geq \log_{1.1}(1/\delta)$, *we have* $\text{NoiseStab}_\delta[\text{Rec-Maj-3}_\ell] \leq \delta^{-1.1}(3^\ell)^{-.15}$.

Lemma 14 ([MO03])**.** *If* $\eta = O(1/d)$, *we have* $\text{NoiseStab}_{\frac{1-\eta}{2}}[\text{Tribes}_d] = O\left(\frac{\eta d^2}{d2^d}\right) = O\left(\frac{1}{2^d}\right)$.

Lemma 15 ([O'D04])**.** *If* h *is a balanced Boolean function and* $\psi : \{0,1\}^r \to \{0,1\}$ *is arbitrary, then for any* δ *we have* $\text{NoiseStab}_\delta[\psi \circ h^{\otimes r}] = \text{NoiseStab}_{\frac{1}{2} - \frac{\text{NoiseStab}_\delta[h]}{2}}[\psi]$.

Proof of Proposition 12: We take C to be $\text{Tribes}_d \circ \text{Rec-Maj-3}_\ell^{\otimes d2^d}$. Since Rec-Maj-3$_\ell$ is balanced, by Lemma 15 we have

$$\text{NoiseStab}_\delta[C] = \text{NoiseStab}_{\frac{1}{2} - \frac{\text{NoiseStab}_\delta[\text{Rec-Maj-3}_\ell]}{2}}[\text{Tribes}_d].$$

Setting $\delta = \Theta(1/\sqrt{k})$ and recalling that $3^\ell \leq k^6$, we have $\ell \geq \log_{1.1}(1/\delta)$ so we may apply Lemma 13 to obtain

$$\text{NoiseStab}_{\Theta(1/\sqrt{k})}[\text{Rec-Maj-3}_\ell] \leq \Theta((\sqrt{k})^{1.1})(k^6)^{-.15} = O(k^{-.35}).$$

Since $O(k^{-.35}) \leq O(1/d)$, we may apply Lemma 14 with the previous inequalities to obtain

$$\text{NoiseStab}_{\Theta(1/\sqrt{k})}[C] \leq O\left(\frac{1}{2^d}\right).$$

The bound (3.8) follows from some easy rearrangement of the bounds on k, m, d and ℓ. It is easy to see that C can be computed by monotone circuits of depth $O(\ell) = O(\log m)$ and size poly(m), and the proposition is proved. ■

3.2.3 Hardness of Learning Polynomial-size Monotone Circuits

Given a value of k, let $m = 3^\ell d2^d$ for ℓ, d as in Proposition 12. Let $\mathcal{G}$ be a pseudorandom family of functions $\{g : \{0,1\}^k \to \{0,1\}\}$ secure against poly(n)-time adversaries, where

$n = mk$. Since we have $k = \omega(\log n)$, we may apply Corollary 11 with the combining function from Proposition 12 and conclude that the class $\mathcal{C} = \{C \circ \text{slice}_g^{\otimes m} \mid g \in \mathcal{G}\}$ is hard to learn to accuracy

$$\frac{1}{2} + O\Big(\frac{k^3\sqrt{\log m}}{\sqrt{m}}\Big) + o(1/n) \le \frac{1}{2} + O\Big(\frac{k^{3.5}\sqrt{\log n}}{\sqrt{n}}\Big). \tag{3.9}$$

We claim that in fact the functions in $\mathcal{C}$ can be computed by poly(n)-size monotone circuits. This follows from a result of Berkowitz [Ber82] which states that if a k-variable slice function is computed by a AND/OR/NOT circuit of size s and depth d, then it is also computed by a monotone AND/OR/MAJ circuit of size $O(s+k)$ and depth $d+1$. Combining these monotone circuits for slice$_g$ with the monotone circuit for C, we obtain a poly(n)-size monotone circuit for each function in $\mathcal{C}$.

By making various different assumptions on the hardness of one-way functions, Proposition 8 lets us obtain different quantitative relationships between k (the input length for the pseudorandom functions) and n (the running time of the adversaries against which they are secure), and thus different quantitative hardness results from (3.9) above:

Theorem 16. *Suppose that standard one-way functions exist. Then for any constant $\epsilon > 0$ there is a class $\mathcal{C}$ of* poly(n)*-size monotone circuits that is hard to learn to accuracy $\frac{1}{2} + \frac{1}{n^{1/2-\epsilon}}$.*

Proof. If poly(n)-hard one-way functions exist then we may take $k = n^c$ in Proposition 8 for arbitrarily small constant c; this corresponds to taking $d = \gamma \log k$ for γ a large constant in Proposition 12. The claimed bound on (3.9) easily follows. (We note that while not every n is of the required form $mk = 3^\ell d 2^d k$, it is not difficult to see that this and our subsequent theorems hold for all (sufficiently large) input lengths n by padding the hard-to-learn functions.) ■

Theorem 17. *Suppose that subexponentially hard (2^{n^α} for some fixed $\alpha > 0$) one-way functions exist. Then there is a class $\mathcal{C}$ of* poly(n)*-size monotone circuits that is hard to learn to accuracy $\frac{1}{2} + \frac{\text{polylog}(n)}{n^{1/2}}$.*

Proof. As above, but now we take $k = \log^\gamma n$ for some sufficiently large constant γ (*i.e.*, $d = c \log k$ for a small constant c). ■

3.3 Hardness of Learning Simple Circuits

In this section we obtain hardness results for learning very simple classes of circuits computing monotone functions under a concrete hardness assumption for a specific computational problem, namely factoring Blum integers. Naor and Reingold [NR04] showed that if factoring Blum integers is computationally hard then there is a pseudorandom function family which we denote $\mathcal{G}^\star$ that is computable in TC^0. From this it easily follows that the functions $\{\text{slice}_g \mid g \in \mathcal{G}^\star\}$ are also computable in TC^0.

We now observe that the result of Berkowitz mentioned earlier [Ber82] for converting slice circuits into monotone circuits applies not only to AND/OR/NOT circuits, but also to TC^0 circuits (composed of MAJ and NOT gates).

This means that the functions in $\{\text{slice}_g \mid g \in \mathcal{G}^\star\}$ are in fact computable in monotone TC^0, *i.e.,* by polynomial-size, constant-depth circuits composed only of AND/OR/MAJ gates. Since the majority function can be computed by polynomial-size, $O(\log n)$-depth AND/OR circuits, (see e.g. [AKS83]), the functions in $\{\text{slice}_g \mid g \in \mathcal{G}^\star\}$ are computable by $O(\log n)$-depth AND/OR circuits. Finally, using the parameters in Theorem 16 we have a combining function C that is a $O(\log n)$-depth polynomial-size AND/OR circuit which implies the following lemma.

Lemma 18. *Let C be the monotone combining function from Proposition 12 and $\mathcal{G}^\star$ be a family of pseudorandom functions computable in TC^0. Then every function in $\{C \circ \text{slice}_g^{\otimes m} \mid g \in \mathcal{G}^\star\}$ is computable in monotone NC^1.*

This directly yields a hardness result for learning monotone NC^1 circuits (the fourth line of Figure 3.1.1):

Theorem 19. *If factoring Blum integers is hard on average for any $\text{poly}(n)$-time algorithm, then for any constant $\epsilon > 0$ there is a class $\mathcal{C}$ of $\text{poly}(n)$-size monotone NC^1 circuits that is hard to learn to accuracy $\frac{1}{2} + \frac{1}{n^{1/2-\epsilon}}$.*

Now we show that under a stronger but still plausible assumption on the hardness of factoring Blum integers, we get a hardness result for learning a class of *constant-depth* monotone circuits which is very close to a class known to be learnable to any constant accuracy in $\mathrm{poly}(n)$ time. Suppose that n-bit Blum integers are 2^{n^α}-hard to factor on average for some fixed $\alpha > 0$ (this hardness assumption was earlier used by Kharitonov [Kha93]). This means there exists $2^{n^{\alpha/2}}$-secure pseudorandom functions that are computable in TC^0. Using such a family of functions in place of $\mathcal{G}^\star$ in the construction for the preceding theorem and fixing $\epsilon = 1/3$, we obtain:

Lemma 20. *Assume that Blum integers are 2^{n^α}-hard to factor on average. Then there is a class $\mathcal{C}$ of $\mathrm{poly}(n)$-size monotone NC^1 circuits that is hard for any $2^{n^{\alpha/20}}$-time algorithm to learn to accuracy $\frac{1}{2} + \frac{1}{n^{1/6}}$.*

Now we "scale down" this class $\mathcal{C}$ as follows. Let n' be such that $n' = (\log n)^\kappa$ for a suitable constant $\kappa > 20/\alpha$, and let us use "n'" as the "n" in the construction of the previous lemma; we call the resulting class of functions $\mathcal{C}'$. In terms of n, the functions in $\mathcal{C}'$ (which are functions over $\{0,1\}^n$ which only depend on the first n' variables) are computed by $(\log n)^{O(\kappa)}$-size, $O(\log\log n)$-depth monotone circuits whose inputs are the first $(\log n)^\kappa$ variables in $x_1, \dots, x_n$. We moreover have that $\mathcal{C}'$ is hard for any $2^{(n')^{\alpha/20}} = 2^{(\log n)^{\kappa\alpha/20}} = \omega(\mathrm{poly}(n))$-time algorithm to learn to some accuracy $\frac{1}{2} + \frac{1}{(n')^{1/6}} = \frac{1}{2} + o(1)$.

We now recall the following variant of Nepomnjaščiĭ's theorem that is implicit in the work of Allender *et al.* [AHM+06].

Lemma 21. *For every language $\mathcal{L} \in \mathsf{NL}$, for all sufficiently large constant d there are AC^0_d circuits of size $2^{n^{O(1)/(d+1)}}$ that recognize $\mathcal{L}$.*

Since every function in $\mathcal{C}'$ can be computed in NC^1 which is contained in NL, combining Lemma 21 with the paragraph that proceeds it, we obtain the following theorem (the final line of Figure 3.1.1):

Theorem 22. *Suppose that Blum integers are subexponentially hard to factor on average. Then there is a class $\mathcal{C}$ of monotone functions that is hard for any $\mathrm{poly}(n)$-time algorithm to learn to accuracy $\frac{1}{2} + o(1)$ and which, for all sufficiently large constant d, are computed by AC^0_d circuits of size $2^{(\log n)^{O(1)/(d+1)}}$.*

This final hardness result is of interest because it is known that constant-depth circuits of only slightly smaller size *can* be learned to any constant accuracy in $\mathrm{poly}(n)$ time under the uniform distribution (without needing membership queries).

Theorem 23 ([Ser04] Corollary 2)**.** *For all $d \geq 2$, the class of AC^0_d circuits of size $2^{O((\log n)^{1/(d+1)})}$ that compute monotone functions can be learned to any constant accuracy $1 - \epsilon$ in* poly(n)*-time.*

Theorem 22 is thus nearly optimal in terms of the size of the constant-depth circuits for which it establishes hardness of learning.

3.4 A Computational Analogue of the BBL Lower Bound

In this section we first present a simple variant of the Blum *et al.* [BBL98] lower-bound construction, obtaining an information-theoretic lower bound on the learnability of the general class of all monotone Boolean functions. The quantitative bound our variant achieves is weaker than that of BBL, but has the advantage that it can be easily "pseudorandomized". Indeed our construction uses a certain probability distribution over monotone DNF formulas, such that a typical random input x satisfies only poly(n) many "candidate terms" (terms which may be present in a random DNF formula drawn from our distribution). By selecting terms for inclusion in the DNF formula in a pseudorandom rather than truly random way, we obtain a class of poly(n)-size monotone circuits that is hard to learn to accuracy $\frac{1}{2} + \frac{1}{\text{polylog}(n)}$ (assuming one-way functions exist).

Below we start with an overview of why it is difficult to obtain a computational analogue of the exact construction of BBL using the "pseudorandomization" approach sketched above, and the idea behind our variant, which overcomes this difficulty. We then provide our information theoretic construction and analysis, followed by its computational analogue.

Idea Recall BBL's information-theoretic lower bound. It works by defining a distribution P_s over monotone functions $\{0,1\}^n \rightarrow \{0,1\}$ which is as follows. (Here s is a numerical parameter which should be thought of as the number of membership queries that a learning algorithm is allowed to make.) Take $t = \log(3sn)$. A draw from P_s is obtained by randomly including each length-t monotone term in the DNF formula independently with probability p', where p' is chosen so that the function is expected to be balanced on "typical inputs" (more precisely, on inputs with exactly $n/2$ 1's). The naive idea for pseudorandomizing this construction is to simply use a pseudorandom function with bias p' to determine whether each possible term of size t should be included or excluded in the DNF formula. However, there is a problem with this approach: we do not know an efficient way to determine whether a typical example x (with, say, $n/2$ ones) has any of its $\binom{n/2}{t}$ candidate terms (each of which is pseudorandomly present/not present in f) actually present in f, so we do not know how to evaluate f on a typical input x in less than $\binom{n/2}{t}$ time.

We get around this difficulty by instead considering a new distribution of random monotone DNF formulas. In our construction we will again use a random function with bias p to determine whether each possible term of length t is present in the DNF formula. However, in our construction, a typical example x will have only a polynomial number of candidate terms that could be satisfied, and thus it is possible to check all of them and evaluate the function in poly(n) time.

The main difficulty of this approach is to ensure that although a typical example has only a polynomial number of candidate terms, the function is still hard to learn in polynomial time. We achieve this by partitioning the variables into blocks of size k and viewing each block as a "super-variable" (corresponding to the AND of all k variables in the block). We then construct the DNF formula by randomly choosing length-t terms over these super-variables. It is not difficult to see that with this approach, we can equivalently view our problem as learning a t-DNF formula f with terms chosen as above, where each of the n/k variables is drawn from a product distribution with bias $1/2^k$. By fine-tuning the parameters that determine t (the size of each term of the DNF formula) and k (the size of the partitions), we are able to achieve an information-theoretic lower bound showing that this distribution over monotone functions is hard to learn to accuracy $1/2 + o(1)$.

Construction Let us partition the variables $x_1, \ldots, x_n$ into $m = n/k$ blocks $B_1, \ldots, B_m$ of k variables each. Let X_i denote the conjunction of all k variables in B_i ($X_1, \ldots, X_m$ are the super-variables). The following is a description of our distribution P over monotone functions. A function f is drawn from P as follows (we will fix the values of k and t later):

- Construct a monotone DNF formula f_1 as follows: each possible conjunction of t super-variables chosen from $\{X_1, \ldots, X_m\}$ is placed in the target function f_1 independently with probability p, where p is defined as the solution to:

$$(1-p)^{\binom{m/2^k}{t}} = 1/2. \tag{3.10}$$

Note that for a uniform $x \in \{0,1\}^n$, we expect the corresponding "super-assignment" $X = (X_1, \dots, X_m)$ to have $m/2^k$ 1's in it, and any super-assignment with this many 1's will be satisfied by $\binom{m/2^k}{t}$ many terms. Thus p is chosen such that a "typical" example X, with $m/2^k$ ones, has probability 1/2 of being labeled positive under f_1.

- Let

$$f(x) = \begin{cases} f_1(x) & \text{if the \# of super-vars satisfied in } x \text{ is } \le m/2^k + (m/2^k)^{2/3}; \\ 1 & \text{otherwise.} \end{cases}$$

Note that because of the final step of the construction, the function f is not actually a DNF formula (though it is a monotone function). Intuitively, the final step is there because if too many super-variables were satisfied in x, there could be too many (more than poly(n)) candidate terms to check, and we would not be able to evaluate f_1 efficiently. We will show later that the probability that the number of super-variables satisfied in x is greater than $m/2^k + (m/2^k)^{2/3}$ is at most $2e^{-(m/2^k)^{1/3}/3} = 1/n^{\omega(1)}$, and thus the function f is $1/n^{\omega(1)}$-close to f_1; so hardness of learning results established for the random DNF formulas f_1 carry over to the actual functions f. For most of our discussion we shall refer to P as a distribution over DNF formulas, meaning the functions f_1.

3.4.1 Information-Theoretic Lower Bound

As discussed previously, we view the distribution P defined above as a distribution over DNF formulas of terms of size t over the super-variables. Each possible combination of t super-variables appears in f_1 independently with probability p and the super-variables are drawn from a product distribution that is 1 with probability $1/2^k$ and 0 with probability $1 - 1/2^k$. We first observe that learning f over the super-variables drawn from the product distribution is equivalent to learning the original function over the original variables. This is because if we are given the original membership query oracle for n-bit examples we can simulate answers to membership queries on m-bit "super-variable" examples and vice versa. Thus we henceforth analyze the product distribution.

We follow the proof technique of BBL. To simplify our analysis, we consider an "augmented" oracle, as done by BBL. Given a query X, with 1's in positions indexed by the set S_X, the oracle will return the first conjunct in lexicographic order that appears in the target function and is satisfied by X. Additionally, the oracle returns 1 if X is positive and 0 if X is negative. (So instead of just giving a single bit as its response, if the example is a positive one the oracle tells the learner the lexicographically first term in the target DNF formula that is satisfied.) Clearly, lower bounds for this augmented oracle imply the same bounds for the standard oracle.

We are interested in analyzing P_s, the conditional distribution over functions drawn from the initial distribution P that are consistent with the information learned by A in the first s queries. We can think of P_s as a vector V_s of $\binom{m}{t}$ elements, one for each possible conjunct of size t. Initially, each element of the vector contains p, the probability that the conjunct is in the target function. When a query is made, the oracle examines one by one the entries that satisfy X. For each entry having value p, we can think of the oracle as flipping a coin and replacing the entry by 0 with probability $1-p$ and by 1 with probability p. After s queries, V_s will contain some entries set to 0, some set to 1 and the rest set to p. Because V_s describes the conditional distribution P_s given the queries made so far, the Bayes-optimal prediction for an example X is simply to answer 1 if $V_s(X) \geq 1/2$ and 0 otherwise.

We now analyze $V_s(X)$, the conditional probability over functions drawn from P that are consistent with the first s queries that a random example, X, drawn from the distribution, evaluates to 1, given the answers to the first s queries. We will show that for $s = \text{poly}(n)$, for X drawn from the product distribution on $\{0,1\}^m$, with probability at least $1 - 1/n^{\omega(1)}$ the value $V_s(X)$ lies in $\frac{1}{2} \pm \frac{1}{\log n}$. This is easily seen to give a lower bound of the type we require.

Following BBL, we first observe that after s queries there can be at most s entries set to one in the vector V_s. We shall also use the following lemma by BBL:

Lemma 24 ([BBL98]). *After s queries, with probability $1 - e^{-s/4}$, there are at most $2s/p$ zeros in V_s.*

We thus may henceforth assume that there are at most $2s/p$ zeros in V_s.

We now establish the following, which is an analogue tailored to our setting of Claim 3 of the paper by BBL [BBL98]:

Lemma 25. *For any vector V_s of size $\binom{m}{t}$ with at most s entries set to 1, at most $2s/p$ entries set to 0, and the remaining entries set to p, for a random example X (drawn from $\{0,1\}^m$ according to the $1/2^k$-biased product distribution), we have that with probability at least $1-\epsilon_1$, the quantity $V_s(X)$ lies in the range*

$$1-(1-p)^{[\binom{m/2^k-(m/2^k)^{1/3}}{t}-\frac{2s\sqrt{n}}{p2^{kt}}]} \le V_s(X) \le 1-(1-p)^{\binom{m/2^k+(m/2^k)^{1/3}}{t}}. \tag{3.11}$$

Here

$$\epsilon_1 = s\cdot(\frac{2\sqrt{n}}{p}+1)2^{-kt}+2e^{-(m/2^k)^{1/3}/3}. \tag{3.12}$$

Proof. Let X be a random example drawn from the $1/2^k$-biased product distribution over $\{0,1\}^m$.

Consider the following 3 events:

- **None of the 1-entries in V_s are satisfied by X.**

  There are at most s 1-entries in V_s and the probability that any one is satisfied by X is 2^{-kt}. Therefore the probability that some 1-entry is satisfied by X is at most $s2^{-kt}$ and the probability that none of the 1-entries in V_s are satisfied by X is at least $1-s2^{-kt}$.

- **At most $(2s\sqrt{n}/p)2^{-kt}$ of the 0-entries in V_s are relevant to X.**

  Since there are at most $2s/p$ entries set to 0 in V_s, the expected number of 0-entries in V_s satisfied by X is at most $(2s/p)2^{-kt}$. By Markov's inequality, the probability that the actual number exceeds this by a $\sqrt{n}$ factor is at most $1/\sqrt{n}$.

- **The number of 1's in X lies in the range $m/2^k \pm (m/2^k)^{2/3}$.**

  Using the Chernoff bound, we have that this occurs with probability at least $1-2e^{-(m/2^k)^{1/3}/3}$. Note that for X's in this range, $f(X)=f_1(X)$. So conditioning on this event occurring, we can assume that $f(X)=f_1(X)$.

Therefore, the probability that all 3 of the above events occurs is at least $1-\epsilon_1$ where $\epsilon_1 = s\cdot(\frac{2\sqrt{n}}{p}+1)2^{-kt}+2e^{-(m/2^k)^{1/3}/3}$.

Given that these events all occur, we show that $V_s(X)$ lies in the desired range. We follow the approach of BBL. For the lower bound, $V_s(X)$ is minimized when X has as few 1's as possible and when as many of the 0-entries in V_s are relevant to X as possible. So $V_s(X)$ is at least

$$V_s(X) \geq 1 - (1-p)^{[\binom{m/2^k - (m/2^k)^{2/3}}{t} - \frac{2s\sqrt{n}}{p2^{kt}}]}.$$

For the upper bound, $V_s(X)$ is maximized when X has as many 1's as possible and as few 0's as possible. So $V_s(X)$ is at most

$$V_s(X) \leq 1 - (1-p)^{\binom{m/2^k + (m/2^k)^{2/3}}{t}},$$

and thus $V_s(X)$ lies in the desired range.. ■

Now let us choose values for k and t. What are our goals in setting these parameters? First off, we want $\binom{m/2^k}{t}$ to be at most poly(n) (so that there are at most poly(n) candidate terms to be checked for a "typical" input). Moreover, for any $s = \text{poly}(n)$ we want (3.11)'s two sides to both be close to $1/2$ (so the accuracy of any s-query learning algorithm is indeed close to $1/2$ on typical inputs), and we want ϵ_1 to be small (so almost all inputs are "typical"). With this motivation, we set $k = \Theta(\log n)$ to be such that $m/2^k$ (recall, $m = n/k$) equals $\log^6 n$, and we set $t = \sqrt{\log n}$. This means $\binom{m/2^k}{t} = \binom{\log^6 n}{\sqrt{\log n}} \leq 2^{6\log(\log n)\sqrt{\log n}} \ll n$. Now (3.10) gives $p \gg 1/n$; together with $k = \Theta(\log n)$, for any $s = \text{poly}(n)$ we have $\epsilon_1 = 1/n^{\omega(1)}$.

Now we analyze (3.11). First the lower bound:

$$\begin{aligned}
V_s(X) &\geq 1 - (1-p)^{[\binom{m/2^k - (m/2^k)^{2/3}}{t} - \frac{2s\sqrt{n}}{p2^{kt}}]} \\
&\geq 1 - (1-p)^{\binom{m/2^k - (m/2^k)^{2/3}}{t}} \left(e^{\frac{3s\sqrt{n}}{p2^{kt}}}\right) \\
&= 1 - (1-p)^{\binom{m/2^k - (m/2^k)^{2/3}}{t}} \left(1 + 1/n^{\omega(1)}\right) \\
&= 1 - \left[2^{-\binom{m/2^k - (m/2^k)^{2/3}}{t} / \binom{m/2^k}{t}}\right] \cdot \left(1 + 1/n^{\omega(1)}\right).
\end{aligned}$$

(In the last step here we are using the definition of p from (3.10).) Let us bound the exponent:

$$\begin{aligned}
\frac{\binom{m/2^k-(m/2^k)^{2/3}}{t}}{\binom{m/2^k}{t}} &\geq \left(\frac{m/2^k-(m/2^k)^{2/3}-t}{m/2^k}\right)^t \\
&= \left(\frac{\log^6 n-\log^4 n-\sqrt{\log n}}{\log^6 n}\right)^{\sqrt{\log n}} \\
&\geq \left(\frac{\log^6 n-2\log^4 n}{\log^6 n}\right)^{\sqrt{\log n}} \\
&= \left(1-\frac{2}{\log^2 n}\right)^{\sqrt{\log n}} \\
&\geq 1-\frac{2}{\log^{1.5} n}.
\end{aligned}$$

So

$$V_s(X) \quad \geq \quad 1-\left[2^{-(1-2/\log^{1.5} n)}\right]\cdot(1+1/n^{\omega(1)}) \geq \frac{1}{2}-\frac{1}{\log n}.$$

Now for the upper bound:

$$\begin{aligned}
V_s(x) &\leq 1-(1-p)^{\binom{m/2^k+(m/2^k)^{2/3}}{t}} \\
&= 1-2^{-\binom{m/2^k+(m/2^k)^{2/3}}{t}/\binom{m/2^k}{t}}.
\end{aligned}$$

Again bounding the exponent:

$$\begin{aligned}
\frac{\binom{m/2^k+(m/2^k)^{2/3}}{t}}{\binom{m/2^k}{t}} &= \frac{\binom{\log^6 n+\log^4 n}{\sqrt{\log n}}}{\binom{\log^6 n}{\sqrt{\log n}}} \\
&\leq \left(\frac{\log^6 n+\log^4 n}{\log^6 n-\sqrt{\log n}}\right)^{\sqrt{\log n}} \\
&\leq \left(1+\frac{2\log^4 n}{\log^6 n-\sqrt{\log n}}\right)^{\sqrt{\log n}} \\
&\leq 1+\frac{4}{\log^{1.5} n}.
\end{aligned}$$

So

$$V_s(X) \leq 1-2^{-\left(1+\frac{4}{\log^{1.5} n}\right)} \leq \frac{1}{2}+\frac{1}{\log n}.$$

The above analysis has thus established the following.

Lemma 26. *Let L be any* $\mathrm{poly}(n)$*-time learning algorithm. If L is run with a target function that is a random draw f from the distribution P described above, then for all but a $1/n^{\omega(1)}$ fraction of inputs $x \in \{0,1\}^n$, the probability that $h(x) = f(x)$ (where h is the hypothesis output by L) is at most $\frac{1}{2} + \frac{1}{\log n}$.*

It is easy to see that by slightly modifying the values of t and k in the above construction, it is actually possible to replace $\frac{1}{\log n}$ with any $\frac{1}{\mathrm{polylog}\, n}$ in the above.

3.4.2 Computational Lower Bound

To obtain a computational analogue of Lemma 26, we "pseudorandomize" the choice of terms in a draw of f_1 from P.

Recall that the construction P placed each possible term (conjunction of t super-variables) in the target function with probability p, as defined in (3.10). We first consider a distribution that uses uniform bits to approximate the probability p. This can be done by approximating $\log(p^{-1})$ with $\mathrm{poly}(n)$ bits, associating each term with independent uniform $\mathrm{poly}(n)$ bits chosen this way, and including that term in the target function if all bits are set to 0. It is easy to see that the resulting construction yields a probability distribution which is statistically close to P, and we denote it by P^{stat}.

Now, using a pseudorandom function rather than a truly random (uniform) one for the source of uniform bits will yield a distribution which we denote by P^{PSR}. Similar arguments to those we give elsewhere in this chapter show that a $\mathrm{poly}(n)$ time adversary cannot distinguish the resulting construction from the original one (or else a distinguisher could be constructed for the pseudorandom function).

To complete the argument, we observe that every function f in the support of P^{PSR} can be evaluated with a $\mathrm{poly}(n)$-size circuit. It is obviously easy to count the number of super-variables that are satisfied in an input x, so we need only argue that the function f_1 can be computed efficiently on a "typical" input x that has "few" super-variables satisfied. But by construction, such an input will satisfy only $\mathrm{poly}(n)$ candidate terms of the monotone DNF formula f_1 and thus a $\mathrm{poly}(n)$-size circuit can check each of these candidate terms separately (by making a call to the pseudorandom function for each candidate term to determine whether it is present or absent). Thus, as a corollary of Lemma 26, we can establish the main result of this section:

Theorem 27. *Suppose that standard one-way functions exist. Then there is a class $\mathcal{C}$ of* $\mathrm{poly}(n)$*-size monotone circuits that is hard to learn to accuracy* $\frac{1}{2} + \frac{1}{\mathrm{polylog}(n)}$.

Chapter 4

The Statistical Hardness of Learning Monotone Functions

Having shown the conditional cryptographic hardness of learning various classes of monotone functions computed by simple circuits, in this chapter we show the unconditional hardness of learning a class of monotone functions computed by even simpler circuits in the statistical query model.

4.1 Introduction

The hardness results shown in Chapter 3 were all *conditional*, *i.e.,* they depended on widely believed, but unproven cryptographic assumptions. Some of these assumptions, such as the existence of one-way functions are weak and considered to be reasonable [Gol05]. Assuming the existence of a particular one-way function with a certain hardness, *e.g.,* factoring Blum integers is 2^{n^ϵ}-hard for some fixed $\epsilon > 0$, is much stronger.

This chapter complements the previous one by proving lower bounds that are *unconditional*, but which are only true for a restriction of the PAC learning model. In particular, we will be using the statistical query (SQ) model of Kearns [Kea98] where instead of being given random examples, the learner is allowed queries of the form: "What is the approximate expected value of the function g on a random example labeled by the concept c?"

The motivation behind Kearns' elegant model is that any algorithm for learning in the

SQ model can automatically be converted to an algorithm for learning in the presence of random classification noise in the standard PAC model (and more general forms of noise as well [Dec93]). With the exception of the algorithms for learning parity functions [FS92, HSW92, BKW03], every known PAC learning algorithm can be couched as an SQ learning algorithm, and can hence be made noise-tolerant. In fact, virtually all known noise-tolerant PAC learning algorithms have been obtained from SQ algorithms [Byl94, BFKV98, DV04].

Unlike the conditional hardness results in PAC learning, Kearns demonstrated that there are information-theoretic barriers to SQ learning. In particular, he showed that the class of parity functions over n variables require exponentially many SQ queries to even be weakly-learned [Kea98]. Later, Blum *et al.* [BFJ$^+$94] completely characterized the weak-learnability of a concept class in the SQ model by introducing the *statistical query dimension* of a concept class, which is defined to be the largest subset of the class such that all its elements are pair-wise nearly orthogonal. The SQ-dimension was subsequently strengthened and extended to other variants [BF02, Yan01, Yan05, Fel08].

The SQ-dimension was used to lower bound the weak-learnability of several concept classes by Klivans and Sherstov [KS06], while Sherstov [She07] gave an upper-bound on the SQ-dimension of halfspaces. On the other hand, under the uniform distribution, it has been known for a long time that monotone functions are easily weak-learned in the SQ model [KLV94, BT06, BBL98, OW09]. Perhaps surprisingly, the optimal algorithm of O'Donnell and Wimmer is quite simple — the algorithm asks for the correlation of the target function with the constant functions 0 and 1, the dictator functions $x_1, \ldots, x_n$, and the majority function. These $n + 3$ statistical queries are enough to guarantee a $\frac{1}{2} + \Omega(\frac{\log n}{\sqrt{n}})$ accurate hypothesis for any monotone function.

In the distribution-free model, weak-learnability implies strong-learnability by boosting [Sch90, Fre95, Sch01]. However, for a fixed distribution, say the uniform distribution, this is not the case, and thus SQ-dimension upper bounds say very little about the strong-learnability of a concept class. This is particularly true for monotone functions where many concept classes have no known strong-learning algorithms. Simon [Sim07] recently introduced the notion of the *strong statistical query dimension* of a concept class which characterizes its strong-learnability. This characterization was later simplified by Feldman

[Fel09].

Our results. We give the first strong statistical query dimension lower bound for a monotone concept class with succinct representation. We show that the class of functions computed by polynomial-size depth-3 AND/OR-circuits has super-polynomial strong SQ-dimension, and thus cannot be efficiently learned to arbitrary accuracy by any statistical query learner.

This lower bound is tantalizingly close to a lower bound for polynomial-size monotone DNF formulas (*i.e.,* depth-2 AND/OR-circuits). The class of polynomial-size (non-monotone) DNF formulas is known to have super-polynomial (weak) SQ-dimension under the uniform distribution, and thus we know that none of our current algorithmic techniques can even weakly-learn DNF formulas. The proof of the lower bound for this class easily follows from the orthogonality of the parity functions under the uniform distribution. A polynomial-sized DNF formula can compute a parity function over $O(\log n)$ variables, thus the SQ-dimension of polynomial-sized DNF formulas is at least $\binom{n}{\log n} = n^{\Omega(\log n)}$.

Most known statistical query lower bounds are based on the orthogonality of the parity functions, and we use the same observation here. In Section 4.3, we embed parity functions into the middle slice of the Boolean hypercube. These slice functions of course are not completely orthogonal to each other so we pick a carefully constructed subset that are all *nearly* orthogonal to each other.

We define the statistical query model and the various SQ-dimensions in Section 4.2. We define our concept class and prove our strong SQ-dimension lower bound in Section 4.3. Finally, in Section 4.4 we show that our concept class can be computed by polynomial-size depth-3 AND/OR-circuits.

4.2 The Statistical Query Model

For the rest of this chapter we will consider the range of Boolean concept classes to be $\{+1, -1\}$ instead of $\{0, 1\}$.

Kearns [Kea98] introduced the *statistical query (SQ) model* where instead of having access to an example oracle $EX(f, \mathcal{D})$, the learner has access to a *statistical query oracle*,

$SQ_{f,\mathcal{D}}$.

Definition 28. *Let $\mathcal{C}$ be a concept class over X. An algorithm A is said to be a* statistical query (SQ) learning algorithm *for $\mathcal{C}$ if for all $0 < \epsilon, \delta < 1$, for all $f \in \mathcal{C}$, if A on input ϵ, δ and access to the statistical query oracle $SQ_{f,\mathcal{D}}$ with probability at least $1-\delta$ outputs a hypothesis h such that $\Pr_{x\in\mathcal{D}}[h(x) \neq f(x)] \leq \epsilon$.*

$SQ_{f,\mathcal{D}}$ takes as input a function $g : X \times \{+1,-1\} \to \{+1,-1\}$ and a real number $\tau \in [0,1]$ and outputs a value v such that:

$$\left| v - \mathop{\mathbb{E}}_{\mathcal{D}}[g(x, f(x))] \right| \leq \tau.$$

A is only allowed to make queries in which g can be computed by a polynomial-size circuit, and τ is at most a fixed $\mathrm{poly}^{-1}(n, \mathrm{size}(f), \epsilon^{-1}, \delta^{-1})$.

Note that if τ is 0 then the learner can make membership queries. If A runs in time $\mathrm{poly}(n, \mathrm{size}(f), \epsilon^{-1}, \delta^{-1})$ we say that A is a *polynomial-time SQ learning algorithm.* A concept class is said to be *efficiently SQ learnable* if there exists polynomial-time SQ learning algorithm for the class.

Learnability in the SQ-model has several interesting consequences and is thus well-studied. If a concept class $\mathcal{C}$ is efficiently learnable from statistical queries, then it is efficiently PAC-learnable with random classification noise (*i.e.,* learning when the labels are incorrect with some probability). The idea is that the learner can simply simulate the oracle $SQ_{f,\mathcal{D}}$ and run the SQ learning algorithm. To answer a particular query (g, τ), the learner can take a sample of random examples from the noisy oracle and output an estimate of $\mathbb{E}_{\mathcal{D}}[g(x, f(x)]$.

One could conjecture that the other direction is true, *i.e.,* every concept class learnable with random classification noise is learnable with statistical queries. Unfortunately, this conclusion is false. Kearns ([Kea98]) showed that the class of parities with *no* classification noise is not learnable from statistical queries. Contrast this with the fact that efficient PAC algorithms (Gaussian elimination) for this class exist [FS92, HSW92]. These algorithms make explicit calculations on the given examples, and thus they cannot be turned into statistical query algorithms. Later, a polynomial-time algorithm was given for learning parities over the first $O(\log n \log\log n)$ bits of the input with random classification noise [BKW03]. This algorithm of course was not a statistical query algorithm.

Blum *et al.* [BFJ+94] generalized Kearns' negative result by introducing the notion of the SQ-dimension, which completely characterized the weak-learnability of concept classes in the SQ model. Bshouty and Feldman [BF02] and Yang [Yan05] later generalized and improved the Blum *et al.* result. We will use Yang's characterization here extended to sets of arbitrary real-valued functions.

Definition 29. *Given a set of real-valued functions $\mathcal{C}$, the* SQ-dimension *of $\mathcal{C}$ with respect to $\mathcal{D}$ (written $\mathrm{SQdim}_{\mathcal{D}}(\mathcal{C})$) is the largest number d such that $\exists\{f_1, \ldots, f_d\} \subseteq \mathcal{C}$ with the property that $\forall i \neq j$,*

$$\frac{|\langle f_i, f_j\rangle_{\mathcal{D}}|}{\|f_i\|_{\mathcal{D}} \cdot \|f_j\|_{\mathcal{D}}} \leq \frac{1}{d}.$$

Intuitively, this says that $\mathcal{C}$ contains d "nearly-uncorrelated" functions. If a concept class $\mathcal{C}$ has SQ-dimension d, then we can weakly learn $\mathcal{C}$ using d queries. Take the maximal set and ask for the correlation between the target function and each function in the set. Since the set is maximal, the target function must have correlation at least $1/d$ with at least one of the functions.

Blum *et al.* showed that the other direction was true as well.

Theorem 30 ([BFJ+94] Theorem 12)**.** *Given a concept class $\mathcal{C}$ and a distribution $\mathcal{D}$ on the input, let $\mathrm{SQdim}(\mathcal{C}, \mathcal{D}) = d$. Then if the tolerance τ is always at least $1/d^{1/3}$, at least $\frac{1}{2}d^{1/3} - 1$ queries are required to learn $\mathcal{C}$ with advantage $1/d^3$.*

In particular, the class of parity functions on n variables has $\mathrm{SQdim}(PAR) = 2^n$. Thus, any SQ algorithm requires an exponential number of queries.

4.2.1 The Strong SQ Dimension

Given a fixed distribution, the statistical query dimension only characterizes the weak SQ-learnability of a class and says nothing about its strong SQ-learnability. For example, any class of monotone functions can be easily weak-learned with respect to the uniform distribution [KLV94, BBL98, OW09], however for many simple classes of monotone functions, *e.g.,* polynomial-sized monotone DNF formulas, no strong-learning algorithms are known

(SQ or not). The first characterization of strong SQ learning was due to Simon [Sim07], but we will follow a subsequent characterization due to Feldman [Fel09].

Let $\mathcal{F}_1^\infty$ denote the set of all functions from $\{0,1\}^n \to [-1,1]$, *i.e.,* all functions with L_∞ norm bounded by 1. We define $B^2_{\mathcal{D}}(f,\epsilon)$ to be $\{g : \|g-f\|^2_{\mathcal{D}} \leq \epsilon\}$, *i.e.,* the ϵ-ball around f, and $\mathbf{0}$ to be the constant 0 function. Finally, for a set of real-valued functions $\mathcal{C}$, let $\mathcal{C} - g = \{f - g : f \in \mathcal{C}\}$.

Definition 31. *Given a concept class $\mathcal{C}$ and $\epsilon > 0$, the* strong SQ-dimension *of $\mathcal{C}$ with respect to $\mathcal{D}$ is:*

$$\mathrm{SSQdim}_{\mathcal{D}}(\mathcal{C},\epsilon) = \max_{g \in \mathcal{F}_1^\infty} \mathrm{SQdim}_{\mathcal{D}}((\mathcal{C} - g) \setminus B^2_{\mathcal{D}}(\mathbf{0},\epsilon)).$$

Just as for the weak SQ-dimension, the strong SQ-dimension completely characterizes the strong SQ-learnability of a concept class.

Theorem 32 ([Fel09])**.** *Given a concept class $\mathcal{C}$ and distribution $\mathcal{D}$, there exists a polynomial p such that $\mathcal{C}$ is SQ-learnable from $p(n,1/\epsilon)$ queries of tolerance $p(n,1/\epsilon)$ if and only if for every $\epsilon > 0$, $\mathrm{SSQdim}_{\mathcal{D}}(\mathcal{C},\epsilon) \leq p'(n,1/\epsilon)$ for some polynomial p'.*

Given this definition and theorem, we can show that a concept class cannot be efficiently strong-learned in the SQ model by finding a suitably chosen g and showing that $\mathrm{SQdim}_{\mathcal{D}}((\mathcal{C} - g) \setminus B^2_{\mathcal{D}}(\mathbf{0},\epsilon)) = n^{\omega(1)}$.

4.3 The Strong SQ Lower Bound

We will exhibit a family of monotone functions that cannot be strong SQ-learned with a polynomial number of queries under the uniform distribution. For the rest of this chapter, let us fix $\mathcal{D}$ to be the uniform distribution over $\{0,1\}^n$. The idea of what follows is that we will embed a family of non-monotone functions with high SQ-dimension – a family of parity functions – into the middle level of the k-dimensional Boolean cube, and thus obtain a class of monotone functions with high strong SQ-dimension.

The classes of functions we consider will only be defined over the first k out of n variables.

Theorem 33. *Let $\mathcal{P}$ be the class of $\binom{k}{j}$ parity functions $\chi\colon \{0,1\}^k \to \{+1,-1\}$ over exactly j out of the first k variables, and let $\mathcal{M}$ be the slice functions slice_χ for $\chi \in \mathcal{P}$. For $k = \log^{1.5}(n)$, $j = \log(n)$, and $\epsilon = o(1/\sqrt{k})$, $\text{SSQdim}(\mathcal{M}, \epsilon) = n^{\omega(1)}$.*

Proof. Let $g = \text{slice}_{\mathbf{0}}$. We will show that $\text{SQdim}((\mathcal{M} - g) \setminus B^2(\mathbf{0}, \epsilon)) = n^{\omega(1)}$. By Stirling's approximation, the middle layer of the k-dimensional hypercube is a $\Theta(1/\sqrt{k})$ fraction of the 2^k points. Thus for $\epsilon = o(1/\sqrt{k})$, $\mathcal{M} - g \cap B^2(\mathbf{0}, \epsilon) = \emptyset$, and we only have to concern ourselves with $\text{SQdim}(\mathcal{M} - g)$ to lower bound the strong SQ-dimension of $\mathcal{M}$.

The functions in $\mathcal{M} - g$ have a nice structure since they output 0 everywhere except the middle layer. Thus, the correlation between any two functions in $\mathcal{M} - g$ will only depend on the outputs of the middle slice. Given two parity functions $\chi_1, \chi_2 \in \mathcal{P}$, let s be the number of variables in the parity $\chi_1 \cdot \chi_2$, *i.e.,* the number of variables in the symmetric difference between χ_1 and χ_2. The correlation between $(\text{slice}_{\chi_1} - g)$ and $(\text{slice}_{\chi_2} - g)$ can be written as $\langle \chi_1\chi_2, \mathbf{1}_{|x|=k/2} \rangle$, which is just a degree-s Fourier coefficient of $\mathbf{1}_{|x|=k/2}$. By symmetry all the degree-s Fourier coefficients of $\mathbf{1}_{|x|=k/2}$ are the same, and since the squares of all the Fourier coefficients sum to one by Parseval's Theorem, $\langle \chi_1\chi_2, \mathbf{1}_{|x|=k/2} \rangle \le \binom{k}{s}^{-1/2}$.

Using the "designs" of Nisan and Wigderson [NW94] we can see that there are at least $\binom{k}{j/2} = n^{\Theta(\log\log n)}$ parities whose symmetric difference is at least $j/2$.

Lemma 34 ([NW94])**.** *There exists a set system of size $\binom{k}{j/2}$ such that each set is a subset of $[k]$ of size exactly j and every pair of sets has symmetric difference at least $j/2$.*

Thus, we have $n^{\Theta(\log\log n)}$ functions in $\mathcal{M} - g$ whose pair-wise correlation is at most $1/n^{\Theta(\log\log n)}$, and the SQ-dimension of $\mathcal{M} - g$ is at least $n^{\Theta(\log\log n)}$. ■

4.4 The Circuit Construction

We will now show that the concept class $\mathcal{M}$ can be computed by small monotone circuits.

Theorem 35. *Let $\mathcal{P}$ be the class of $\binom{k}{j}$ parity functions $\chi\colon \{0,1\}^k \to \{+1,-1\}$ over exactly j out of the first k variables, and let $\mathcal{M}$ be the slice functions slice_χ for $\chi \in \mathcal{P}$. For $k = \log^{1.5}(n)$ and $j = \log(n)$, the functions $\text{slice}_\chi \in \mathcal{M}$ can computed by $\text{poly}(n)$-size depth-3* AND/OR-*circuits.*

Proof. Let Th^k_j be the threshold function that outputs TRUE if at least j of the k inputs are set to 1, and FALSE otherwise. The threshold function Th^k_j for $k = o(\log^2(n)/\log\log(n))$ is computable by a monotone formula of $\mathrm{poly}(n)$ size and depth 3 [KPPY84].

Let χ be a parity function on j out of the first k variables. Let Th^j_i take the j variables of χ as input, and let $\mathrm{Th}^{k-j}_{k/2-i}$ take the $k-j$ variables outside of χ as input. Observe that slice_χ is equivalent to $\vee_{\mathrm{odd} i<j}[\mathrm{Th}^j_i \wedge \mathrm{Th}^{k-j}_{k/2-i}]$. If an input x has fewer than $k/2$ ones, then there can be no i such that Th^j_i and $\mathrm{Th}^{k-j}_{k/2-i}$ both hold, so this function outputs 0 as it should. If x has more than $k/2$ ones, some ℓ of them are in χ, and at least $k/2 - \ell + 1$ of them are in the $k-j$ variables outside χ. If ℓ is odd then $i = \ell$ makes the OR true, and if ℓ is even then $i = (\ell - 1)$ makes the OR true. Finally, if x has exactly $k/2$ ones, and an odd number of them are in χ, the formula is satisfied; if an even number of them are in χ, the formula is not satisfied.

Each Th^j_i and $\mathrm{Th}^{k-j}_{k/2-i}$ can be computed by a depth-3 monotone $\mathrm{poly}(n)$-size circuit with an OR gate on top. Using the distributive law we can convert $\mathrm{Th}^j_i \wedge \mathrm{Th}^{k-j}_{k/2-i}$ to also be a depth-3 $\mathrm{poly}(n)$-size circuit with an OR gate on top. This OR gate can be collapsed with the top $j/2$-wise OR gate, and we have a polynomial-size monotone depth-3 circuit for slice_χ. ■

Note that even though we just proved a super-polynomial strong SQ dimension lower bound on $\mathcal{M}$, the concept class $\mathcal{M}$ is easily PAC-learnable. The learner can output 1 for inputs with more than $k/2$ 1's in the first k positions, and output 0 for inputs with fewer than $k/2$ 1's. For the inputs with exactly $k/2$ 1's in the first k positions, it can run the parity learning algorithm over the first k variables [FS92, HSW92], and learn χ. Thus, even if we can prove a super-polynomial strong SQ dimension lower bound for monotone DNF formulas, it is unclear if the class is hard to learn in general.

Chapter 5

Learning Random Monotone DNF

We now turn our attention away from the hardness of learning and focus on learning monotone functions with succinct representations. Having shown the hardness of learning depth-3 monotone circuits in the statistical query model, we show a statistical query algorithm for learning *random* depth-2 monotone circuits.

5.1 Introduction

Learning polynomial-size DNF formulas from random examples is an outstanding open question in computational learning theory, dating back more than 20 years to Valiant's introduction of the PAC learning model [Val84]. The most intensively studied variant of the DNF formula learning problem is PAC learning DNF formulas under the uniform distribution. Despite much effort, no polynomial-time algorithms are known for this problem.

A tantalizing question that has been posed as a goal by many authors (see *e.g.,* [Jac97, JT97, BBL98, Blu03, Ser04]) is to learn *monotone* DNF formulas, which only contain unnegated Boolean variables, under the uniform distribution. Besides being a natural restriction of the uniform distribution DNF formula learning problem, this problem is interesting because several impediments to learning general DNF formulas under uniform – known lower bounds for statistical query based algorithms [BFJ$^+$94], the apparent hardness of learning the subclass of $\log(n)$-juntas [Blu03] – do not apply in the monotone case. We solve a natural average-case version of this problem using previously unknown Fourier properties of monotone functions.

Previous work. Many partial results have been obtained on learning monotone DNF formula under the uniform distribution. Verbeurgt [Ver90] gave an $n^{O(\log n)}$-time uniform distribution algorithm for learning any poly(n)-term DNF formula, monotone or not. Several authors [KMSP94, SM00, BT06] have given results on learning monotone t-term DNF formulas for larger and larger values of t; most recently, Servedio [Ser04] gave a uniform distribution algorithm that learns any $2^{O(\sqrt{\log n})}$-term monotone DNF formula to any constant accuracy $\epsilon = \Theta(1)$ in poly(n) time. O'Donnell and Servedio [OS07] have recently shown that poly(n)-leaf *decision trees* that compute monotone functions (a subclass of poly(n)-term monotone DNF formulas) can be learned to any constant accuracy under uniform in poly(n) time. Various other problems related to learning different types of monotone functions under uniform have also been studied, see *e.g.,* [KLV94, BBL98, Ver98, HM91, AM02].

Aizenstein and Pitt [AP95] first proposed a model of random DNF formulas and gave an exact learning algorithm that learns random DNF formulas generated in this way. As noted in [AP95] and [JS05b], this model admits a trivial learning algorithm in the uniform PAC setting. Jackson and Servedio [JS05a] gave a uniform distribution algorithm that learns log-depth decision trees on average in a natural random model. Previous work on average-case uniform PAC DNF formula learning, also by Jackson and Servedio, is described below.

Our results. The main result of this section is a polynomial-time algorithm that can learn random poly(n)-term monotone DNF formulas with high probability. (We give a full description of the exact probability distribution defining our random DNF formulas in Section 5.4; briefly, the reader should think of our random t-term monotone DNF formulas as being obtained by independently drawing t monotone conjunctions uniformly from the set of all conjunctions of length $\log t$ over variables $x_1, \ldots, x_n$. Although many other distributions could be considered, this seems a natural starting point. Some justification for the choice of term length is given in Sections 5.4 and 5.6.)

Theorem 36. **[Informally]** *Let $t(n)$ be any function such that $t(n) \leq \text{poly}(n)$, and let $c > 0$ be any fixed constant. Then random monotone $t(n)$-term DNF formulas are PAC* *learnable (with failure probability $\delta = n^{-c}$) to accuracy ϵ in* $\mathrm{poly}(n, 1/\epsilon)$ *time under the uniform distribution. The algorithm outputs a monotone DNF formula as its hypothesis.*

In independent and concurrent work, Sellie [Sel08] has given an alternate proof of this theorem using different techniques. Sellie's result was recently generalized to non-monotone DNF formulas as well [Sel09].

Our technique. Jackson and Servedio [JS05b] showed that for any $\gamma > 0$, a result similar to Theorem 36 holds for random t-term monotone DNF formulas with $t \leq n^{2-\gamma}$. The main open problem stated in their paper was to prove Theorem 36. Our work solves this problem by using the previous algorithm to handle $t \leq n^{3/2}$, developing new Fourier lemmas for monotone DNF formulas, and using these lemmas together with more general versions of techniques from [JS05b] to handle $t \geq n^{3/2}$.

The crux of our strategy is to establish a connection between the term structure of certain monotone DNF formulas and their low-order Fourier coefficients. There is an extensive body of research on Fourier properties of monotone Boolean functions [BT06, MO03, BBL98], polynomial-size DNF formulas [Jac97, Man94], and related classes such as constant-depth circuits and decision trees [LMN93, KM93, JKS02, OS07]. These results typically establish that *every* function in the class has a Fourier spectrum with certain properties; unfortunately, the Fourier properties that have been obtainable to date for general statements of this sort have not been sufficient to yield polynomial-time learning algorithms.

We take a different approach by carefully defining a set of conditions, and showing that *if a monotone DNF formula f satisfies these conditions then the structure of the terms of f will be reflected in the low-order Fourier coefficients of f.* In the paper by Jackson and Servedio [JS05b], the degree two Fourier coefficients were shown to reveal the structure of the terms for certain (including random) monotone DNF formulas having at most $n^{2-\gamma}$ terms. In this work we develop new lemmas about the Fourier coefficients of more general monotone DNF formulas, and use these new lemmas to establish a connection between term structure and constant degree Fourier coefficients for monotone DNF formulas with any polynomial number of terms. Roughly speaking, this connection holds for monotone DNF formulas that satisfy the following conditions:

- each term has a reasonably large fraction of assignments which satisfy it and no other term;
- for each small tuple of distinct terms, only a small fraction of assignments simultaneously satisfy all terms in the tuple; and
- for each small tuple of variables, only a small number of terms contains the entire tuple.

The "small" tuples referred to above should be thought of as tuples of constant size. The constant degree coefficients capture the structure of the terms in the following sense: tuples of variables that all co-occur in some term will have a large magnitude Fourier coefficient, and tuples of variables that do not all co-occur in some term will have a small magnitude Fourier coefficient (even if subsets of the tuple do co-occur in some terms). We show this in Section 5.2.

Next we show a reconstruction procedure for obtaining the monotone DNF formula from tuple-wise co-occurrence information. Given a hypergraph with a vertex for each variable, the procedure turns each co-occurrence into a hyperedge, and then searches for all hypercliques of size corresponding to the term length. The hypercliques that are found correspond to the terms of the monotone DNF formula hypothesis that the algorithm constructs. This procedure is described in Section 5.3; we show that it succeeds in constructing a high-accuracy hypothesis if the monotone DNF formula f satisfies a few additional conditions. This is a significant generalization of a reconstruction procedure from the paper by Jackson and Servedio [JS05b] that was based on finding cliques in a graph (in the $n^{2-\gamma}$-term DNF formula setting, the algorithm deals only with co-occurrences of pairs of variables so it is sufficient to consider only ordinary graphs rather than hypergraphs).

The ingredients described so far thus give us an efficient algorithm to learn any monotone DNF formula that satisfies all of the required conditions. Finally, we show that random monotone DNF formulas satisfy all the required conditions with high probability. We do this in Section 5.4 via a fairly delicate probabilistic argument. Section 5.5 combines the above ingredients to prove Theorem 36. We close the chapter by showing that our technique lets us easily recapture the result of Hancock and Mansour [HM91] that read-k monotone

DNF formulas are learnable in polynomial time under the uniform distribution.

5.1.1 Preliminaries

We use capital letters for subsets of $[n]$. We will use calligraphic letters such as $\mathcal{C}$ to denote sets of sets and script letters such as $\mathscr{X}$ to denote sets of sets of sets. We write U_n to denote the uniform distribution over the Boolean cube $\{0,1\}^n$.

Recall that every monotone Boolean function has a unique representation as a reduced monotone DNF formula. We say that a term T of such a monotone DNF formula is *uniquely satisfied* by input x if x satisfies T and no other term of f.

Our learning model is an "average-case" variant of the well-studied uniform distribution PAC learning model. Let $\mathcal{D}_{\mathcal{C}}$ be a probability distribution over some fixed class $\mathcal{C}$ of Boolean functions over $\{0,1\}^n$, and let f (drawn from $\mathcal{D}_{\mathcal{C}}$) be an unknown target function. A learning algorithm A for $\mathcal{D}_{\mathcal{C}}$ takes as input an accuracy parameter $0 < \epsilon < 1$ and a confidence parameter $0 < \delta < 1$. We say that A *learns* $\mathcal{D}_{\mathcal{C}}$ *under* U_n if for every $0 < \epsilon, \delta < 1$, with probability at least $1 - \delta$ (over both the random examples used for learning and the random draw of f from $\mathcal{D}_{\mathcal{C}}$) algorithm A outputs a hypothesis h which has error at most ϵ.

5.2 Fourier Coefficients and the Term Structure of Monotone DNF

Throughout Section 5.2 let $f(x_1, \ldots, x_n)$ be a monotone DNF formula and let $S \subseteq \{1, \ldots, n\}$ be a fixed subset of variables. We write s to denote $|S|$ throughout this section. The Fourier coefficient, written $\hat{f}(S)$, measures the correlation between f and the parity of the variables in S.

The main result of this section is Lemma 3, which shows that under suitable conditions on f, the value $|\hat{f}(S)|$ is "large" if and only if f has a term containing all the variables of S. To prove this, we observe that the inputs which uniquely satisfy such a term will make a certain contribution to $\hat{f}(S)$. (In Section 5.2.1 we explain this in more detail and show how to view $\hat{f}(S)$ as a sum of contributions from inputs to f.) It remains then to show that the contribution from other inputs is small. The main technical novelty comes in Sections 5.2.2

and 5.2.3, where we show that all other inputs which make a contribution to $\hat{f}(S)$ must satisfy the terms of f in a special way, and use this property to show that under suitable conditions on f, the fraction of such inputs must be small.

5.2.1 Rewriting $\hat{f}(S)$.

We observe that $\hat{f}(S)$ can be expressed in terms of 2^s conditional probabilities, each of which is the probability that f is satisfied conditioned on a particular setting of the variables in S. That is:

$$\begin{aligned}\hat{f}(S) &\overset{\text{def}}{=} \mathop{\mathbb{E}}_{x\in U^n}\left[(-1)^{\sum_{i\in S} x_i}\cdot f(x)\right] = \frac{1}{2^n}\sum_{x\in\{0,1\}^n}(-1)^{\sum_{i\in S} x_i}\cdot f(x)\\ &= \frac{1}{2^n}\sum_{U\subseteq S}(-1)^{|U|}\sum_{x\in Z_S(U)} f(x) = \frac{1}{2^s}\sum_{U\subseteq S}(-1)^{|U|}\Pr_x[f(x)=1 \mid x\in Z_S(U)],\end{aligned}$$

where $Z_S(U)$ denotes the set of those $x\in\{0,1\}^n$ such that $x_i = 1$ for all $i\in U$ and $x_i = 0$ for all $i \in S\setminus U$. If f has some term T containing all the variables in S, then $\Pr_x[f(x) = 1 \mid x\in Z_S(S)]$ is at least as large as $\Pr_x[T$ is uniquely satisfied in $f \mid x\in Z_S(S)]$. On the other hand, if f has no such term, then $\Pr_x[f(x) = 1 \mid x \in Z_S(S)]$ does not receive this contribution. We will show that this contribution is the chief determinant of the magnitude of $\hat{f}(S)$.

It is helpful to rewrite $\hat{f}(S)$ as a sum of contributions from each input $x\in\{0,1\}^n$. To this end, we decompose f according to the variables of S. Given a subset $U\subseteq S$, we will write g_U to denote the disjunction of terms in f that contain every variable indexed by $U\subseteq S$ and no variable indexed by $S\setminus U$, but with the variables indexed by U removed from each term. (So for example if $f = x_1x_2x_4x_6 \vee x_1x_2x_5 \vee x_1x_2x_3 \vee x_3x_5 \vee x_1x_5x_6$ and $S = \{1,2,3\}$ and $U=\{1,2\}$, then $g_U = x_4x_6 \vee x_5$.) Thus we can split f into disjoint sets of terms: $f = \bigvee_{U\subseteq S}(t_U \wedge g_U)$, where t_U is the term consisting of exactly the variables indexed by U.

Suppose we are given $U\subseteq S$ and an x that belongs to $Z_S(U)$. We have that $f(x) = 1$ if and only if $g_{U'}(x)$ is true for some $U'\subseteq U$. (Note that $t_{U'}(x)$ is true for every $U'\subseteq U$ since x belongs to $Z_S(U)$.) Thus we can rewrite the Fourier coefficients $\hat{f}(S)$ as follows: (Below we write $\mathbf{1}(P)$ to denote the indicator function that takes value 1 if predicate P is true and value 0 if P is false.)

$$\begin{aligned}\hat{f}(S) &= \frac{1}{2^n}\sum_{U\subseteq S}(-1)^{|U|}\sum_{x\in Z_S(U)} f(x) = \sum_{U\subseteq S}(-1)^{|U|}\frac{1}{2^n}\sum_{x\in Z_S(U)}\mathbf{1}\left(\bigvee_{U'\subseteq U} g_{U'}(x)\right)\\ &= \sum_{U\subseteq S}(-1)^{|U|}\frac{1}{2^s}\frac{1}{2^n}\sum_{x\in\{0,1\}^n}\mathbf{1}\left(\bigvee_{U'\subseteq U} g_{U'}(x)\right)\\ &= \sum_{x\in\{0,1\}^n}\frac{1}{2^s}\frac{1}{2^n}\sum_{U\subseteq S}(-1)^{|U|}\mathbf{1}\left(\bigvee_{U'\subseteq U} g_{U'}(x)\right).\end{aligned}$$

We can rewrite this as:

$$\hat{f}(S) = \sum_{x\in\{0,1\}^n}\mathrm{Con}_S(x), \quad \text{where} \quad \mathrm{Con}_S(x) \overset{\text{def}}{=} \frac{1}{2^s}\frac{1}{2^n}\sum_{U\subseteq S}(-1)^{|U|}\mathbf{1}\left(\bigvee_{U'\subseteq U} g_{U'}(x)\right). \quad (5.1)$$

The value $\mathrm{Con}_S(x)$ may be viewed as the "contribution" that x makes to $\hat{f}(S)$. Recall that when f has a term T which contains all the variables in S, those $x \in Z_S(S)$ which uniquely satisfy T will contribute to $\hat{f}(S)$. We will show that under suitable conditions on f, the other x's make little or no contribution.

5.2.2 Bounding the Contribution to $\hat{f}(S)$ from Various Inputs.

The variable $\mathcal{C}$ will denote a subset of $\mathcal{P}(S)$, the power set of S; *i.e.,* $\mathcal{C}$ denotes a collection of subsets of S. We may view $\mathcal{C}$ as defining a set of g_U's (those g_U's for which U belongs to $\mathcal{C}$).

We may partition the set of inputs $\{0,1\}^n$ into $2^{|\mathcal{P}(S)|} = 2^{2^s}$ parts according to what subset of the 2^s functions $\{g_U\}_{U\subseteq S}$ each $x \in \{0,1\}^n$ satisfies. For $\mathcal{C}$ a subset of $\mathcal{P}(S)$ we denote the corresponding piece of the partition by $P_{\mathcal{C}}$; so $P_{\mathcal{C}}$ consists of precisely those $x \in \{0,1\}^n$ that satisfy $\left(\bigwedge_{U\in\mathcal{C}} g_U\right) \wedge \left(\bigwedge_{U\notin\mathcal{C}} \overline{g}_U\right)$. Note that for any given fixed $\mathcal{C}$, each x in $P_{\mathcal{C}}$ has exactly the same contribution $\mathrm{Con}_S(x)$ to the Fourier coefficient $\hat{f}(S)$ as every other x' in $P_{\mathcal{C}}$; this is simply because x and x' will satisfy exactly the same set of $g_{U'}$'s in (5.1). More generally, we have the following:

Lemma 37. *Let $\mathcal{C}$ be any subset of $\mathcal{P}(S)$. Suppose that there exist $U_1, U_2 \in \mathcal{C}$ such that $U_1 \subsetneq U_2$. Then for any y, z where $y \in P_{\mathcal{C}}$ and $z \in P_{\mathcal{C}\setminus U_2}$, we have that:* $\mathrm{Con}_S(y) = \mathrm{Con}_S(z)$.

Proof. Consider Equation 5.1. Fix any subset U of S. We shall show that the indicator variable $\mathbf{1}\left(\bigvee_{U'\subseteq U} g_{U'}(x)\right)$ takes the same value on y and on z.

Recall that y satisfies precisely those g_r's such that $r \in \mathcal{C}$, and z satisfies precisely those g_r's such that $r \in (\mathcal{C} \setminus U_2)$. We have that:

1. $\bigvee_{U'\subseteq U} g_{U'}(y)$ is true if and only if there exists some $U' \subseteq U$, $U' \in \mathcal{C}$; and

2. $\bigvee_{U'\subseteq U} g_{U'}(z)$ is true if and only if there exists some $U'' \subseteq U$, $U'' \in (\mathcal{C} \setminus U_2)$.

Since $U_1 \subsetneq U_2$ and $U_1, U_2 \in \mathcal{C}$, there exists a U' as described above if and only if there exists a U'' as described above. ■

Given a collection $\mathcal{C}$ of subsets of S, we write $\mathrm{Con}_S(\mathcal{C})$ to denote $\sum_{x\in P_{\mathcal{C}}} \mathrm{Con}_S(x)$, and we refer to this quantity as the contribution that $\mathcal{C}$ makes to the Fourier coefficient $\hat{f}(S)$. It is clear that we have $\hat{f}(S) = \sum_{\mathcal{C}\subseteq\mathcal{P}(S)} \mathrm{Con}_S(\mathcal{C})$.

The following lemma establishes a broad class of $\mathcal{C}$'s for which $\mathrm{Con}_S(\mathcal{C})$ is zero:

Lemma 38. *Let $\mathcal{C}$ be any collection of subsets of S. If $\bigcup_{U\in\mathcal{C}} U \neq S$ then $\mathrm{Con}_S(x) = 0$ for each $x \in P_{\mathcal{C}}$ and hence $\mathrm{Con}_S(\mathcal{C}) = 0$.*

Before proving Lemma 38 we first introduce some notation and make a few easy observations. Let $\mathrm{odd}(U) \subset \mathcal{P}(S)$ be the set of all the odd-sized subsets of S that are supersets of U, and let $\mathrm{even}(U)$ be defined similarly. For any $U \subsetneq S$ we have $|\mathrm{odd}(U)| = |\mathrm{even}(U)|$ since there are exactly $2^{|S|-|U|}$ subsets of S containing U, half of which are even and half of which are odd. Note that if U is the entire set S, then S is the only superset of U, and of course $|S|$ cannot be both even and odd. Finally, note that given subsets $U_1, \ldots, U_k$ of S, we have:

$$\cap_{i=1}^{k} \mathrm{odd}(U_i) = \mathrm{odd}(\cup_{i=1}^{k} U_i). \tag{5.2}$$

(This just says that the intersection of the $\mathrm{odd}(U_i)$'s is equal to the set of all odd subsets of S that contain the union of all the U_i's.) A similar equality $\cap_{i=1}^{k}\mathrm{even}(U_i) = \mathrm{even}(\cup_{i=1}^{k} U_i)$ also holds.

Now we can give the proof:

Proof of Lemma 38. We know that each x in $P_{\mathcal{C}}$ makes the same contribution to $\hat{f}(S)$. So fix any $x \in P_{\mathcal{C}}$; it suffices to show that the quantity $\sum_{U \subseteq S}(-1)^{|U|}\mathbf{1}\left(\bigvee_{U' \subseteq U} g_{U'}(x)\right)$ is zero. This quantity will be zero if x satisfies an equal number of $\bigvee_{U' \subseteq U} g_{U'}(x)$ for which $|U|$ is even, and for which $|U|$ is odd. The U for which x satisfies $\bigvee_{U' \subseteq U} g_{U'}(x)$ are the U for which there exist some $U' \in \mathcal{C}$ such that $U' \subseteq U$. Thus, we need to count the number of even and odd-sized $U \subseteq S$ containing some $U' \in \mathcal{C}$, and show that $|\cup_{U' \in \mathcal{C}} \text{odd}(U')| = |\cup_{U' \in \mathcal{C}} \text{even}(U')|$. Let $\mathcal{C} = \{U_1, \dots, U_k\} \subseteq \mathcal{P}(\mathcal{S})$. By inclusion-exclusion,

$$\left|\cup_{U' \in \mathcal{C}} \text{odd}(U')\right| = \sum_{i=1}^{k} |\text{odd}(U_i)| - \sum_{i_1,i_2} |\text{odd}(U_{i_1}) \cap \text{odd}(U_{i_2})| \dots + (-1)^{k-1} \left|\cap_{i=1}^{k} \text{odd}(U_i)\right|, \tag{5.3}$$

and we have a similar expression for $|\cup_{U' \in \mathcal{C}} \text{even}(U')|$ (identical to the RHS of (5.3) except with "even" everywhere in place of "odd").

By (5.2) we can rewrite each intersections of some odd(U_i)'s as odd$(\cup U_i)$, and similarly we can rewrite each intersection of some even(U_i)'s as even$(\cup U_i)$'s. Thus the RHS of (5.3) can be rewritten as a sum of $|\text{odd}(\cup U_i)|$'s, and similarly $|\cup_{U' \in \mathcal{C}} \text{even}(U')|$ can be rewritten as an identical sum of $|\text{even}(\cup U_i)|$'s. Since by assumption each of these $\cup U_i$'s cannot be the whole set S, for each $\cup U_i$ we have $|\text{odd}(\cup U_i)| = |\text{even}(\cup U_i)|$. Therefore all the terms of $|\cup_{U' \in \mathcal{C}} \text{odd}(U')|$ in (5.3) will match up with all the terms of $|\cup_{U' \in \mathcal{C}} \text{even}(U')|$. It follows that $|\cup_{U' \in \mathcal{C}} \text{odd}(U')|$ is indeed equal to $|\cup_{U' \in \mathcal{C}} \text{even}(U')|$, and the lemma is proved. ■

It remains to analyze those $\mathcal{C}$'s for which $\bigcup_{U \in \mathcal{C}} U = S$; for such a $\mathcal{C}$ we say that $\mathcal{C}$ *covers* S.

Recall from the previous discussion that $\text{Con}_S(\mathcal{C}) = |P_{\mathcal{C}}| \cdot \text{Con}_S(x)$ where x is any element of $P_{\mathcal{C}}$. Since $|\text{Con}_S(x)| \leq \frac{1}{2^n}$ for all $x \in \{0,1\}^n$, for any collection $\mathcal{C}$, we have that

$$|\text{Con}_S(\mathcal{C})| \quad \leq \quad \Pr_{x \in U_n}[x \in P_{\mathcal{C}}] \quad = \quad \Pr_{x \in U_n}[(\bigwedge_{U \in \mathcal{C}} g_U) \wedge (\bigwedge_{U \notin \mathcal{C}} \overline{g}_s)] \quad \leq \quad \Pr_{x \in U_n}[(\bigwedge_{U \in \mathcal{C}} g_U)].$$

We are interested in bounding this probability for $\mathcal{C} \neq \{S\}$ (we will deal with the special case $\mathcal{C} = \{S\}$ separately later). Recall that each g_U is a disjunction of terms; the expression $\bigwedge_{U \in \mathcal{C}} g_U$ is satisfied by precisely those x that satisfy at least one term from each g_U as U ranges over all elements of $\mathcal{C}$. For $j \geq 1$ let us define a quantity B_j as follows

$$B_j \overset{\text{def}}{=} \max_{i_1,\dots,i_j} \Pr_{x \in U_n}[x \text{ simultaneously satisfies terms } T_{i_1}, \dots, T_{i_j} \text{ in } \vee_{U \subseteq S}(g_U)]$$

where the max is taken over all j-tuples of distinct terms in $\vee_{U \subseteq S}(g_U)$. Then it is not hard to see that by a union bound, we have

$$|\mathrm{Con}_S(\mathcal{C})| \leq B_{|\mathcal{C}|} \prod_{U \in \mathcal{C}} (\#g_U), \tag{5.4}$$

where $\#g_U$ denotes the number of terms in the monotone DNF formula g_U.

The idea of why (5.4) is a useful bound is as follows. Intuitively, one would expect that the value of B_j decreases as j (the number of terms that must be satisfied) increases. One would also expect the value of $\#g_U$ to decrease as the size of U increases (if U contains more variables then fewer terms in f will contain all of those variables). This means that there is a trade-off which helps us bound (5.4): if $|\mathcal{C}|$ is large then $B_{|\mathcal{C}|}$ is small, but if $|\mathcal{C}|$ is small then (since we know that $\bigcup_{U \in \mathcal{C}} U = S$) some U is large and so $\prod_{U \in \mathcal{C}} \#g_U$ will be smaller.

5.2.3 Bounding $\hat{f}(S)$ Based on Whether S Co-occurs in Some Term of f.

We are now ready to state formally the conditions on $\hat{f}$ that allow us to detect a co-occurrence of variables in the value of the corresponding Fourier coefficient.

Lemma 39. *Let $f : \{0,1\}^n \to \{-1,1\}$ be a monotone DNF formula. Fix a set $S \subset [n]$ of size $|S| = s$ and let*

$$\mathscr{Y} = \{\mathcal{C} \subseteq \mathcal{P}(S) : \mathcal{C} \text{ covers } S \text{ and } S \notin \mathcal{C}\}.$$

Suppose that we define $\alpha, \beta_1, \ldots, \beta_{2^s}$ and $\Phi : \mathscr{Y} \to \mathbb{R}$ so that:

(C1) *Each term in f is uniquely satisfied with probability at least α;*

(C2) *For $1 \leq j \leq 2^s$, each j-tuple of terms in f is simultaneously satisfied with probability at most β_j; and*

(C3) *For every $\mathcal{C}_{\mathscr{Y}} \in \mathscr{Y}$ we have $\prod_{U \in \mathcal{C}_{\mathscr{Y}}} (\#g_U) \leq \Phi(\mathcal{C}_{\mathscr{Y}})$.*

Then,

1. *If the variables in S do not simultaneously co-occur in any term of f, then*

$$|\hat{f}(S)| \leq \Upsilon \quad \text{where} \quad \Upsilon := \sum_{\mathcal{C}_{\mathscr{Y}} \in \mathscr{Y}} \left(2^s \beta_{|\mathcal{C}_{\mathscr{Y}}|} \Phi(\mathcal{C}_{\mathscr{Y}})\right);$$

2. If the variables in S do simultaneously co-occur in some term of f, then $|\hat{f}(S)| \geq \frac{\alpha}{2^s} - 2 \cdot \Upsilon$.

Using Lemma 39, if f satisfies conditions **(C1)** through **(C3)** with values of β_j and $\Phi(\cdot)$ so that there is a "gap" between $\alpha/2^s$ and 3Υ, then we can determine whether all the variables in S simultaneously co-occur in a term by estimating the magnitude of $\hat{f}(S)$.

Proof. Let $\mathcal{C}^\star$ denote the 'special' element of $P(S)$ that consists solely of the subset S, *i.e.*, $\mathcal{C}^\star = \{S\}$, and let $\mathscr{X} = \{\mathcal{C} \subseteq \mathcal{P}(S) : \mathcal{C}$ covers S and $S \in \mathcal{C}$ and $\mathcal{C} \neq \mathcal{C}^\star\}$. Using Lemma 38, we have

$$\hat{f}(S) = \mathrm{Con}_S(\mathcal{C}^\star) + \sum_{\mathcal{C}_\mathscr{Y} \in \mathscr{Y}} \mathrm{Con}_S(\mathcal{C}_\mathscr{Y}) + \sum_{\mathcal{C}_\mathscr{X} \in \mathscr{X}} \mathrm{Con}_S(\mathcal{C}_\mathscr{X}). \tag{5.5}$$

We first prove point 1. Suppose that the variables of S do not simultaneously co-occur in any term of f. Then g_S is the empty disjunction and $\#g_S = 0$, so $\mathrm{Con}_S(\mathcal{C}) = 0$ for any $\mathcal{C}$ containing S. Thus in this case we have $\hat{f}(S) = \sum_{\mathcal{C}_\mathscr{Y} \in \mathscr{Y}} \mathrm{Con}_S(\mathcal{C}_\mathscr{Y})$; using (5.4) and condition **(C3)**, it follows that $|\hat{f}(S)|$ is at most $\sum_{\mathcal{C}_\mathscr{Y} \in \mathscr{Y}} B_{|\mathcal{C}_\mathscr{Y}|} \Phi(\mathcal{C}_\mathscr{Y})$.

We now claim that $B_{|\mathcal{C}_\mathscr{Y}|} \leq 2^s \beta_{|\mathcal{C}_\mathscr{Y}|}$; we establish this by showing that $B_j \leq 2^s \beta_j$ for all j. In other words, we shall bound the probability of simultaneously satisfying any fixed collection of j terms in the DNF formula $f' = \vee_{U \subseteq S}(g_U)$. We have that for $1 \leq j \leq 2^s$, each j-tuple of terms in f is simultaneously satisfied with probability at most β_j. Consider any fixed sequence $T'_{i_1}, \ldots, T'_{i_j}$ of terms from f'. Let $T_{i_1}, \ldots, T_{i_j}$ denote the sequence of terms in f from which the terms $T'_{i_1}, \ldots, T'_{i_j}$ were derived, *i.e.*, each term T consists of $T' \wedge (\wedge_{i \in U} x_i)$ for some $U \subseteq S$. Since $T_{i_1} \wedge \cdots \wedge T_{i_j}$ is simply a monotone conjunction and $T'_{i_1} \wedge \cdots \wedge T'_{i_j}$ is simply the corresponding conjunction obtained by removing at most $|S| = s$ variables from $T_{i_1} \wedge \cdots \wedge T_{i_j}$, we have that $\Pr_x[T'_{i_1} \wedge \cdots \wedge T'_{i_j}] \leq 2^s \beta_j$.

So in this case we have:

$$|\hat{f}(S)| \leq \sum_{\mathcal{C}_\mathscr{Y} \in \mathscr{Y}} |\mathrm{Con}_S(\mathcal{C}_\mathscr{Y})| \leq \sum_{\mathcal{C}_\mathscr{Y} \in \mathscr{Y}} B_{|\mathcal{C}_\mathscr{Y}|} \Phi(\mathcal{C}_\mathscr{Y}) \leq \sum_{\mathcal{C}_\mathscr{Y} \in \mathscr{Y}} \left(2^s \beta_{|\mathcal{C}_\mathscr{Y}|} \Phi(\mathcal{C}_\mathscr{Y})\right) = \Upsilon.$$

Now we turn to point 2. Suppose that the variables of S do co-occur in some term of f. Let x be any element of $P_{\mathcal{C}^\star}$, so x satisfies g_U if and only if $U = S$. It is easy to see from (5.1) that for such an x we have $\mathrm{Con}_S(x) = (-1)^{|S|}/(2^n 2^s)$. We thus have that

$$\mathrm{Con}_S(\mathcal{C}^\star) = \frac{(-1)^{|S|}}{2^s} \cdot \Pr[x \in P_{\mathcal{C}^\star}] = \frac{(-1)^{|S|}}{2^s} \Pr[g_S \wedge (\bigwedge_{U \subsetneq S} \overline{g}_U)]. \tag{5.6}$$

Since S co-occurs in some term of f, we have that g_S contains at least one term T. By condition **(C1)**, the corresponding term $(T \wedge (\wedge_{i \in S} x_i))$ of f is uniquely satisfied with probability at least α. Since each assignment that uniquely satisfies $(T \wedge (\wedge_{i \in S} x_i))$ (among all the terms of f) must satisfy $g_S \wedge (\bigwedge_{U \subsetneq S} \overline{g}_U)$, we have that the magnitude of (5.6) is at least $\alpha/2^s$.

We now show that $|\sum_{\mathcal{C}_{\mathscr{X}} \in \mathscr{X}} \mathrm{Con}_S(\mathcal{C}_{\mathscr{X}})| \leq \Upsilon$, which completes the proof, since we already have that $|\sum_{\mathcal{C}_{\mathscr{Y}} \in \mathscr{Y}} \mathrm{Con}_S(\mathcal{C}_{\mathscr{Y}})| \leq \sum_{\mathcal{C}_{\mathscr{Y}} \in \mathscr{Y}} |\mathrm{Con}_S(\mathcal{C}_{\mathscr{Y}})| \leq \Upsilon$. First note that if the set $\mathcal{C}_{\mathscr{X}} \setminus \{S\}$ does not cover S, then by Lemmas 37 and 38 we have that $\mathrm{Con}_S(x) = 0$ for each $x \in P_{\mathcal{C}_{\mathscr{X}}}$ and thus $\mathrm{Con}_S(\mathcal{C}_{\mathscr{X}}) = 0$. So we may restrict our attention to those $\mathcal{C}_{\mathscr{X}}$ such that $\mathcal{C}_{\mathscr{X}} \setminus \{S\}$ covers S. Now since such a $\mathcal{C}_{\mathscr{X}} \setminus \{S\}$ is simply some $\mathcal{C}_{\mathscr{Y}} \in \mathscr{Y}$, and each $\mathcal{C}_{\mathscr{Y}} \in \mathscr{Y}$ is obtained as $\mathcal{C}_{\mathscr{X}} \setminus \{S\}$ for at most one $\mathcal{C}_{\mathscr{X}} \in \mathscr{X}$, we have:

$$\left| \sum_{\mathcal{C}_{\mathscr{X}} \in \mathscr{X}} \mathrm{Con}_S(\mathcal{C}_{\mathscr{X}}) \right| \leq \sum_{\mathcal{C}_{\mathscr{Y}} \in \mathscr{Y}} |\mathrm{Con}_S(\mathcal{C}_{\mathscr{Y}})| \leq \Upsilon.$$

■

5.3 Hypothesis Formation

In this section, we show that if a target monotone DNF formula f satisfies the conditions of Lemma 39 and two other simple conditions stated below (see Theorem 40), then it is possible to learn f from uniform random examples.

Theorem 40. *Let f be a t-term monotone DNF formula. Fix $s \in [n]$. Suppose that*

- *For all sets $S \subset [n], |S| = s$, conditions* **(C1)** *through* **(C3)** *of Lemma 39 hold for certain values α, β_j, and $\Phi(\cdot)$ satisfying $\Delta > 0$, where $\Delta := \alpha/2^s - 3 \cdot \Upsilon$. (Recall that $\Upsilon := \sum_{\mathcal{C}_{\mathscr{Y}} \in \mathscr{Y}} \left(2^s \beta_{|\mathcal{C}_{\mathscr{Y}}|} \Phi(\mathcal{C}_{\mathscr{Y}})\right)$, where $\mathscr{Y} = \{\mathcal{C} \subseteq \mathcal{P}(S) : \mathcal{C}$ covers S and $S \notin \mathcal{C}\}$.)*

(C4) *Every set S of s co-occurring variables in f appears in at most γ terms (here $\gamma \geq 2$); and*

(C5) *Every term of f contains at most κ variables (note that $s \leq \kappa \leq n$).*

Then $\mathcal{A}$ (see Figure 5.1) PAC learns f to accuracy ϵ with confidence $1 - \delta$ given access to $EX(f, U_n)$, and runs in time poly$(n^{s+\gamma}, t, 1/\Delta, \gamma^\kappa, 1/\epsilon, \log(1/\delta))$.

Algorithm $\mathcal{A}$ (inputs are $\epsilon, \delta, s, \alpha, \Upsilon, \gamma, \kappa$, and access to $EX(f, U_n)$)

1. Define $\Delta := \alpha/2^s - 3 \cdot \Upsilon$.

2. For each $S \subset [n], |S| = s$, empirically estimate $\hat{f}(S)$ to within $\pm\Delta/3$ (with confidence $1 - \delta/3$ that all estimates have the required accuracy); let $\tilde{f}(S)$ be the empirical estimate thus obtained. Mark as "good" each S for which $|\tilde{f}(S)| \geq \Upsilon + \frac{\Delta}{2}$.

3. Let G_f denote the following n-vertex hypergraph: the vertices of G_f correspond to variables $x_1, \ldots, x_n$, and G_f contains each s-vertex hyperedge S if and only if S was marked as "good" in the previous step.

4. Run algorithm $\mathcal{A}'$ (see Figure 5.2) to identify the set HC_f of all of the k-hypercliques in G_f, as k ranges over $\{s, \ldots, \kappa\}$.

5. Run the standard elimination algorithm for disjunctions—with ϵ as the accuracy input parameter and $\delta/3$ as the confidence—over the "features" that are the monotone conjunctions corresponding to the hypercliques identified in the previous step. Output the resulting hypothesis h (which is a monotone DNF formula).

Figure 5.1: The Algorithm $\mathcal{A}$.

Proof. Lemma 39 implies that for each set $S \subset [n], |S| = s$,

- if the variables in S all co-occur in some term of f, then $|\hat{f}(S)|$ is at least $\Delta/2$ larger than $\Upsilon + \Delta/2$;

- if the variables in S do not all co-occur in some term of f, then $|\hat{f}(S)|$ is at least $\Delta/2$ smaller than $\Upsilon + \Delta/2$.

A straightforward application of Hoeffding bounds (to estimate the Fourier coefficients using a random sample of uniformly distributed examples) shows that Step 1 of Algorithm $\mathcal{A}$ can be executed in poly$(n^s, 1/\Delta, \log(1/\delta))$ time, and that with probability $1 - \delta/3$ the S's that are marked as "good" will be precisely the s-tuples of variables that co-occur in some term of f.

Algorithm $\mathcal{A}'$ (input is the list of "good" sets S identified in Step 1 of Algorithm $\mathcal{A}$)

1. For each good set S, run Algorithm $\mathcal{A}''$ (see Figure 5.3) to identify the set N_S of all variables in $[n] \setminus S$ that occur in some term that also contains all variables in S.

2. For all $s \leq k \leq \kappa$, using brute-force search over all subsets N' of at most $(k - s)$ many elements from N_S, check whether $N' \cup S$ is a k-hyperclique in G_f.

Figure 5.2: The Algorithm $\mathcal{A}'$.

Algorithm $\mathcal{A}''$ (input is a good set S)

1. For each subset N of at most γ variables from $[n] \setminus S$, perform the following:

   (a) Empirically estimate $\widehat{f_{N\leftarrow 0}}(S)$ to additive accuracy $\pm\Delta/3$; let $\widetilde{f_{N\leftarrow 0}}(S)$ be the empirical estimate thus obtained. Mark each N for which $\widetilde{f_{N\leftarrow 0}}(S) \geq \Upsilon + \frac{\Delta}{2}$.

2. Let N_S be the union of all the N's that were marked in the previous step. Return N_S.

Figure 5.3: The Algorithm $\mathcal{A}''$.

Conceptually, the algorithm next constructs the hypergraph G_f that has one vertex per variable in f and that includes an s-vertex hyperedge if and only if the corresponding s variables co-occur in some term of f. Clearly there is a k-hyperclique in G_f for each term of k variables in f. So if we could find all of the k-hypercliques in G_f (where again k ranges between s and κ), then we could create a set HC_f of monotone conjunctions of variables such that f could be represented as an OR of t of these conjunctions. Treating each of the conjunctions in HC_f as a variable in the standard elimination algorithm for learning disjunctions (see *e.g.,* Chapter 1 of [KV94]) would then enable us to properly PAC learn f to accuracy ϵ with probability at least $1 - \delta/3$ in time polynomial in n, t, $|HC_f|$, $1/\epsilon$, and $\log(1/\delta)$. Thus, $\mathcal{A}$ will use a subalgorithm $\mathcal{A}'$ to find all of the k-hypercliques in G_f and will then apply the elimination algorithm over the corresponding conjunctions to learn the final approximator h.

We now explain the subalgorithm $\mathcal{A}'$ for locating the set HC_f of k-hypercliques. For each set S of s co-occurring variables, let $N_S \subseteq ([n] \setminus S)$ be defined as follows: a variable x_i is in N_S if and only if x_i is present in some term that contains all of the variables in S. Since by assumption there are at most γ terms containing such variables and each term contains at most κ variables, this means that $|N_S| < \kappa\gamma$. The subalgorithm will use this bound as follows. For each set S of s co-occurring variables, $\mathcal{A}'$ will determine the set N_S using a procedure $\mathcal{A}''$ described shortly. Then, for each $s \leq k \leq \kappa$ and each $(k-s)$-element subset N' of N_S, $\mathcal{A}'$ will test whether or not $N' \cup S$ is a k-hyperclique in G_f. The set of all k-hypercliques found in this way is HC_f. For each S, the number of sets tested in this process is at most

$$\sum_{i=0}^{\kappa} \binom{|N_S|}{i} \leq \sum_{i=0}^{\kappa} \binom{\kappa\gamma}{i} \leq \left(\frac{e\kappa\gamma}{\kappa}\right)^{\kappa} = (e\gamma)^{\kappa}.$$

Thus, $|HC_f| = O(n^s(e\gamma)^{\kappa})$, and this is an upper bound on the time required to execute Step 2 of subalgorithm $\mathcal{A}'$.

Finally, we need to define the procedure $\mathcal{A}''$ for finding N_S for a given set S of s co-occurring variables. Fix such an S and let N_γ be a set of at most γ variables in $([n] \setminus S)$ having the following properties:

(P1) In the projection $f_{N_\gamma \leftarrow 0}$ of f in which all of the variables of N_γ are fixed to 0, the variables in S do not co-occur in any term; and

(P2) For every set $N'_\gamma \subset N_\gamma$ such that $|N'_\gamma| = |N_\gamma| - 1$, the variables in S do co-occur in at least one term of $f_{N'_\gamma \leftarrow 0}$.

We will use the following claim:

Claim 41. *N_S is the union of all sets N_γ of cardinality at most γ that satisfy* **(P1)** *and* **(P2)**.

Proof. We first show that the union of all sets satisfying **(P1)** and **(P2)** is a subset of N_S. To see this, note that if variable x_i is not in N_S (*i.e.,* x_i does not co-occur with S in any term), then any set N_γ that includes x_i cannot satisfy both properties. This is because if

N_γ satisfies **(P1)** (*i.e.,* S does not co-occur in any term of $f_{N_\gamma \leftarrow 0}$), then the set $N'_\gamma = N_\gamma \backslash x_i$ will also be such that S do not co-occur in any term of $f_{N'_\gamma \leftarrow 0}$, since x does not co-occur with S in any term.

Next, consider the minimal monotone DNF representation D_f of the target f. Let D_{f_S} be the monotone DNF expression obtained from D_f by removing from D_f all terms in which the variables of S do not co-occur and then fixing all of the variables in S to 1. Since D_{f_S} has at most γ terms, there is an equivalent minimal CNF C_{f_S} in which each clause contains at most γ variables. For each clause C_i in C_{f_S}, the set of variables in C_i satisfies both **(P1)** and **(P2)**: setting all of the variables in C_i to 0 falsifies both C_{f_S} and D_{f_S} and therefore removes from f all terms in which the variables of S co-occur; but setting any proper subset of the variables in C_i to 0 does not falsify D_{f_S} and therefore leaves at least one term in f in which the variables of S co-occur. Furthermore, all of the variables in D_{f_S} are also relevant in C_{f_S}, so every variable in D_{f_S} appears in at least one clause of C_{f_S}. It follows that the union of the variables in the sets N_γ satisfying **(P1)** and **(P2)** is a superset of the set of variables in D_{f_S}, that is, the set N_S. ■

There are only $O(n^\gamma)$ possible candidate sets N_γ to consider, so our problem now reduces to the following: given a set N of at most γ variables, determine whether the variables in S co-occur in $f_{N\leftarrow 0}$.

Recall that since f satisfies the three conditions **(C1)**, **(C2)** and **(C3)**, Lemma 39 implies that $|\hat{f}(S)|$ is either at most Υ (if the variables in S do not co-occur in any term of f) or at least $\frac{\alpha}{2^s} - 2 \cdot \Upsilon$ (if the variables in S do co-occur in some term). We now claim that the function $f_{N\leftarrow 0}$ has this property as well: *i.e.,* $|\widehat{f_{N\leftarrow 0}}(S)|$ is either at most the same value Υ (if the variables in S do not co-occur in any term of $f_{N\leftarrow 0}$) or at least the same value $\frac{\alpha}{2^s} - 2\cdot\Upsilon$ (if the variables in S do co-occur in some term of $f_{N\leftarrow 0}$). To see this, observe that the function $f_{N\leftarrow 0}$ is just f with some terms removed. Since each term in f is uniquely satisfied with probability at least α (this is condition **(C1)**), the same must be true of $f_{N\leftarrow 0}$ since removing terms from f can only increase the probability of being uniquely satisfied for the remaining terms. Since each j-tuple of terms in f is simultaneously satisfied with probability at most β_j (this is condition **(C2)**), the same must be true for j-tuples of terms in $f_{N\leftarrow 0}$. Finally, for condition **(C3)**, the value of $\#g_U$ can only decrease in passing from

f to $f_{N\leftarrow 0}$. Thus, the upper bound of Υ that follows from applying Lemma 39 to f is also a legitimate upper bound when the lemma is applied to $|\widehat{f_{N\leftarrow 0}}(S)|$, and similarly the lower bound of $\frac{\alpha}{2^s} - 2 \cdot \Upsilon$ is also a legitimate lower bound when the lemma is applied to $f_{N\leftarrow 0}$. Therefore, for every $|N| \leq \gamma$, a sufficiently accurate (within $\Delta/2$) estimate of $\widehat{f_{N\leftarrow 0}}(S)$ (as obtained in Step 1 of subalgorithm $\mathcal{A}''$) can be used to determine whether or not the variables in S co-occur in any term of $f_{N\leftarrow 0}$.

To obtain the required estimate for $\widehat{f_{N\leftarrow 0}}$, observe that for a given set N, we can simulate a uniform example oracle for $f_{N\leftarrow 0}$ by filtering the examples from the uniform oracle for f so that only examples setting the variables in N to 0 are accepted. Since $|N| \leq \gamma$, the filter accepts with probability at least $1/2^\gamma$. A Hoeffding bound argument then shows that the Fourier coefficients $\widehat{f_{N\leftarrow 0}}(S)$ can be estimated (with probability of failure no more than a small fraction of δ) from an example oracle for f in time polynomial in n, 2^γ, $1/\Delta$, and $\log(1/\delta)$.

Algorithm $\mathcal{A}''$, then, estimates Fourier coefficients of restricted versions of f, using a sample size sufficient to ensure that all of these coefficients are sufficiently accurate over all calls to $\mathcal{A}''$ with probability at least $1 - \delta/3$. These estimated coefficients are then used by $\mathcal{A}''$ to locate the set N_S as just described. The overall algorithm $\mathcal{A}$ therefore succeeds with probability at least $1 - \delta$, and it is not hard to see that it runs in the time bound claimed. ■

Required parameters. In the above description of Algorithm $\mathcal{A}$, we assumed that it is given the values of $s, \alpha, \Upsilon, \gamma$, and κ. In fact it is not necessary to assume this; a standard argument gives a variant of the algorithm which succeeds without being given the values of these parameters.

The idea is simply to have the algorithm "guess" the values of each of these parameters, either exactly or to an adequate accuracy. The parameters s, γ and κ take positive integer values bounded by $\text{poly}(n)$. The other parameters α, Υ take values between 0 and 1; a standard argument shows that if approximate values α' and Υ' (that differ from the true values by at most $1/\text{poly}(n)$) are used instead of the true values, the algorithm will still succeed. Thus there are at most $\text{poly}(n)$ total possible settings for $(s, \gamma, \kappa, \alpha, \Upsilon)$ that need to be tried. We can run Algorithm $\mathcal{A}$ for each of these candidate parameter settings, and test the resulting hypothesis; when we find the "right" parameter setting, we will obtain a high-accuracy hypothesis (and when this occurs, it is easy to recognize that it has occurred, simply by testing each hypothesis on a new sample of random labeled examples). This parameter guessing incurs an additional polynomial factor overhead. Thus Theorem 40 holds true for the extended version of Algorithm $\mathcal{A}$ that takes only ϵ, δ as input parameters.

5.4 Random Monotone DNF

Let $\mathcal{M}_n^{t,k}$ be the probability distribution over monotone t-term DNF formulas induced by the following process: each term is independently and uniformly chosen at random from all $\binom{n}{k}$ monotone ANDs of size exactly k over $x_1, \ldots, x_n$.

Given a value of t, throughout this section we consider the $\mathcal{M}_n^{t,k}$ distribution where $k = \lfloor \log t \rfloor$ (we will relax this and consider a broader range of values for k in Section 5.6). To motivate this choice, consider a random draw of f from $\mathcal{M}_n^{t,k}$. If k is too large relative to t then a random $f \in \mathcal{M}_n^{t,k}$ will likely have $\Pr_{x \in U_n}[f(x) = 1] \approx 0$, and if k is too small relative to t then a random $f \in \mathcal{M}_n^{t,k}$ will likely have $\Pr_{x \in U_n}[f(x) = 1] \approx 1$; such functions are trivial to learn to high accuracy using either the constant-0 or constant-1 hypothesis. A straightforward analysis (see *e.g.,* [JS05b]) shows that for $k = \lfloor \log t \rfloor$ we have that $\mathbb{E}_{f \in \mathcal{M}_n^{t,k}}[\Pr_{x \in U_n}[f(x) = 1]]$ is bounded away from both 0 and 1, and thus we feel that this is an appealing and natural choice.

5.4.1 Probabilistic analysis.

In this section we will establish various useful probabilistic lemmas regarding random monotone DNF formulas of polynomially bounded size.

Assumptions: Throughout the rest of Section 5.4 we assume that $t(n)$ is any function such that $n^{3/2} \le t(n) \le \text{poly}(n)$. To handle the case when $t(n) \le n^{3/2}$, we will use the results from [JS05b]. Let $a(n)$ be such that $t(n) = n^{a(n)}$. For brevity we write t for $t(n)$ and a for $a(n)$ below, but the reader should keep in mind that a actually denotes a function $\frac{3}{2} \le a = a(n) \le O(1)$.

The first lemma provides a bound of the sort needed by condition **(C3)** of Lemma 39:

Lemma 42. *Let $|S| = s = \lfloor a \rfloor + 2$. Fix any $\mathcal{C}_{\mathscr{Y}} \in \mathscr{Y}$. Let $\delta_{\text{terms}} = n^{-\Omega(\log n)}$. With probability at least $1 - \delta_{\text{terms}}$ over the random draw of f from $\mathcal{M}_n^{t,k}$, we have that for some absolute constant c and all sufficiently large n,*

$$\prod_{U \in \mathcal{C}_{\mathscr{Y}}} (\#g_U) \leq c \cdot \frac{t^{|\mathcal{C}_{\mathscr{Y}}|-1} k^{2^s}}{\sqrt{n}}. \tag{5.7}$$

Proof. We prove the lemma assuming that that $\emptyset \notin \mathcal{C}_{\mathscr{Y}}$. This is sufficient because if $\emptyset \in \mathcal{C}_{\mathscr{Y}}$, then $\mathcal{C}'_{\mathscr{Y}} = \mathcal{C}_{\mathscr{Y}} \setminus \emptyset$ is still contained in $\mathscr{Y}$, and applying the result to $\mathcal{C}'_{\mathscr{Y}}$ gives that $\prod_{U \in \mathcal{C}'_{\mathscr{Y}}} (\#g_U) \leq c \cdot \frac{t^{|\mathcal{C}'_{\mathscr{Y}}|-1} k^{2^s}}{\sqrt{n}}$ with probability at least $1 - \delta_{\text{terms}}$. Since $\#g_\emptyset \leq t$ and $|\mathcal{C}'_{\mathscr{Y}}| = |\mathcal{C}_{\mathscr{Y}}| - 1$, the conclusion of the lemma holds for $\mathcal{C}_{\mathscr{Y}}$ as well.

Fix any $\emptyset \neq U \in \mathcal{C}_{\mathscr{Y}}$ (note that since $U \in \mathcal{C}_{\mathscr{Y}}$ we also have $U \neq S$, and hence $|U| \leq s - 1 = \lfloor a \rfloor + 1$.). Recall that f is chosen by picking each term T_i to be a uniformly chosen set of k distinct variables. The probability (over a random choice of f) that T_1 contains all the elements of U and none of the elements of $S \setminus U$ is $\binom{n-s}{k-|U|}/\binom{n}{k}$; let us write p_U to denote this quantity. Using the facts that $k = \Theta(\log n)$ and $1 \leq |U| < s = O(1)$, one can verify that:

$$\frac{1}{2} (k/n)^{|U|} \leq p_U \leq (k/n)^{|U|}. \tag{5.8}$$

Since each of the t terms of f is chosen independently, we have that $\#g_U$ is binomially distributed according to $B(t, p_U)$, so $t \cdot \frac{1}{2}(\frac{k}{n})^{|U|} \leq \mathbb{E}[\#g_U] = tp_U \leq t(\frac{k}{n})^{|U|}$. Now recall that the Chernoff bound gives that $\Pr[X \geq (1+\zeta)\,\mathbb{E}[X]] \leq e^{-\zeta^2 tp/3}$ where X is an independent and identically distributed random variable. For X drawn from $B(t,p)$ and taking $\zeta = 1$, we get:

$$\Pr\left[\#g_U > 2t\,(k/n)^{|U|}\right] \leq \Pr[\#g_U > 2tp_U] \leq \exp(-tp_U/3) \leq \exp(-t(k/n)^{|U|}/6). \tag{5.9}$$

Suppose first that $|U| \leq s - 2 = \lfloor a \rfloor$. If $|U| = 1$, then since $t \geq n^{3/2}$ we have that (5.9) is at most $\exp(-\sqrt{n}\log n)$. On the other hand, if $|U| > 1$ then $\lfloor a \rfloor \geq 2$, and since $t/n^{|U|} \geq t/n^{\lfloor a \rfloor} \geq 1$ we have that (5.9) $\leq \exp(-k^{|U|}/6) \leq n^{-\Omega(\log n)}$.

Now suppose that $|U| = s - 1$. In this case we use the following form of the Hoeffding bound (see *e.g.,* Exercise 4.1 in [MR95]): if $\zeta > 2e - 1$, then $\Pr[X > (1+\zeta)\,\mathbb{E}[X]] \leq 2^{-(1+\zeta)\,\mathbb{E}[X]}$ for X drawn from $B(t,p)$. Let ζ be such that $(1+\zeta)t(k/n)^{|U|} = t^{1/2a}$; note that this gives:

$$1+\zeta = t^{(1/(2a))-1}(n/k)^{|U|} = (\sqrt{n}/t)(n/k)^{|U|} \geq (\sqrt{n}/t)(n/k)^{a} = \sqrt{n}/\mathrm{polylog}(n) \gg 2e,$$

so we may indeed apply the Hoeffding bound for this choice of ζ. Using (5.8), we obtain

$$\Pr[\#g_U > t^{1/2a}] \leq \Pr[\#g_U > (1+\zeta)tp_U] \leq 2^{\frac{-t^{1/2a}}{2}} \leq 2^{-\sqrt{n}/2}.$$

Taking a union bound over all possible sets $U \neq \emptyset$ (at most $2^s = O(1)$ many possibilities), we have that with probability at least $1-\delta_{\text{terms}}$ over the draw of f, every such set $U \in \mathcal{C}_{\mathscr{Y}}$ satisfies:

- if $|U| \leq s-2$ then $\#g_U \leq 2t(k/n)^{|U|}$; and
- if $|U| = s-1$ then $\#g_U \leq t^{1/2a}$.

We henceforth assume the above conditions are satisfied, and now show that this gives the bound (5.7).

We partition $\mathcal{C}_{\mathscr{Y}}$ according to the size of U: let $\mathcal{C}^A_{\mathscr{Y}} = \{U \in \mathcal{C}_{\mathscr{Y}} : |U| = s-1\}$ and $\mathcal{C}^B_{\mathscr{Y}} = \mathcal{C}_{\mathscr{Y}} \setminus \mathcal{C}^A_{\mathscr{Y}} = \{U \in \mathcal{C}_{\mathscr{Y}} : |U| \leq s-2\}$. Then,

$$\prod_{U \in \mathcal{C}_{\mathscr{Y}}} (\#g_U) = \prod_{U \in \mathcal{C}^A_{\mathscr{Y}}} (\#g_U) \prod_{U \in \mathcal{C}^B_{\mathscr{Y}}} (\#g_U) \leq t^{|\mathcal{C}^A_{\mathscr{Y}}|/2a} \cdot (2t)^{|\mathcal{C}^B_{\mathscr{Y}}|} (k/n)^{\sum_{U \in \mathcal{C}^B_{\mathscr{Y}}} |U|}.$$

By definition of $\mathcal{C}_{\mathscr{Y}}$ we have that $\sum_{U \in \mathcal{C}_{\mathscr{Y}}} |U| \geq s$. Now if $|\mathcal{C}^A_{\mathscr{Y}}| = 0$, then we have

$$\prod_{U \in \mathcal{C}_{\mathscr{Y}}} (\#g_U) \leq (2t)^{|\mathcal{C}_{\mathscr{Y}}|}(k/n)^{\sum_{U \in \mathcal{C}_{\mathscr{Y}}} |U|} \leq (2t)^{|\mathcal{C}_{\mathscr{Y}}|}(k/n)^s = 2^{|\mathcal{C}_{\mathscr{Y}}|}\frac{t^{|\mathcal{C}_{\mathscr{Y}}|-1}k^s}{n^{s-a}} \leq 2^{|\mathcal{C}_{\mathscr{Y}}|}\frac{t^{|\mathcal{C}_{\mathscr{Y}}|-1}k^s}{n}.$$

On the other hand, if $|\mathcal{C}^A_{\mathscr{Y}}| > 0$, then

$$\prod_{U \in \mathcal{C}_{\mathscr{Y}}} (\#g_U) \leq 2^{|\mathcal{C}^B_{\mathscr{Y}}|} t^{|\mathcal{C}^A_{\mathscr{Y}}|/2a + |\mathcal{C}^B_{\mathscr{Y}}|}(k/n)^{|\mathcal{C}^B_{\mathscr{Y}}|}.$$

Since $|\mathcal{C}_{\mathscr{Y}}| - (\frac{2a-1}{2a})|\mathcal{C}^A_{\mathscr{Y}}| = |\mathcal{C}^A_{\mathscr{Y}}|/2a + |\mathcal{C}^B_{\mathscr{Y}}|$, observing that $|\mathcal{C}^B_{\mathscr{Y}}| \leq 2^s$ it suffices to show that

$$2^{|\mathcal{C}^B_{\mathscr{Y}}|}k^{2^s}\frac{t^{|\mathcal{C}_{\mathscr{Y}}|}}{n^{\frac{2a-1}{2}|\mathcal{C}^A_{y}| + |\mathcal{C}^B_{\mathscr{Y}}|}} \leq c \cdot \frac{t^{|\mathcal{C}_{\mathscr{Y}}|-1}k^{2^s}}{\sqrt{n}},$$

which holds when $\frac{2a-1}{2}|\mathcal{C}^A_{\mathscr{Y}}| + |\mathcal{C}^B_{\mathscr{Y}}| \geq a + 1/2$. This inequality follows from the fact that $|\mathcal{C}_{\mathscr{Y}}| \geq 2$ (since $S \notin \mathcal{C}_{\mathscr{Y}}$ and $\cup_{U \in \mathcal{C}_{\mathscr{Y}}} U = S$), $|\mathcal{C}^A_{\mathscr{Y}}| \geq 1$, and $a \geq \frac{3}{2}$. ■

The following lemma shows that for f drawn from $\mathcal{M}_n^{t,k}$, with high probability each term is "uniquely satisfied" by a noticeable fraction of assignments as required by condition **(C1)**. (Note that since $k = O(\log n)$ and $t > n^{3/2}$, we have $\delta_{\text{usat}} = n^{-\Omega(\log\log n)}$ in the following.)

Lemma 43. *Let $\delta_{\text{usat}} := \exp(\frac{-tk}{3n}) + t^2(\frac{k}{n})^{\log\log t}$. For n sufficiently large, with probability at least $1 - \delta_{\text{usat}}$ over the random draw of $f = T_1 \vee \cdots \vee T_t$ from $\mathcal{M}_n^{t,k}$, f is such that for all $i = 1, \dots, t$ we have $\Pr_x[T_i$ is satisfied by x but no other T_j is satisfied by x $] \geq \frac{\Theta(1)}{2^k}$.*

Proof. For a monotone t-term DNF formula $f = T_1 \vee \cdots \vee T_t$, let f^i denote the projected function obtained from f by removing the term T_i from f and restricting all of the variables which were present in term T_i to 1. For $\ell \neq i$ we write T_ℓ^i to denote the term obtained by setting all variables in T_i to 1 in T_ℓ, *i.e.,* T_ℓ^i is the term in f^i corresponding to T_ℓ. Now the probability that T_i is satisfied and no other T_j is satisfied is given by $\Pr[T_i] \cdot \Pr[\overline{T_\ell^i}$ for all $\ell \neq i \mid T_i] = \Pr[T_i] \cdot \Pr[\overline{f^i}]$. Since $\Pr[T_i] = \frac{1}{2^k}$, it suffices to bound $\Pr[\overline{f^i}]$ from below. As in [JS05b], we show that the following four facts all hold with probability $1-\delta_{usat}$:

1. $\Pr[\overline{f^i}] \geq \prod_{\ell:\ell\neq i} \Pr[\overline{T_\ell^i}]$.
2. $\prod_{\ell:T_\ell^i\equiv T_\ell} \Pr[\overline{T_\ell^i}] > 1/16$.
3. $|\{T_\ell^i : \ell \neq i \wedge T_\ell^i \not\equiv T_\ell\}| \leq \frac{2tk^2}{n}$.
4. No term in f^i has fewer than $k - \log\log t$ variables.

Together, these conditions imply that

$$\Pr[\overline{f^i}] \geq \prod_{\ell:T_\ell\equiv T_\ell^i} \Pr[\overline{T_\ell^i}] \prod_{\ell:T_\ell\not\equiv T_\ell^i,\ell\neq i} \Pr[\overline{T_\ell^i}] \geq \frac{1}{16}\left(1 - \frac{\log t}{2^k}\right)^{2tk^2/n} \geq \frac{1}{32}.$$

We now prove (1)–(4). To prove (1) note that

$$\Pr[\overline{f}] = \Pr[\overline{T_1} \wedge \overline{T_2} \wedge \cdots \wedge \overline{T_t}] = \Pr[\overline{T_1}|\overline{T_2} \wedge \cdots \wedge \overline{T_t}]\Pr[\overline{T_2}|\overline{T_3} \wedge \cdots \wedge \overline{T_t}] \cdots \Pr[\overline{T_{t-1}}|\overline{T_t}]\Pr[\overline{T_t}]$$

which is at least $\prod_{i=1}^t \Pr[\overline{T_i}]$ since f is monotone. (Conditioning on terms being unsatisfied can only increase the number of variables set to 0 and thus can only increase the chances a particular term is unsatisfied).

For any i and ℓ such that $T_\ell^i \equiv T_\ell$, we have $\Pr[\overline{T_\ell^i}] = \Pr[\overline{T_\ell}] = 1 - \Pr[T_\ell] = 1 - \frac{1}{2^k}$. Certainly there are at most t such T_ℓ^i, so (2) follows from the fact that $k = \lfloor \log t \rfloor$ so $(1 - \frac{1}{2^k})^t > 1/16$.

For (3), first we prove that with probability at least $1 - \exp(\frac{-tk}{3n})$, any variable appears in at most $\frac{2tk}{n}$ many terms. Each variable v_j appears in each fixed term T_ℓ with probability k/n. Since the terms are chosen independently, the number of occurrences of v_j is binomially distributed according to $B(t,p)$ with $p = k/n$. Now recall that the Chernoff bound gives that $\Pr[X \geq (1+\zeta)\,\mathbb{E}[X]] \leq e^{-\zeta^2 tp/3}$ where X is drawn from $B(t,p)$. Taking $\zeta = 1$, we get that $\Pr[X > \frac{2tk}{n}] < \exp(\frac{-tk}{3n})$. If $T_\ell \not\equiv T_\ell^i$ then T_ℓ must contain some variable from T_i. Assuming every variable appears in at most $2tk/n$ terms, and term T_i has at most k variables, there can be at most $k \cdot 2tk/n$ such terms.

Finally, Lemma 3.5 of Jackson and Servedio's paper [JS05b] gives that (4) holds with probability at least $1 - t^2(\frac{k^2}{n})^{\log\log t}$. Thus we have that conditions (1)–(4) all hold with probability at least $1 - \delta_{usat}$. ■

We now upper bound the probability that any j distinct terms of a random DNF formula $f \in \mathcal{M}_n^{t,k}$ will be satisfied simultaneously (condition **(C2)**). (In the following lemma, note that for $j = \Theta(1)$, since $t = n^{\Theta(1)}$ and $k = \Theta(\log n)$ we have that the quantity δ_{simult} is $n^{-\Theta(\log\log n)}$.)

Lemma 44. *Let $1 \leq j \leq 2^s$, and let $\delta_{\text{simult}} := \frac{t^j e^{jk-\log k}(jk-\log k)^{\log k}}{n^{\log k}}$. With probability at least $1-\delta_{\text{simult}}$ over the random draw of $f = T_1 \vee \cdots \vee T_t$ from $\mathcal{M}_n^{t,k}$, for all $1 \leq \iota_1 < \cdots < \iota_j \leq t$ we have $\Pr[T_{\iota_1} \wedge \ldots \wedge T_{\iota_j}] \leq \beta_j$, where $\beta_j := \frac{k}{2^{jk}}$.*

Proof. Fix any sequence $\iota_1 < \cdots < \iota_j$ of j terms. Let $v \leq jk$ be the number of distinct variables that occur in these terms. First, we will bound the probability that $v > w := jk - \log k$. Consider any particular fixed set of w variables. The probability that none of the j terms includes any variable outside of the w variables is precisely $(\binom{w}{k}/\binom{n}{k})^j$. Thus, the probability that $v \leq w$ is by the union bound:

$$\Pr[v \leq w] \leq \binom{n}{w}\left(\frac{\binom{w}{k}}{\binom{n}{k}}\right)^j \leq \left(\frac{en}{w}\right)^w \left(\frac{w}{n}\right)^{jk} \leq \frac{e^{jk-\log k}(jk-\log k)^{\log k}}{n^{\log k}}.$$

Taking a union bound over all (at most t^j) sequences $1 \leq \iota_1 < \cdots < \iota_j \leq t$, we have that with probability $1 - \delta_{\text{simult}}$, every sequence of j terms contains at least w distinct variables, and thus for every sequence we have $\Pr[T_{\iota_1} \wedge \cdots \wedge T_{\iota_j}] \leq 2^{-w} = k/2^{jk}$. ■

Finally, the following lemma shows that for all sufficiently large n, with high probability over the choice of f, every set S of s variables appears in at most γ terms, where γ is independent of n (see condition **(C4)**).

Lemma 45. *Fix any constant $c > 0$. Let $s = \lfloor a \rfloor + 2$ and let $\gamma = a + c + 1$. Let $\delta_\gamma = n^{-c}$. Then for n sufficiently large, with probability at least $1 - \delta_\gamma$ over the random draw of f from $\mathcal{M}_n^{t,k}$, we have that every s-tuple of variables appears in at most γ terms of f.*

Proof. For any fixed $r \in \{1, \ldots, t\}$ and any fixed S such that $|S| = s$, we have Pr[all variables in S occur in T_r] $= \frac{k(k-1)\cdots(k-s+1)}{n(n-1)\cdots(n-s+1)} \leq \left(\frac{k}{n}\right)^s$. Since terms are chosen independently, the probability that the variables in S co-occur in a fixed collection of $\gamma + 1$ terms is at most $\left(\frac{k}{n}\right)^{s(\gamma+1)}$. By the union bound, the probability that these variables co-occur in any collection of $\gamma + 1$ terms is at most $\binom{t}{\gamma+1} \cdot \left(\frac{k}{n}\right)^{s(\gamma+1)} \leq \left(\frac{tk^s}{n^s}\right)^{\gamma+1}$. Using the union bound again, we have that the probability that any s-tuple of variables co-occurs in more than γ terms is at most $\binom{n}{s} \cdot \left(\frac{tk^s}{n^s}\right)^{\gamma+1}$. Recalling that $t = n^a$, that $s = \lfloor a \rfloor + 2$, and that $k = \lfloor \log t \rfloor = O(\log n)$, we have that this probability is at most $\text{polylog}(n) \cdot n^{a(\gamma+1)-(a+1)\gamma} = \text{polylog}(n) \cdot n^{a-\gamma}$. By our choice of γ this is at most δ_γ, and the proof is done. ■

5.5 Proof of the Main Theorem

Theorem 46. **[Formally]** *Let $t(n)$ be any function such that $t(n) \leq \text{poly}(n)$, let $a(n) = O(1)$ be such that $t(n) = n^{a(n)}$, and let $c > 0$ be any fixed constant. Then for any $n^{-c} < \delta < 1$ and $0 < \epsilon < 1$, $\mathcal{M}_n^{t(n),\lfloor \log t(n) \rfloor}$ is PAC learnable under U_n in poly($n^{2a(n)+c+3}$,$(a(n) + c + 1)^{\log t(n)}$,$t(n)$,$1/\epsilon$, $\log 1/\delta$) time.*

Proof. The result is proved for $t(n) \leq n^{3/2}$ already by Jackson and Servedio [JS05b], so we henceforth assume that $t(n) \geq n^{3/2}$. We use Theorem 40 and show that for $s = \lfloor a(n) \rfloor + 2$, random monotone $t(n)$-term DNF formulas, with probability at least $1 - \delta$, satisfy conditions **(C1)**–**(C5)** with values $\alpha, \beta_j, \Phi(\cdot), \Delta, \gamma$, and κ such that $\Delta > 0$ and the quantities

$n^{s+\gamma}, 1/\Delta$, and γ^{κ} are polynomial in n. This will show that the extended version of Algorithm $\mathcal{A}$ defined in Section 5.3 PAC learns random monotone $t(n)$-term DNF formulas in time $\mathrm{poly}(n, 1/\epsilon)$. Let $t = t(n)$ and $k = \lfloor \log t \rfloor$, and let f be drawn randomly from $\mathcal{M}_n^{t,k}$. By Lemmas 42–45, with probability at least $1 - \delta_{\text{usat}} - \delta_{\gamma} - 2^{2^s}\delta_{\text{terms}} - \delta_{\text{simult}}$, f will satisfy **(C1)**–**(C5)** with the following values:

$$\textbf{(C1)}\ \alpha > \tfrac{\Theta(1)}{2^k};\ \ \textbf{(C2)}\ \beta_j \le \tfrac{k}{2^{jk}} \text{ for } 1 \le j \le 2^s;$$
$$\textbf{(C3)}\ \Phi(\mathcal{C}_{\mathscr{Y}}) \le O(1)\tfrac{t^{|\mathcal{C}_{\mathscr{Y}}|-1}k^{2^s}}{\sqrt{n}} \text{ for all } \mathcal{C}_{\mathscr{Y}} \in \mathscr{Y};\ \textbf{(C4)}\ \gamma \le a(n) + c + 1;$$
$$\textbf{(C5)}\ \kappa = k = \lfloor \log t \rfloor,$$

which gives us that $n^{s+\gamma} = n^{2a+c+3}$ and $\gamma^{\kappa} = (a+c+1)^{\lfloor \log t \rfloor}$. Finally, we show that $\Delta = \Omega(1/t)$ so $1/\Delta$ is polynomial in n:

$$\begin{aligned}
\Delta &= \alpha/2^s - 3\cdot\Upsilon = \frac{\Theta(1)}{t2^s} - 3\sum_{\mathcal{C}_{\mathscr{Y}} \in \mathscr{Y}} 2^s \beta_{|\mathcal{C}_{\mathscr{Y}}|}\Phi(\mathcal{C}_{\mathscr{Y}}) \\
&\ge \frac{\Theta(1)}{t2^s} - \Theta(1)\sum_{\mathcal{C}_{\mathscr{Y}} \in \mathscr{Y}} 2^s \frac{k}{t^{|\mathcal{C}_{\mathscr{Y}}|}} \cdot \frac{t^{|\mathcal{C}_{\mathscr{Y}}|-1}k^{2^s}}{\sqrt{n}} \\
&= \frac{\Theta(1)}{t2^s} - \frac{\Theta(1)k^{2^s+1}}{t\sqrt{n}} = \Omega(1/t).
\end{aligned}$$

■

5.6 Discussion

Robustness of parameter settings. Throughout Sections 5.4 and 5.5 we have assumed for simplicity that the term length k in our random t-term monotone DNF formula is exactly $\lfloor \log t \rfloor$. In fact, the results extend to a broader range of k's; one can straightforwardly verify that by very minor modifications of the given proofs, Theorem 36 holds for $\mathcal{M}_n^{t,k}$ for any $(\log t) - O(1) \le k \le O(\log t)$.

Relation to previous results. Our results are powerful enough to subsume some known "worst-case" results on learning restricted classes of monotone DNF formulas. Hancock and Mansour [HM91] have shown that read-k monotone DNF formulas are learnable under the uniform distribution in $\mathrm{poly}(n)$ time for constant k. Their result extends an earlier result of Kearns *et al.* [KLV94] showing that read-once DNF formulas (which can be assumed to be monotone without loss of generality) are polynomial-time learnable under the uniform distribution.

It is not hard to see that (a very restricted special case of) our algorithm can be used to learn read-k monotone DNF formulas in polynomial time. Note first that we may assume the unknown target read-k DNF formula f has $\frac{\epsilon}{2} \leq \Pr[f(x) = 1] \leq 1 - \frac{\epsilon}{2}$, since otherwise it is trivial to learn to accuracy ϵ.

We show that we can apply Theorem 40 to learn f. Any read-k DNF formula has at most kn total occurrences of variables, so we certainly have that f is a $t(n)$-term DNF formula with $t(n) = O(n)$. We will take $s = 1$. Since f is a read-k DNF formula, we may take $\gamma = 2$ in condition **(C4)**. By the usual reasoning, we may suppose without loss of generality that each term of f contains at most $O(\log \frac{n}{\epsilon})$ many variables (this is because the probability that any longer term is ever satisfied by any example in a poly(n/ϵ)-size set of random examples is negligibly small). Thus we may take $\kappa = O(\log \frac{n}{\epsilon})$ in condition **(C5)**.

Turning to Lemma 39, since $s = 1$ we have that the collection $\mathscr{Y}$ is in fact empty – for $S = \{x_i\}$, the only $\mathcal{C} \subseteq \mathcal{P}(S)$ that cover S are $\mathcal{C} = \{\emptyset, \{x_i\}\}$ and $\mathcal{C} = \{\{x_i\}\}$, both of which clearly contain S. We thus have $\Upsilon = 0$, so $\Delta = \frac{\alpha}{2}$ and it remains only to prove that α is "not too small," *i.e.,* that each term in f is uniquely satisfied with probability at least $\Omega(1/poly(n/\epsilon))$. An easy argument in [HM91] gives precisely the desired result; they show that for any monotone read-k DNF formula f that has $\Pr[f(x) = 0] = p$, every term T that contains $\mathcal{C}$ variables satisfies $\Pr[T$ is true and all other terms are false$] \geq p2^{-k|\mathcal{C}|}$. Since we have $p \geq \frac{\epsilon}{2}$ and $\mathcal{C} \leq \kappa = O(\log \frac{n}{\epsilon})$, we obtain $\alpha \geq \Omega(1/\operatorname{poly}(n/\epsilon))$ as required. So we may apply Theorem 40 to conclude that our algorithm learns f in time $\operatorname{poly}(n, 1/\epsilon, \log(1/\delta))$.

Chapter 6

The Structure of Monotone Decision Trees

In this chapter we consider the concept class of polynomial-size monotone decision trees – a subset of polynomial-size monotone DNF formulas. We show the relationship between the average depth of a monotone decision tree, the influences of the variables and its variance.

6.1 Introduction

Decision trees are one of the most important concept classes in the field of computational learning theory. They are perhaps the smallest concept class for which no efficient algorithms exist, yet they are also widely used in experimental and applied machine learning. The heuristics used in machine learning (*e.g.,* the C4.5 and CART software packages [BFSO84, Qui93]) "grow" a tree from the root to its leaves by repeatedly replacing an existing leaf with a node labeled with a variable that minimizes the empirical error with respect to the given data sample.

Unfortunately, these heuristics have been difficult to analyze in the PAC model of learning. The most theoretically rigorous work on these heuristics [KM96] showed that they were in some sense *boosting* algorithms. If the error-minimizing nodes were assumed to be good weak-learners, the decision tree growing algorithms would boost them into strong learners.

The best algorithm for PAC-learning decision trees is still the one by Ehrenfeucht and

Haussler from 1989 [EH89], which learns size-s decision trees in time $O(n^{\log s})$. Even when the inputs are assumed to be distributed uniformly, finding an efficient algorithm for polynomial-size decision trees remains an open problem.

As is usually the case when researchers have difficulty solving a problem, many alternative models to uniform-distribution PAC-learning have been considered. Efficient algorithms for learning polynomial-size decision trees exist when: the algorithms are allowed to make *membership queries* [KM93]; when the examples are assumed to come from a random walk on the Boolean hypercube [BMOS03]; when the learner is allowed *extended* statistical queries [BF02]; or as in Chapter 5 when there is a natural distribution over the concept class [JS05a].

Much like DNF formulas, the lack of success should be somewhat expected since poly(n)-size decision trees can compute parity functions of size $\log(n)$ over n variables, and thus have super-polynomial statistical-query dimension. This, of course, is not an issue when learning polynomial-size decision trees that compute *monotone* functions (we call these "monotone decision trees"). In fact, there is an algorithm for learning poly(n)-size decision trees that compute monotone functions to accuracy ϵ in $O(n^{1/\epsilon^2})$ time [OS07] (which is polynomial-time for constant ϵ). Their algorithm outputs a real-valued polynomial threshold function as its hypothesis.

6.1.1 Our Results

Ideally, one would hope for an algorithm that learns polynomial-size monotone decision trees to arbitrary accuracy, under any distribution, and outputs a polynomial-size monotone decision tree as a hypothesis. We look to the machine learning heuristics for inspiration and propose the following algorithm.

Let T be a decision tree computing a function $f : \{0,1\}^n \to \{+1,-1\}$. We write $\delta_i(T)$ to be the probability that coordinate x_i is queried by the decision tree on a uniform random input, and we write:

$$\Delta(T) \overset{\text{def}}{=} \sum_{i=1}^{n} \delta_i(T) = \mathop{\mathbb{E}}_{x}\left[\#\text{ coords } T \text{ queries on } x\right].$$

$\Delta(T)$ can also be thought of as the average depth of the decision tree, or as a refinement of the notion of the size of the decision tree as, $\Delta(T) \leq \log(\text{size}(T))$ [OS07]. As an example, consider the decision tree in Figure 6.1. It has $\delta_1(T) = 3/4$, $\delta_2(T) = 1$, $\delta_3(T) = 1/2$, and $\Delta(T) = 9/4$.

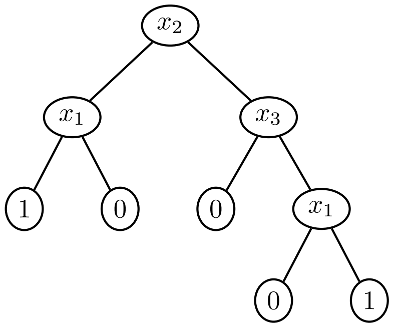

Figure 6.1: A decision tree with $\Delta(T) = 9/4$.

When learning a decision tree with average-depth Δ, our proposed algorithm will grow a decision tree as the hypothesis by repeatedly replacing an existing leaf with a node labeled with the variable that has the greatest *influence* (defined in Section 6.2). We will do this for $\text{poly}(\Delta, \epsilon^{-1})$ rounds, and hopefully have an ϵ-accurate hypothesis at the end. Unlike non-monotone functions, the variable influences of monotone functions can be estimated given uniform random examples. There are also two recent results that give us hope that our proposed algorithm can be analyzed successfully.

The first, which was the main technical result that lead to the $O(n^{1/\epsilon^2})$-time algorithm for learning monotone decision trees by O'Donnell and Servedio [OS07] is:

Theorem 47 ([OS07] Theorem 2)**.** *Let f be a monotone Boolean function. Then, the average sensitivity of f is at most $\sqrt{\Delta(f)}$, where $\Delta(f)$ is the average-depth of the decision tree for f that minimizes Δ.*

We give a slight refinement[1] of their theorem for decision trees:

[1]In the paper by O'Donnell and Servedio, the definition of $\Delta(f)$ was extended to the average number of variables fixed in the "subcube partition" representation of the function, which is a generalization of the decision-tree representation. Thus, their Theorem 2 is incomparable with our theorem which only holds for decision trees.

Theorem 48. *Let f be a monotone Boolean function. Then, the average sensitivity of f is at most $\sqrt{\mathrm{Var}[f] \cdot \Delta(f)}$.*

The second by O'Donnell *et al.* [OSSS05], which was used to prove a lower bound on the randomized query complexity of nontrivial monotone graph properties is:

Theorem 49 ([OSSS05] Theorem 1.1)**.** *Let f be a Boolean function, and let T be a decision tree computing f. Then,*

$$\mathrm{Var}[f] \leq \sum_{i=1}^{n} \delta_i(T)\mathrm{Inf}_i(f).$$

Their proof of this result relied on some delicate probabilistic reasoning about the independence of certain hybrid inputs to the decision tree. We give a simple inductive proof of this same theorem, which closely follows the structure of our proposed algorithm.

We give the needed background on influences in Section 6.2. Then we reprove the O'Donnell *et al.* [OSSS05] result in Section 6.3. Finally, we give our variant of the bound on average sensitivity in Section 6.4.

6.2 Influence

The notion of variable influences was introduced by Ben-Or and Linial [BOL87] in the context of computational game theory, and it has proven to be very useful in Boolean function analysis [KKL88, BKS99, KOS04, FKN02, OS07].

Definition 50. *For $f : \{0,1\}^n \to \mathbb{R}$, $i \in [n]$, the* influence *of i on f is:*

$$\mathrm{Inf}_i(f) = \sum_{S: i \in S} \hat{f}(S)^2.$$

Proposition 51. *For any Boolean function $f : \{0,1\}^n \to \{+1,-1\}$,*

$$\mathrm{Inf}_i(f) = \Pr_{x \in \{0,1\}^n}[f(x) \neq f(x^{(i)})],$$

where $x^{(i)}$ denotes x with its i-th bit flipped (negated).

Proof.

$$\begin{aligned}
\Pr_{x\in\{0,1\}^n}[f(x)\neq f(x^{(i)})] &= \mathop{\mathbb{E}}_x\left[\frac{1}{2}-\frac{1}{2}f(x)f(x^{(i)})\right]\\
&= \frac{1}{2}-\frac{1}{2}\mathop{\mathbb{E}}_x\left[f(x)f(x^{(i)})\right]\\
&= \frac{1}{2}-\frac{1}{2}\sum_{S,T\subseteq[n]}\hat{f}(S)\hat{f}(T)\mathop{\mathbb{E}}_x\left[\chi_S(x)\chi_T(x^{(i)})\right]\\
&= \frac{1}{2}-\frac{1}{2}\sum_{S,T}\hat{f}(S)\hat{f}(T)\mathop{\mathbb{E}}_x\left[\chi_{S\oplus T}(x)(-1)^{\mathbf{1}_{i\in T}}\right]\\
&= \frac{1}{2}-\frac{1}{2}\sum_{S}\hat{f}(S)^2(-1)^{\mathbf{1}_{i\in S}}\\
&= \frac{1}{2}\sum_S\hat{f}(S)^2-\frac{1}{2}\sum_S\hat{f}(S)^2(-1)^{\mathbf{1}_{i\in S}}\\
&= \sum_{S:i\in S}\hat{f}(S)^2,
\end{aligned}$$

where the penultimate equality is due to Parseval's Theorem. ■

The *total influence* is defined to be the sum of all the influences:

$$\mathbb{I}(f) := \sum_{i=1}^{n}\mathrm{Inf}_i(f) = \sum_{i=1}^{n}\sum_{S:i\in S}\hat{f}(S)^2 = \sum_{S\subseteq[n]}|S|\hat{f}(S)^2.$$

The *sensitivity* of a Boolean function f on an input x is the number of neighbors of x with different values. The *average sensitivity* of f is the average of the sensitivities of all inputs:

$$\mathop{\mathbb{E}}_x\left[\left|\{i\in[n]:f(x)\neq f(x^{(i)})\}\right|\right].$$

It should be easy to see that the average sensitivity is the same as the total influence.

The influences of monotone functions have the nice property that they are just the degree-1 Fourier coefficients.

Proposition 52. *If $f:\{0,1\}^n\to\{+1,-1\}$ is monotone, then $\hat{f}(\{i\})=\mathrm{Inf}_i(f)$.*

Proof. By Proposition 51,

$$\begin{aligned}
\mathrm{Inf}_i(f) &= \Pr_x[f(x)\neq f(x^{(i)})]\\
&= \Pr_x[f(x)=(-1)^{x_i}\wedge f(x^{(i)})=-(-1)^{x_i}]\\
&\quad+\Pr_x[f(x)=-(-1)^{x_i}\wedge f(x^{(i)})=(-1)^{x_i}].
\end{aligned}$$

However, the second term is always 0 when f is monotone. Similarly,

$$\hat{f}(\{i\}) = \mathop{\mathbb{E}}_x[f(x)(-1)^{x_i}] = \Pr_x[f(x) = (-1)^{x_i}] - \Pr_x[f(x) = -(-1)^{x_i}].$$

We can expand this to:

$$\begin{aligned} & \Pr_x[f(x) = (-1)^{x_i} \wedge f(x^{(i)}) = -(-1)^{x_i}] + \Pr_x[f(x) = (-1)^{x_i} \wedge f(x^{(i)}) = (-1)^{x_i}] \\ - \; & \Pr_x[f(x) = -(-1)^{x_i} \wedge f(x^{(i)}) = (-1)^{x_i}] - \Pr_x[f(x) = -(-1)^{x_i} \wedge f(x^{(i)}) = -(-1)^{x_i}]. \end{aligned}$$

The second and fourth terms cancel each other out, and the third is always 0 when f is monotone. ■

Thus, one can estimate the influences of monotone functions from uniform examples (by a simple application of the Hoeffding bound).

The variance of $f : \{0,1\}^n \to \mathbb{R}$ can also be expressed as the sum of squares of Fourier coefficients.

$$\mathrm{Var}[f] = \mathbb{E}[f^2] - \mathbb{E}[f]^2 = \sum_S \hat{f}(S)^2 - \hat{f}(\emptyset)^2 = \sum_{S:|S|\geq 1} \hat{f}(S)^2.$$

We can conclude that $\mathrm{Var}[f] \leq \mathbb{I}(f)$ for all f. (This inequality is also known as the Poincaré Inequality for the discrete cube.)

6.3 A Poincaré-type Inequality for Decision Trees

In this section we reprove the O'Donnell *et al.* [OSSS05] inequality:

$$\mathrm{Var}[f] \leq \sum_{i=1}^n \delta_i(T)\mathrm{Inf}_i(f).$$

This inequality can be viewed as a refinement of the Poincaré Inequality that takes into account the complexity of the function's representation.

We will prove this inequality by induction. To do so, we will consider the function's behavior under the two cases when the root variable takes the value 0 and the value 1, but first we will review a fact from probability theory.

For a function $f : \{0,1\}^n \to \mathbb{R}$, let $c_i = \mathbb{E}[f|(x_1, \ldots, x_i)] - \mathbb{E}[f|(x_1, \ldots, x_{i-1})]$, that is the average of f over the points y that coincide with x in the first i coordinates. The sequence

$\{c_i\}$ is a martingale difference sequence. Let g be another real-valued function, and let $\{d_i\}$ be its martingale difference sequence. Then $\mathrm{Cov}[f,g] = \sum_{i=1}^{n} \mathbb{E}\, c_i d_i$. We'll prove this fact for the sake of completeness.

Fact 53. *Let $f, g : \{0,1\}^n \to \mathbb{R}$ be real-valued functions, with martingale difference sequences:*

$$c_i = \mathbb{E}[f|(x_1,\ldots,x_i)] - \mathbb{E}[f|(x_1,\ldots,x_{i-1})] \quad \textit{and} \quad d_i = \mathbb{E}[g|(x_1,\ldots,x_i)] - \mathbb{E}[g|(x_1,\ldots,x_{i-1})].$$

Then $\mathrm{Cov}[f,g] = \sum_{i=1}^{n} \mathbb{E}\, c_i d_i$.

Proof. We'll use some basic properties of conditional expectation and martingales. For $j < k$, we have

$$\begin{aligned} \mathbb{E}\, c_j d_k &= \mathbb{E}\,\mathbb{E}\left[c_j d_k|(x_1,\ldots,x_{k-1})\right] \\ &= \mathbb{E}\, c_j\, \mathbb{E}\left[d_k|(x_1,\ldots,x_{k-1})\right] \\ &= \mathbb{E}\, c_j \cdot 0 = 0. \end{aligned}$$

Given this, the rest of the proof is straight-forward:

$$\begin{aligned} \mathrm{Cov}[f,g] &= \mathbb{E}[f - \mathbb{E} f]\,\mathbb{E}[g - \mathbb{E} g] \\ &= \mathbb{E}\left[\sum c_i\right] \mathbb{E}\left[\sum d_i\right] \\ &= \mathbb{E} \sum_j \sum_k c_j d_k \\ &= \sum_{i=1}^{n} \mathbb{E}\, c_i d_i. \end{aligned}$$

■

Let f_0 and f_1 denote the function f restricted to inputs where $x_n = 0$ and $x_n = 1$ respectively. Then we have that $c_n = f - (f_0 + f_1)/2$. We can rewrite the covariance as

$$\mathrm{Cov}[f,g] = \sum_{i=1}^{n-1} \mathbb{E}\, c_i d_i + \mathbb{E}\left(f - \frac{f_0 + f_1}{2}\right)\left(g - \frac{g_0 + g_1}{2}\right).$$

Now let $f, g : \{0,1\}^n \to \{+1,-1\}$ be Boolean functions. For $i \neq n$, we get the following expression:

$$\mathrm{Inf}_i(f) = \frac{1}{2}\mathrm{Inf}_i(f_0) + \frac{1}{2}\mathrm{Inf}_i(f_1),$$

and we get: $\mathrm{Inf}_n(f) = \Pr_{x\in\{0,1\}^{n-1}}[f_0(x) \neq f_1(x)]$, for the influence of x_n on f.

Going back to our expression for the covariance, notice that the quantity $(f-(f_0+f_1)/2)(g-(g_0+g_1)/2)$ is 0 unless both $f_0 \neq f_1$ and $g_0 \neq g_1$. We can thus upper-bound the covariance by:

$$\mathrm{Cov}[f,g] \leq \sum_{i=1}^{n-1} \mathbb{E}\, c_i d_i + \mathrm{Inf}_n(g),$$

where our choice of g is arbitrary. Note that this upper-bound is an equality when we consider the special case of $f = g$,

$$\mathrm{Cov}[f,f] = \mathrm{Var}[f] = \sum_{i=1}^{n-1} \mathbb{E}\, c_i^2 + \mathrm{Inf}_n(f).$$

Let T be a decision tree computing f. Without loss of generality, let x_n be the root of the tree, and let T_0 and T_1 be the left and right subtrees. Then we have:

$$\delta_i(T) = \frac{1}{2}\delta_i(T_0) + \frac{1}{2}\delta_i(T_1).$$

We are now ready to prove the main statement.

Theorem 54. *Let $f, g : \{0,1\}^n \to \{+1,-1\}$ be Boolean functions, and let T be a decision tree computing f. Then,*

$$\mathrm{Cov}[f,g] \leq \sum_{i=1}^{n} \delta_i(T)\mathrm{Inf}_i(g).$$

Proof. We'll prove the statement by induction on the number of variables. When there is only one variable, the function is either constant, in which case $\mathrm{Var}[f] = \mathrm{Inf}_1(f) = \delta_1(T) = 0$, or x_1, in which case $\mathrm{Var}[f] = \mathrm{Inf}_1(f) = \delta_1(T) = 1$. One can easily verify that the theorem holds for all the possible combinations.

Now we'll consider f and g on n variables. As in the discussion above, let us assume without loss of generality that the root of T queries x_n. Let f_0 and f_1 denote the function f restricted to inputs where $x_n = 0$ and $x_n = 1$ respectively. Let $c_{i,0} = \mathbb{E}[f_0|(x_1,\ldots,x_i)] - \mathbb{E}[f_0|(x_1,\ldots,x_{i-1})]$. Define $g_0, g_1, c_{i,1}, d_{i,0}, d_{i,1}$ similarly. Then we have $c_i = (c_{i,0}+c_{i,1})/2$, $d_i = (d_{i,0}+d_{i,1})/2$, and we can write the covariance as:

$$\mathrm{Cov}[f,g] = \sum_{i=1}^{n} \mathbb{E}\, c_i d_i = \frac{1}{4}\sum_{a,b\in\{0,1\},i\in[n-1]} \mathbb{E}\, c_{i,a} d_{i,b} + \mathbb{E}\, c_n d_n. = \frac{1}{4}\sum_{a,b\in\{0,1\}} \mathrm{Cov}[f_a, g_b] + \mathbb{E}\, c_n d_n.$$

Since f_a and g_b are functions on $n-1$ variables we can use the induction hypothesis, and we have:

$$\mathrm{Cov}[f,g] \le \frac{1}{4} \sum_{a,b\in\{0,1\}, i\in[n-1]} \delta_i(T_a)\mathrm{Inf}_i(g_b) + \mathbb{E}\, c_n d_n = \sum_{i\in[n-1]} \delta_i(T)\mathrm{Inf}_i(g) + \mathbb{E}\, c_n d_n.$$

Finally, we have that $\mathbb{E}\, c_n d_n \le \mathrm{Inf}_n(g)$, and $\delta_n(T) = 1$ since x_n is the root of the tree. Thus, the induction holds. ■

As a corollary we get:

Corollary 55. *Let $f : \{0,1\}^n \to \{+1,-1\}$ be a Boolean function, and let T be a decision tree computing f. Then,*

$$\mathrm{Var}[f] \le \sum_{i=1}^{n} \delta_i(T)\mathrm{Inf}_i(f).$$

The corollary implies a lower bound on the maximum influence variable of f:

$$\mathrm{Inf}_{max}(f) \ge \mathrm{Var}[f]/\Delta(T).$$

6.4 The Average Sensitivity of Monotone Decision Trees

In this section we prove our variant of the O'Donnell and Servedio inequality [OSSS05]:

$$\mathbb{I}(f) \le \sqrt{\Delta(f)}.$$

This inequality can be viewed as an edge-isoperimetric inequality for the discrete cube, and yields the classic upper bound on the average sensitivity of monotone functions:

$$\mathbb{I}(f) \le \mathbb{I}(\mathrm{Maj}) = \Theta(\sqrt{n}),$$

which follows from the Kruskal-Katona theorem.

We will be using Jensen's inequality in the proof of our theorem.

Fact 56 (Jensen's inequality)**.** *For a real convex function ϕ, numbers $b_1, \dots, b_n$ in its domain, and positive weights a_i,*

$$\phi\left(\frac{\sum a_i b_i}{\sum a_i}\right) \le \frac{\sum a_i \phi(b_i)}{\sum a_i}.$$

Theorem 57. *Let f be a monotone Boolean function, and let T be a decision tree computing f. Then,*

$$\sum_{i=1}^{n} \mathrm{Inf}_i(f) \leq \sqrt{\mathrm{Var}[f] \cdot \Delta(T)}.$$

Proof. Recall that the value $\delta_i(T)$ is the probability over a uniform input that the decision tree T queries the variable x_i. Let $i_1, \ldots, i_\ell$ be all the instances of x_i in the tree. Then

$$\delta_i(T) = \sum_{j=1}^{\ell} a_j^i,$$

where $a_j^i := 2^{-\mathrm{depth}(i_j)}$.

Now consider $\mathrm{Inf}_i(f)$. We can rewrite the influence in terms of the subfunctions of f that consist of the subtrees whose roots are x_i. Consider some instance i_j of x_i in T. Let b_j^i be the influence of x_i in the function calculated by the sub-tree rooted at i_j. Note that the only instance of x_i in the sub-tree rooted at i_j is i_j itself, and that any input to the tree f can only go to one instance of x_i in the tree. Thus, we can write:

$$\mathrm{Inf}_i(f) = \sum_{j=1}^{\ell} a_j^i b_j^i.$$

Finally, let us consider the variance. As in the proof of Theorem 54, let us consider the two subfunctions f_0 and f_1 that are the function f restricted to inputs where the root variable takes the value 0 and 1, respectively.

Since f is monotone, $\mathbb{E}[f_0] = \mathbb{E}[f] - \mathrm{Inf}_n(f)$ and $\mathbb{E}[f_1] = \mathbb{E}[f] + \mathrm{Inf}_n(f)$. We know that $\mathrm{Var}[f] = 1 - \mathbb{E}[f]^2$, and so we have:

$$\mathrm{Var}[f_0] = 1 - \mathbb{E}[f_0]^2 = 1 - (\mathbb{E}[f] - \mathrm{Inf}_n(f))^2.$$

If we average the variances of both f_0 and f_1 we get:

$$\begin{aligned}\frac{\mathrm{Var}[f_0] + \mathrm{Var}[f_1]}{2} &= \frac{1 - (\mathbb{E}[f] + \mathrm{Inf}_n(f))^2 + 1 - (\mathbb{E}[f] - \mathrm{Inf}_n(f))^2}{2} \\ &= 1 - \mathbb{E}[f]^2 - \mathrm{Inf}_n(f)^2 \\ &= \mathrm{Var}[f] - \mathrm{Inf}_n(f)^2.\end{aligned}$$

That is, each node in the decision tree T contributes its influence squared weighted by its depth to the variance. So,

$$\mathrm{Var}[f] = \sum_{i=1}^{n} \sum_{j} a_j^i (b_j^i)^2.$$

Now we will use Jensen's inequality:

$$\left(\frac{\sum_{i,j} a^i_j b^i_j}{\sum_{i,j} a^i_j}\right)^2 \leq \frac{\sum_{i,j} a^i_j (b^i_j)^2}{\sum_{i,j} a^i_j}.$$

Or in other words:

$$\left(\frac{\mathbb{I}(f)}{\Delta(f)}\right)^2 \leq \frac{\mathrm{Var}[f]}{\Delta(f)},$$

as was to be shown. ■

It's important to note that our bound crucially relies on the fact that f is monotone. The parity function on n variables is not monotone, and it has $\mathrm{Var}[f] = 1$ and $\mathbb{I}(f) = \Delta(f) = n$.

Chapter 7

Teaching DNF in the Average Case

In our last chapter, we return to monotone DNF formulas, and we show that they are teachable in the average case. In the teaching model of learning the example oracle acts as a helpful teacher and provides "useful" examples instead of random ones. We also show that non-monotone DNF formulas, juntas, and sparse GF_2 formulas are teachable in the average case.

7.1 Introduction

Many results in computational learning theory consider learners that have some form of access to an oracle that provides labeled examples. Viewed as teachers, these oracles tend to be unhelpful as they typically either provide random examples selected according to some distribution, or they put the onus on the learner to select the examples herself. In noisy learning models, oracles are even allowed to lie from time to time.

In this chapter we study a learning model in which the oracle acts as a helpful teacher [GK92, GRS93, SM90]. Given a target concept c that belongs to a concept class $\mathcal{C}$, the teacher provides the learner with a carefully chosen set of examples that are labeled according to c. This set of labeled examples is called a *teaching set* and must have the property that no other concept $c' \neq c$ in $\mathcal{C}$ is consistent with the teaching set; thus every learner that outputs a consistent hypothesis will correctly identify c as the target concept. The minimum number of examples in any teaching set for c is called the *teaching dimension of*

c with respect to $\mathcal{C}$, and the maximum value of the teaching dimension over all concepts in $\mathcal{C}$ is the *teaching dimension of $\mathcal{C}$*.

Some concept classes that are easy to learn can be very difficult to teach in the worst case in this framework. As one example, let the concept class $\mathcal{C}$ over a finite domain X contain $|X|+1$ concepts which are the $|X|$ singletons and the empty set. Any teaching set for the empty set must contain every example in X, since if $x \in X$ is missing from the set then the singleton concept $\{x\}$ is not ruled out by the set. Thus the teaching dimension for this concept class is $|X|$.

Many interesting concept classes include the empty set and all singletons, and thus have teaching dimension $|X|$. Consequently for many concept classes the (worst-case) teaching dimension is not a very interesting measure. With this motivation, researchers have considered the *average teaching dimension*, namely the average value of the teaching dimension of c as c ranges over all of $\mathcal{C}$.

Anthony *et al.* [ABST95] showed that the average teaching dimension of the class of linearly separable Boolean functions over $\{0,1\}^n$ is $O(n^2)$. Kuhlmann [Kuh99] showed that concept classes with VC dimension 1 over finite domains have constant average teaching dimension and also gave a bound on the average teaching dimension of concept classes $\mathcal{B}^d(c)$ (balls of center c and size $\leq d$). Kushilevitz *et al.* [KLRS96] constructed a concept class $\mathcal{C}$ that has an average teaching dimension of $\Omega(\sqrt{|\mathcal{C}|})$ (this lower bound was also proved in [CS98]) and also showed that every concept class has average teaching dimension at most $O(\sqrt{|\mathcal{C}|})$. More recently, Balbach [Bal05] showed that the classes of 2-term DNF formulas and 1-decision lists each have average teaching dimension linear in n.

Our Results. Our main results are the following theorems, proved in Sections 7.3 and 7.4, which show that the well-studied concept classes of monotone DNF formulas and DNF formulas are efficiently teachable in the average case:

Theorem 58. *Fix any $1 \leq s \leq 2^{\Theta(n)}$ and let $\mathcal{C}$ be the concept class of all Boolean functions over $\{0,1\}^n$ representable as a monotone DNF formulas with at most s terms. Then the average teaching dimension of $\mathcal{C}$ is $O(ns)$.*

Theorem 59. *Fix any $1 \leq s \leq 2^{\Theta(n)}$ and let $\mathcal{C}$ be the concept class of all Boolean functions over $\{0,1\}^n$ representable as a DNF with at most s terms. Then the average teaching* *dimension of $\mathcal{C}$ is $O(ns)$.*

Theorem 59 is a broad generalization of Balbach's result on the average teaching dimension of the concept class of DNF formulas with at most two terms. It is easy to see that even the class of at-most-2-term DNF formulas has exponential worst-case teaching dimension; as we show in Section 7.3, the worst-case teaching dimension of at-most-s-term monotone DNF formulas is exponential as well. Thus our results show that there is a dramatic difference between the worst-case and average teaching dimensions for these concept classes.

We also consider some other well-studied concept classes, namely juntas and sparse GF_2 polynomials. For the class of k-juntas, we show in Section 7.5 that while the worst-case teaching dimension has a logarithmic dependence on n (the number of irrelevant variables), the average teaching dimension has no dependence on n. For a certain class of sparse GF_2 polynomials (roughly, the class of GF_2 polynomials with fewer than $\log n$ terms; see Section 7.6), we show that while the worst-case teaching dimension is $n^{\Theta(\log \log n)}$, the average teaching dimension is $O(n \log n)$. Thus in each case we establish an asymptotic separation between the worst-case teaching dimension and the average teaching dimension. Our results here suggest that rich and interesting concept classes that are difficult to learn in many models may in fact be easy to teach in the average case.

7.2 Preliminaries

Recall that for a given instance $x \in X$, the value of $c(x)$ is referred to as a *label*, and for $y \in \{0, 1\}$, the pair (x, y), is referred to as a *labeled example.* If $y = 0$ ($y = 1$) then the pair is called a *negative* (*positive*) *example.* A concept class $\mathcal{C}$ is *consistent* with a set of labeled examples if $c(x) = y$ for all the examples in the set.

A set S of labeled examples is a *teaching set for c with respect to $\mathcal{C}$* if c is the only concept in $\mathcal{C}$ that is consistent with S; thus every learner that outputs a consistent hypothesis from $\mathcal{C}$ will correctly identify c as the target concept. We will also refer to a teaching set S for c as $TS(c)$. The minimum number of examples in any teaching set for c is called the *teaching dimension of c with respect to $\mathcal{C}$* (sometimes written $TD(c)$ when $\mathcal{C}$ is understood), and the maximum value of the teaching dimension over all concepts in $\mathcal{C}$ is the (worst-case) *teaching dimension of* $\mathcal{C}$. The *average teaching dimension* of $\mathcal{C}$ is the average value of the teaching dimension of c with respect to $\mathcal{C}$ for all c, *i.e.,* $\frac{1}{|\mathcal{C}|}\sum_{c\in\mathcal{C}} TD(c)$. Given a set S of variables, we write $\mathbf{0}|_{S=1}$ to denote the truth assignment that sets each variable in S to 1 and sets all other variables to 0. The truth assignment $\mathbf{1}|_{S=0}$ is defined similarly.

DNF Formulas. Recall that a *term* is a conjunction of Boolean literals. A term over n variables is represented by a string $T \in \{0, 1, *\}^n$, where the k-th character of T is denoted $T[k]$. The value of $T[k]$ is 0, 1, or $*$ depending on whether x_k occurs negated, unnegated, or not at all in the term. If $x \in \{0,1\}^n$ is an assignment that satisfies T, we sometimes say that T *covers* x. Note that the satisfying assignments of a term T form a subcube of dimension $n - |T|$ within the n-dimensional hypercube $\{0,1\}^n$. Also recall that *s-term DNF formula* ϕ is an OR of s terms $\phi = T_1 \vee \cdots \vee T_s$. A satisfying assignment to the DNF formula is sometimes referred to as a *positive point* and an unsatisfying assignment as a *negative point.*

A term T_i is said to be *compatible* with a set of labeled examples S if T_i does not cover any negative example in S. A term T_i is said to *imply* another term T_j if every positive point of T_i is also a positive point of T_j. We similarly say that a term T implies a DNF formula ϕ, or that a DNF formula ϕ_1 implies another DNF formula ϕ_2. Two different DNF formulas ϕ_1 and ϕ_2 are said to be *logically equivalent* if each implies the other, *i.e.,* if they are different syntactic representations of the same Boolean function. Throughout this chapter we will use the Greek letter ϕ to denote formulas (which are syntactic objects) and Roman letters $f, g, \ldots$ to denote Boolean functions (which are abstract mappings from $\{0,1\}^n$ to $\{0,1\}$).

We write $\mathcal{D}_s$ to denote the class of "exactly-s-term" DNF formulas; this is the class of all Boolean functions $f : \{0,1\} \to \{0,1\}^n$ that have some s-term DNF formula representation and have no s'-term DNF formula representation for any $s' < s$. Similarly, we write $\mathcal{D}_{\leq s}$ to denote the class of "at-most-s-term" DNF formulas, which is $\mathcal{D}_{\leq s} = \cup_{s' \leq s}\mathcal{D}_{s'}$. Note that the elements of $\mathcal{D}_s$ and $\mathcal{D}_{\leq s}$ are "semantic" functions, not syntactic formulas. The class $\mathcal{D}_{\leq s}$ corresponds to the standard notion of "s-term DNF formulas" which is a well studied concept class in computational learning theory. Note that the identically-false concept is computed by an empty disjunction of terms, *i.e.,* a zero-term DNF formula, and thus this concept belongs to $\mathcal{D}_{\leq s}$ for all s. The concept classes of exactly-s-term monotone DNF formulas and at-most-s-term monotone DNF formulas are denoted $\mathcal{M}_s$ and $\mathcal{M}_{\leq s}$ and are defined analogously with $\mathcal{D}_s$ and $\mathcal{D}_{\leq s}$ above. The following fact is well known:

Fact 60. *If $f \in \mathcal{M}_s$ then there is a unique (up to ordering of the terms) s-term monotone DNF representation $\phi = T_1 \vee \cdots \vee T_s$ for f.*

7.3 Monotone DNF formulas

Worst-case teaching dimension of at-most-s-term monotone DNF formulas. Here we state upper and lower bounds on the worst-case teaching dimension of $\mathcal{M}_{\leq s}$.

Theorem 61. *The teaching dimension of $\mathcal{M}_{\leq s}$ is at most $n^s + s$.*

Proof. Let f be an element of $\mathcal{M}_k$ for some $k \leq s$. We have that f is represented by a unique monotone DNF formula $\phi = T_1 \vee \cdots \vee T_k$, where each T_k corresponds to a minterm (minimal satisfying assignment) of f. For the rest of the proof we will view each term T_i as the set of variables that it contains; note that these sets are pairwise incomparable, *i.e.,* no T_i is contained in any other T_j.

We will show that the following set of examples is a teaching set for f:

- For each term T_i in $T_1, \ldots, T_k$ we give the positive example $\mathbf{0}|_{T_i=1}$; this is clearly at most s examples.
- We also give a set of negative examples which consists of precisely those examples that have exactly one variable of each term set to zero and all other variables set to one. In other words, for every set $S \subseteq \cup_{i=1}^{k} T_i$ that satisfies $|S \cap T_i| = 1$ for all i, we give the example $\mathbf{1}|_{S=0}$. Since there are at most n^s ways to choose exactly one variable from each of the s terms, this is at most n^s examples.

We first note that any $g \in \mathcal{M}_{\leq s}$ (in fact any monotone function g) that is consistent with the negative set must label negative any assignment which does not satisfy at least one of the terms $T_1, \ldots, T_k$. This is because for any assignment y which satisfies none of the k terms, there is an example y' in the negative set such that $y \leq y'$ with respect to the bitwise partial order on $\{0,1\}^n$. Since $g(y') = 0$ and g is monotone, this implies that $g(y) = 0$.

It follows that for all T_i, for all $x_j \in T_i$, the example $\mathbf{0}|_{(T_i \setminus x_j)=1}$ must be negative under f. Thus for each term T_i, we have that $\mathbf{0}|_{T_i=1}$ is positive while flipping any positive bit in $\mathbf{0}|_{T_i=1}$ makes f negative. Consequently, any $g \in \mathcal{M}_{\leq s}$ which is consistent with both the positive and negative examples must contain each of the terms $T_1, \ldots, T_k$. If $k = s$ then since g cannot contain any other terms, we must have that g is equivalent to f. If $k < s$, suppose that g contains some other non-redundant term T_{s+1}. Then there must be an assignment that is positive under g but which does not satisfy any of $T_1, \ldots, T_k$. The negative set shows that this is not possible. ■

Theorem 62. *Given s, let $s' \leq s$ be any value such that $(s'-1)$ divides n. Then the teaching dimension of $\mathcal{M}_{\leq s}$ is at least $(\frac{n}{s'-1})^{s'-1}$.*

Proof. We exhibit a concept in $\mathcal{M}_{\leq s-1}$ whose teaching set must contain all the negative examples in the teaching set for the proof of Theorem 61 in order to disambiguate it from various concepts in $\mathcal{M}_s$. Let $d = n/(s'-1)$ and consider the tribes function, $\text{Tribes}_d^{s'-1}$ (see Definition 1). Suppose not all of the negative examples from the proof of Theorem 61 are part of a teaching set for $\text{Tribes}_d^{s'-1}$, *i.e.,* that there is some $S \subseteq \cup_{i=1}^{s'-1} T_i$ with $|S \cap T_i| = 1$ such that $\mathbf{1}|_{S=0} \notin TS(\text{Tribes}_d^{s'-1})$. Let T_S be the term exactly satisfied by $\mathbf{1}|_{S=0}$ so that T_S includes variable x_i if and only if the i'th bit of $\mathbf{1}|_{S=0}$ is set to 1. Then the concept $f = \text{Tribes}_d^{s'-1} \vee T_S$ will label $TS(\text{Tribes}_d^{s'-1})$ consistently with $\text{Tribes}_d^{s'-1}$. Clearly any positive example in $TS(\text{Tribes}_d^{s'-1})$ will also be positive under f. Take any negative example $y \in TS(\text{Tribes}_d^{s'-1})$. Unless y satisfies T_S, it is negative under f. But to satisfy T_S, $y \geq \mathbf{1}|_{S=0}$. This is impossible, since if $y > \mathbf{1}|_{S=0}$ it would be a positive example, and $y = \mathbf{1}|_{S=0}$ is not in the teaching set by assumption.

Thus $TS(\text{Tribes}_d^{s'-1})$ must contain a negative example for every $S \subseteq \cup_{i=1}^{s'-1} T_i$ satisfying $|S \cap T_i| = 1$ for all i. For $\text{Tribes}_d^{s'-1}$ there are $d^{s'-1} = (\frac{n}{s'-1})^{s'-1}$ such sets. ■

Average-case teaching dimension of at-most-s-term monotone DNF formulas. We now prove Theorem 58. The idea is to show that almost every at-most-s-term monotone DNF formula in fact has exactly s terms; as we shall see, these exactly-s-term monotone

DNF formulas can be taught very efficiently with $O(ns)$ examples. The remaining concepts are so few that they can be handled with a brute-force approach and the overall average teaching dimension will still be $O(ns)$.

We start with a simple lemma from [GK92]:

Lemma 63 ([GK92])**.** *Let c be any concept in $\mathcal{M}_s$. Then the teaching dimension of c with respect to $\mathcal{M}_{\leq s}$ is at most $(n+1)s$.*

Proof sketch: Let $\phi = T_1 \vee \cdots \vee T_s$ be the unique s-term monotone DNF formula for c. For each $i = 1, \ldots, s$ the teaching set contains the positive example $\mathbf{0}|_{T_i=1}$ and contains $|T_i|$ many negative examples which are the neighbors of $\mathbf{0}|_{T_i=1}$ that are obtained by setting one of the 1s to 0. Each of the s terms thus contributes at most $n+1$ examples; an easy argument based on Fact 60 given in [GK92] shows that this is indeed a teaching set.

Lemma 64. *For $1 \leq i < \frac{1}{4}e^{\frac{n}{72}}$, we have $\frac{2^{ni-1}}{i!} \leq |\mathcal{M}_i| \leq \frac{2^{ni}}{i!}$.*

Proof. The upper bound is easy: the number of i-term monotone DNF formulas is at most the number of ways to choose i terms from the set of all 2^n many monotone terms over variables $x_1, \ldots, x_n$. The latter quantity is $\binom{2^n}{i} \leq \frac{2^{ni}}{i!}$.

For the lower bound we consider all 2^{ni} ways to select a sequence of i terms (with replacement) from the set of all 2^n possible monotone terms. We show that at least half of these 2^{ni} ways result in a sequence $T_1, \ldots, T_i$ of terms which are pairwise incomparable, *i.e.,* no T_i implies any other T_j. Each such sequence yields an i-term monotone DNF formula, and each such monotone DNF formula occurs $i!$ times because of different orderings of the terms in a sequence. This gives the lower bound.

Note that a collection of i monotone terms $T_1, \ldots, T_i$ will be pairwise incomparable if the following two conditions hold: (1) Each of the i terms contains between $5n/12$ and $7n/12$ many variables; and (2) Viewing each term T_i as a set of variables, for any $j \neq k$ the symmetric difference $|T_j \Delta T_k|$ is of size at least $n/4$. (This is because if $|T_j|, |T_k| \in [5n/12, 7n/12]$ and $T_j \subseteq T_k$, then the symmetric difference must be of size at most $n/6$.)

For condition (1), Hoeffding's bound implies that a uniformly selected monotone term T will contain fewer than $5n/12$ or more than $7n/12$ many variables with probability at most $2e^{-n/72}$, so a union bound gives that condition (1) fails with probability at most $2ie^{-n/72}$.

For condition (2), observe that given two uniform random terms T_j, T_k, each variable x_ℓ is independently in their symmetric difference with probability 1/2. Thus Hoeffding's bound implies that $|T_j \Delta T_k| < n/4$ with probability at most $e^{-n/8}$. By a union bound, the probability that condition (2) fails is at most $\binom{i}{2}e^{-n/8}$. Thus for $i < \frac{1}{4}e^{\frac{n}{72}}$, the probability that conditions (1) and (2) both hold is at least 1/2. ■

Fix $1 \leq s \leq \frac{1}{4}e^{\frac{n}{72}}$. It is easy to check that by Lemma 64, for any $k < s$ we have $|\mathcal{M}_k| < \frac{1}{2}|\mathcal{M}_{k+1}|$. Thus (again by Lemma 64) we have $|\mathcal{M}_{\leq s-1}| \leq \frac{2^{ns-n+1}}{(s-1)!}$ while $|\mathcal{M}_s| \geq \frac{2^{ns-1}}{s!}$. Combining these bounds gives that $\frac{|\mathcal{M}_s|}{|\mathcal{M}_{\leq s-1}|} \geq \frac{2^n}{4s}$. By Lemma 63, each concept $c \in \mathcal{M}_{\leq s}$ which is in $\mathcal{M}_s$ can be taught using $n(s+1)$ examples. Each of the remaining concepts can surely be taught using at most 2^n examples. We thus have that the average teaching dimension of $\mathcal{M}_{\leq s}$ is at most:

$$\frac{(n+1)s|\mathcal{M}_s| + 2^n|\mathcal{M}_{\leq s-1}|}{|\mathcal{M}_s| + |\mathcal{M}_{\leq s-1}|} \leq (n+1)s + \frac{2^n}{1 + 2^n/4s} \leq (n+1)s + 4s,$$

giving us the following result which is a slightly sharper version of Theorem 58:

Theorem 65. *Let s be any value $1 \leq s \leq \frac{1}{4}e^{\frac{n}{72}}$. The class $\mathcal{M}_{\leq s}$ of at-most-s-term monotone DNF formulas has average teaching dimension at most $s(n+5)$.*

Note that if $s > \frac{1}{4}e^{\frac{n}{72}}$, then 2^n is bounded by some fixed polynomial in s, and thus the worst-case teaching dimension 2^n is actually $\text{poly}(n, s)$ for such a large s. This gives the following corollary which says that the class of at-most-s-term monotone DNF formulas is efficiently teachable on average for all possible values of s:

Corollary 66. *Let s be any value $1 \leq s \leq 2^n$. The class $\mathcal{M}_{\leq s}$ of at-most-s-term monotone DNF formulas has average teaching dimension* $\text{poly}(n, s)$.

7.4 DNF Formulas

Now we will tackle the teaching dimension of the unrestricted class of size-at-most-s DNF formulas. The high-level approach is similar to the monotone case, but the details are more complicated. The idea is to identify a subset $\mathcal{S}$ of $\mathcal{D}_{\leq s}$ and show that (i) any function $f \in \mathcal{S}$ can be uniquely specified within all of $\mathcal{D}_{\leq s}$ using only $O(ns)$ examples; and (ii) at most a $\frac{O(s)}{2^n}$ fraction of all functions in $\mathcal{D}_{\leq s}$ do not belong to $\mathcal{S}$. Given (i) and (ii) it is easy to conclude that the average teaching dimension of $\mathcal{D}_{\leq s}$ is $O(ns)$.

The challenge is to devise a set $\mathcal{S}$ that satisfies both conditions (i) and (ii). In the monotone case, using Fact 60, it was easy to show that $\mathcal{M}_s$ is an easy-to-teach subset, but non-monotone DNF formulas are much more complicated (no analogue of Fact 60 holds for non-monotone DNF formulas) and it is not at all clear that all functions in $\mathcal{D}_s$ are easy to teach. Thus we must use a more complicated set $\mathcal{S}$ of easy-to-teach functions; we define this set and prove that it is indeed easy to teach in Section 7.4.2. (This argument uses Balbach's results for exactly-2-term DNF formulas.) The argument that (ii) holds for $\mathcal{S}$ is correspondingly more complex than the counting argument for monotone DNF formulas because of $\mathcal{S}$'s more involved structure; we give this in Section 7.4.3.

7.4.1 Preliminaries

We will borrow some terminology from Balbach [Bal05]. Two terms T_i and T_j have a *strong difference* at k if $T_i[k], T_j[k] \in \{0,1\}$ and $T_i[k] \neq T_j[k]$ (*e.g.,* $x_1\bar{x}_5x_6$ and $\bar{x}_5\bar{x}_6x_{12}x_{23}$ have a strong difference at position 6). Two terms have a *weak difference* at k if $T_i[k] \in \{0,1\}$ and $T_j[k] = *$ or vice-versa. Two weak differences at positions k and ℓ are *of the same kind* if $T_i[k], T_i[\ell] \in \{0,1\}$ and $T_j[k] = T_j[\ell] = *$ or vice-versa, that is both $*$'s occur in the same term (*e.g.,* $\bar{x}_5x_6$ and $\bar{x}_5\bar{x}_6x_{12}x_{23}$ have two weak differences of the same kind at positions 12 and 23). Two weak differences at positions k and ℓ are *of different kinds* if $T_i[k], T_j[\ell] \in \{0,1\}$ and $T_j[k] = T_i[\ell] = *$ or vice-versa (*e.g.,* $\bar{x}_5x_6$ and $\bar{x}_5x_{12}$ have two weak differences of different kinds at positions 6 and 12).

Now we introduce some new terminology. Given $y \in \{0,1\}^n$ which satisfies a term T, we denote by $N_T(y)$ the set consisting of y and all its neighbors that do *not* satisfy T. A satisfying assignment $y \in \{0,1\}^n$ of a term T in ϕ is called a *cogent corner point of* T if all the neighbors of y that satisfy ϕ satisfy T, and all the neighbors that do not satisfy T do not satisfy ϕ. Note that if y is a cogent corner point of T, then each of the neighbors of y in $N_T(y)$ does not satisfy ϕ. A pair of points $y, z \in \{0,1\}^n$ that satisfy a term T are said to be *antipodal around* T if $y_k = \overline{z}_k$ for all k such that $T[k] = *$. A pair of points are *cogent antipodal points around* T if they are both cogent corner points of T and antipodal around

T. This leads us to our first preliminary lemma:

Lemma 67. *Let $\phi = T_1 \vee \cdots \vee T_s$ be any DNF formula. Let y be a cogent corner point of T_i. Any $\widehat{T}$ that covers y and is compatible with $N_{T_i}(y)$ must imply T_i.*

Proof. Let $\widehat{T}$ be any term that covers y. Observe that for each literal ℓ in T_i, if $\widehat{T}$ did not contain ℓ then $\widehat{T}$ would not be compatible with $N_{T_i}(y)$ since the corresponding negative neighbor of y is contained in $N_{T_i}(y)$ but would be covered by $\widehat{T}$. It follows that every literal in T_i is also present in $\widehat{T}$, and consequently $\widehat{T}$ implies T_i. ∎

Two terms are said to be *close* if they have at most one strong difference. Note that there is no strong difference between two terms if and only if they have some satisfying assignment in common, and there is one strong difference between two terms if and only if they have neighboring satisfying assignments.

Given a Boolean function $f : \{0,1\} \to \{0,1\}^n$, we let G_f denote the undirected graph whose vertices are the satisfying assignments of f and whose edges are pairs of neighboring satisfying assignments. A *cluster* C of f is a set of satisfying assignments that form a connected component in G_f. We sometimes abuse notation and write C to refer to the Boolean function whose satisfying assignments are precisely the points in C. We say that a DNF formula ϕ computes cluster C if the set of satisfying assignments for ϕ is precisely C. The DNF-size of a cluster C is the minimum number of terms in any DNF formula that computes C. For intuition, we can view a cluster as being a connected set of positive points that have a "buffer" of negative points separating them from all other positive points. The following lemma is immediate:

Lemma 68. *Let f be an element of $\mathcal{D}_s$, i.e., f is an exactly-s-term DNF formula. Let $C_1, \ldots, C_r$ be the clusters of f. Then* $\text{DNF-size}(C_1) + \cdots + \text{DNF-size}(C_r) = s$.

7.4.2 Teaching $\mathcal{S}$

We are now ready to define our "nice" (easy to teach) subset $\mathcal{S} \subseteq \mathcal{D}_{\leq s}$ of size-at-most-s DNF formulas. (We emphasize that $\mathcal{S}$ is a set of functions, not of DNF expressions.) $\mathcal{S}$ consists of those exactly-s-term DNF formulas (so in fact $\mathcal{S} \subseteq \mathcal{D}_s$) all of whose clusters either: (1) have DNF-size 1; (2) have DNF-size 2; or (3) have DNF-size k, for some k, and are computed by a DNF formula $\phi = T_1 \vee \cdots \vee T_k$ in which each T_i has a pair of cogent antipodal points around it.

Note that if a cluster has DNF-size 1, then it clearly satisfies condition (3) above (in fact every pair of antipodal points for the term is cogent). Thus we can simplify the description of $\mathcal{S}$: it is the set of all exactly s-term DNF formulas all of whose clusters either: (i) have DNF-size k and are computed by a DNF formula $\phi = T_1 \vee \cdots \vee T_k$ in which each T_i has a pair of cogent antipodal points around it, or (ii) have DNF-size exactly 2. (Note that there do in fact exist Boolean functions of DNF-size 2 for which any two-term representation $T_1 \vee T_2$ has some term T_i with no pair of cogent antipodal points around it, *e.g.*, $\overline{x}_1\overline{x}_3 \vee x_2x_3$, and thus condition (ii) is non-redundant.)

The teaching set for functions in $\mathcal{S}$. We will use the following theorem due to Balbach [Bal05]:

Theorem 69. *Let c be any element of $\mathcal{D}_2$ (i.e., an exactly-2-term DNF formula). The teaching dimension of c with respect to $\mathcal{D}_{\leq 2}$ is at most $2n + 4$.*

The teaching set specified in [Bal05] to prove Theorem 69 consists of at most 5 positive points along with some negative points. Given $f \in \mathcal{D}_2$, we define $BTS(f)$ to be the union of the teaching set specified in [Bal05] together with all negative neighbors of the (at most five) positive points described above (the set specified in [Bal05] already contains some of these points). With this definition a straightforward consequence of the analysis of [Bal05] is the following:

Lemma 70. *Let $\phi = T_1 \vee \cdots \vee T_s$ be a DNF formula that has a cluster C with DNF-size 2. Let $BTS(C)$ be as described above. Let y be a satisfying assignment for ϕ that is contained in C. Then any term $\widehat{T}$ that covers y and is consistent with $BTS(C)$ must imply C.*

Given any function $f \in \mathcal{S}$, our teaching set $TS(f)$ for f will be as follows. For each cluster C of f, if C:

- **satisfies condition (i):** then for each term T_i described in condition (i), the set $TS(f)$ contains a pair y, z of cogent antipodal points for T_i (these are positive examples) and contains all negative neighbors of these two positive examples (*i.e.*, $TS(f)$

contains $N_{T_i}(y)$ and $N_{T_i}(z)$). Thus $TS(f)$ includes at most $k(2+2n)$ many points from such a cluster, where $k = \text{DNF-size}(C)$.

- **does not satisfy condition (i) but satisfies (ii):** then we will give the set $BTS(C)$ described above. By Theorem 69 and the definition of $BTS(C)$, we have that $BTS(C)$ contains at most $7n+4$ points.

Lemma 68 now implies that $TS(f)$ contains at most $O(ns)$ points.

Correctness of the teaching set construction. We now prove that the set $TS(f)$ is indeed a teaching set that uniquely specifies f within all of $\mathcal{D}_{\leq s}$.

We first observe that any term compatible with $TS(f)$ can only cover positive examples from one cluster of ϕ.

Lemma 71. *Let y be any positive example in $TS(f)$ and let T be any term that covers y and is compatible with $TS(f)$. Let C be the cluster of ϕ that covers y. Then if z is any positive example in $TS(f)$ that is not covered by C, T does not cover z.*

Proof. If C satisfies condition (i) then y must be a cogent corner point and Lemma 67 gives the desired conclusion. If C does not satisfy (i) but satisfies (ii), then the conclusion follows from Lemma 70. ■

The next two lemmas show that any set of terms that covers the positive examples of a given cluster must precisely compute the entire cluster and only the cluster of the original function:

Lemma 72. *Let C be any case (i) cluster of DNF-size k. Let P_C be the intersection of the positive examples in $TS(f)$ with C. Let $\widehat{T}_1, \ldots, \widehat{T}_j$ be any set of $j \leq k$ terms such that the DNF formula $\widehat{T}_1 \vee \cdots \vee \widehat{T}_j$ both: (a) is compatible with $TS(f)$, and (b) covers every point in P_C. Then it must be the case that $j = k$ and $\widehat{T}_1 \vee \cdots \vee \widehat{T}_j$ exactly computes C (in fact each term $\widehat{T}_i$ is equivalent to T_i up to reordering).*

Proof. By Lemma 67, a term $\widehat{T}$ that covers a cogent antipodal point from term T_i cannot cover any of the other $2k-2$ cogent antipodal points from other terms, and thus we must have $j = k$ since fewer than k terms cannot cover all of P_C. Moreover, any term $\widehat{T}_i$ must cover a pair of antipodal points corresponding to a single term (which without loss of generality we call T_i). For each antipodal pair corresponding to a term T_i, the covering term $\widehat{T_i}$ must be of size at least $|T_i|$, and since they are cogent antipodal points, the covering term cannot be any longer than $|T_i|$, so in fact we have that $\widehat{T_i}$ and T_i are identical. This proves the lemma. ■

Lemma 73. *Let C be any case (ii) cluster. Let P_C be the intersection of the positive examples in $TS(f)$ with C. Let $\widehat{T}_1, \dots, \widehat{T}_j$ be any set of $j \leq 2$ terms such that the DNF formula $\widehat{T}_1 \vee \cdots \vee \widehat{T}_j$ both: (a) is compatible with $TS(f)$, and (b) covers every point in P_C. Then it must be the case that $j = 2$ and $\widehat{T}_1 \vee \widehat{T}_2$ exactly computes C.*

Proof. The fact that $BTS(C)$ is a teaching set (for the exactly-2-term DNF formula corresponding to C, relative to $\mathcal{D}_{\leq 2}$) implies the desired result, since no single term or 2-term DNF formula not equivalent to C can be consistent with $BTS(C)$, and any DNF formula $\widehat{T}_1 \vee \cdots \vee \widehat{T}_j$ as specified in the lemma must be consistent with $BTS(C)$. ■

The pieces are in place for us to prove our theorem:

Theorem 74. *For any $f \in \mathcal{S}$, the set $TS(f)$ uniquely specifies f within $\mathcal{D}_{\leq s}$.*

Proof. By Lemma 71, positive points from each cluster can only be covered by terms that do not include any positive points from other clusters. By Lemmas 72 and 73, for each cluster C, the minimum number of terms required to cover all positive points in the cluster (and still be compatible with $TS(f)$) is precisely the DNF-size of C. Since f is an exactly-s-term DNF formula, Lemma 68 implies that using more than DNF-size(C) many terms to cover all the positive points in any cluster C will "short-change" some other cluster and cause some positive point to be uncovered. Thus any at-most-s-term DNF formula ϕ that is consistent with $TS(f)$ must have the property that for each cluster C, at most DNF-size(C) of its terms cover the points in P_C; so by Lemmas 72 and 73, these terms exactly compute C, and thus ϕ must exactly compute f. ■

7.4.3 Average-Case Teaching Dimension of DNFs

In this section we will show that all but at most a $\frac{O(s)}{2^n}$ fraction of functions in $\mathcal{D}_{\leq s}$ are in fact in $\mathcal{S}$. We do this by showing that at least a $1 - \frac{O(s)}{2^n}$ fraction of functions in $\mathcal{D}_{\leq s}$ are in the easy-to-teach set $\mathcal{S}$, *i.e.,* they belong to $\mathcal{D}_s$ and are such that each cluster satisfies either condition (i) or (ii) from Section 7.4.2. Since we have shown that each $f \in \mathcal{S}$ can be uniquely specified within $\mathcal{D}_{\leq s}$ using $O(ns)$ examples, this will easily yield that the average teaching dimension over all of $\mathcal{D}_{\leq s}$ is $O(ns)$.

First we show that most functions in $\mathcal{D}_{\leq s}$ are in fact in $\mathcal{D}_s$. We can bound $|\mathcal{D}_i|$ using the same approach as we did for monotone DNF formulas.

Lemma 75. *For $i < (9/7)^{n/3}$, we have $\frac{1}{2} \cdot \frac{3^{ni}}{i!} \leq |\mathcal{D}_i| \leq \frac{3^{ni}}{i!}$.*

Proof. As in Lemma 64, the upper bound is easy; we may bound the number of functions in $\mathcal{D}_i$ by the number of ways to choose i terms from the set of all 3^n possible terms over variables $x_1, \ldots, x_n$. This is $\binom{3^n}{i} \leq \frac{3^{ni}}{i!}$.

For the lower bound, we first note that a DNF formula consisting of i terms that are all pairwise far from each other cannot be logically equivalent to any other DNF formula over a different set of i terms. We will show that at least half of all 3^{ni} possible sequences of i terms have the property that all i terms in the sequence are pairwise far from each other; this gives the lower bound (since each such set of i terms can be ordered in $i!$ different ways).

So consider a uniform random draw of i terms $T_1, \ldots, T_i$ from the set of all 3^n possible terms. The probability that T_1 and T_2 are close is the probability that they have no strong differences plus the probability that they have exactly one strong difference. This is $(7/9)^n + n(7/9)^{n-1}(2/9) < (n+1)(7/9)^n$. By a union bound over all pairs of terms, the probability that any pair of terms is close at most $\binom{i}{2}(n+1)(7/9)^n$ which is less than 1/2 for $i < (9/7)^{n/3}$. ■

As in Section 7.3, as a corollary we have that $\frac{|\mathcal{D}_s|}{|\mathcal{D}_{\leq s-1}|} \geq \frac{3^n}{4s}$ for $s \leq (9/7)^{n/3}$.

We now bound the number of DNF formulas in $\mathcal{D}_s$ that are not in $\mathcal{S}$. To do this, we consider choosing s terms at random with replacement from all 3^n terms:

Lemma 76. *Fix any $s \leq (9/8)^{n/25}$. Let $f = T_1, \ldots, T_s$ be a sequence of exactly s terms selected by independently choosing each T_i uniformly from the set of all 3^n possible terms. Let $A(T_i)$ denote the event that term T_i in f has no cogent antipodal pairs, and $B(T_i)$*

denote the event that there is more than one other term close to T_i in f. Then $\Pr[\exists T_i \in f : A(T_i) \wedge B(T_i)] \le \frac{O(s)}{2^n}$*, where the probability is taken over the choice of f.*

Using Lemma 76 we can bound the number of functions $f \in \mathcal{D}_s$ that are not in $\mathcal{S}$. If $f \in \mathcal{D}_s \setminus \mathcal{S}$, then f must have a DNF representation $\phi = T_1 \vee \cdots \vee T_s$ in which some term T_i:

1. has no cogent antipodal pairs, and

2. has at least two other terms T_j, T_k that are close to it.

(If there were no such term, then for any representation $\phi = T_1 \vee \cdots \vee T_s$ for the function f, every T_i is contained in either a cluster of DNF-size 1 or 2, or a cluster of DNF-size k with a pair of good antipodal points around it. But then ϕ would be in $\mathcal{S}$.)

We will call such a syntactic DNF formula "bad." Lemma 76 tells us that the number of bad syntactic formulas is at most $\frac{3^{ns}O(s)}{2^n}$, since there are 3^{ns} syntactic formulas. Notice that any bad formula ϕ must have s distinct terms (since the function it computes belongs to $\mathcal{D}_s$), and since these terms can be ordered in $s!$ different ways, there are at least $s!$ bad formulas that compute the same function as ϕ. Consequently the number of bad functions in $\mathcal{D}_s$, $|\mathcal{D}_s \setminus \mathcal{S}|$, is at most $\frac{O(s)}{2^n}\frac{3^{ns}}{s!}$. By Lemma 75, $|\mathcal{D}_s|$ is at at least $\frac{3^{ns}}{2s!}$. This gives the following:

Corollary 77. $\frac{|\mathcal{D}_s \setminus \mathcal{S}|}{|\mathcal{D}_s|} \le \frac{O(s)}{2^n}$.

We now proceed to prove Lemma 76.

Proof. The bulk of the argument is in showing that $\Pr[A(T_1) \wedge B(T_1)]$ is at most $O(1) \cdot 2^{-n}$; once this is shown a union bound gives the final result.

We condition on the outcome of T_1. Using the fact that each variable occurs independently in T_1 (either positive or negated) with probability $2/3$, a Chernoff bound gives that $\Pr[|T_1| < .08n] \le 2^{-n}$, so we have that

$$\Pr[A(T_1) \wedge B(T_1)] \le 2^{-n} + \sum_{\mathcal{T} : |\mathcal{T}| \ge .08n} \Pr[A(T_1) \wedge B(T_1) \mid (T_1 = \mathcal{T})] \cdot \Pr[T_1 = \mathcal{T}].$$

Next we show that $\Pr[A(T_1) \wedge B(T_1) \mid (T_1 = \mathcal{T})] \le O(1) \cdot 2^{-n}$ for every $\mathcal{T}$ satisfying $|\mathcal{T}| \ge .08n$; this implies an $O(1) \cdot 2^{-n}$ bound on $\Pr[A(T_1) \wedge B(T_1)]$. To do this we consider a third event which we denote by $C(T_1)$; this is the event that T_1 is close to at most 25 of the terms $T_2, \ldots, T_s$. Clearly we have that

$$\begin{aligned}\Pr[A(T_1) \wedge B(T_1) \mid (T_1 = \mathcal{T})] = \Pr[A(T_1) \wedge B(T_1) \wedge \neg C(T_1) \mid (T_1 = \mathcal{T})] \\ + \Pr[A(T_1) \wedge B(T_1) \wedge C(T_1) \mid (T_1 = \mathcal{T})] \end{aligned} \tag{7.1}$$

and we proceed by bounding each of the terms in (7.1).

The first term is at most $\Pr[\neg C(T_1) \mid (T_1 = \mathcal{T})]$. Fix any $\alpha \in [.08, 1]$ and any term $\mathcal{T}$ of length αn, and fix $T_1 = \mathcal{T}$. Then the probability (over a random draw of T_2 as in the statement of the lemma) that T_2 is close to T_1 is the probability that T_1 and T_2 have one strong difference plus the probability that T_1 and T_2 have no strong difference, which is exactly $\alpha n \frac{1}{3} \left(\frac{2}{3}\right)^{\alpha n - 1} + \left(\frac{2}{3}\right)^{\alpha n} \leq 2\alpha n \left(\frac{2}{3}\right)^{\alpha n}$. Using the independence of the terms $T_2, \ldots, T_s$ and a union bound, it follows that the probability that there exists any set of K terms in f which are all close to T_1 is at most $\binom{s}{K}(2\alpha n)^K \left(\frac{2}{3}\right)^{K\alpha n}$. It is not hard to verify that for any $1 \leq s \leq (9/8)^{n/25}$, any $K \geq 26$, and any $\alpha \in [.08, 1]$, this quantity is asymptotically less than 2^{-n}.

It remains to bound the second term of (7.1) by $O(1) \cdot 2^{-n}$. We do this using the following observation:

Proposition 78. *Let $f = T_1, \ldots, T_s$ be any sequence of s terms. If T_1 has no cogent antipodal pairs with respect to f and is close to at most K of the terms $T_2, \ldots, T_s$, then there must be some term among $T_2, \ldots, T_s$ that is close to T_1 and contains at most $k = \lceil \log K \rceil + 1$ variables not already in T_1.*

Proof. We show that if every term in f close to T_1 contains more than k variables not already in T_1, there must remain some cogent antipodal pair for T_1. Let r be the number of variables in T_1 and let $\ell = n - r$. For any $z \in \{0,1\}^\ell$ let $Q_{T_1}(z)$ denote the set of points in $\{0,1\}^n$ consisting of the antipodal pair induced by z on T_1 (these two points each satisfy T_1) and the $2r$ neighbors of these points that do not satisfy T_1. Thus $Q_{T_1}(z) = Q_{T_1}(\overline{z})$, and there are $2^{\ell-1}$ distinct $Q_{T_1}(z)$, each representing a possible cogent antipodal pair.

Consider a term T_i that is close to T_1, and partition its satisfying assignments according to the 2^ℓ assignments on the ℓ variables not contained in T_1. Since T_i will only eliminate the cogent antipodal pair represented by the neighborhood $Q_{T_1}(z)$ if it covers some point in $Q_{T_1}(z)$, T_i can only eliminate as many cogent antipodal pairs as it has partitions. But if T_i contains more than k of the ℓ variables not already in T_1, then there are fewer than $2^{\ell-k}$ different ways to set the ℓ bits outside of T_1 to construct a satisfying assignment for T_i, and T_i has fewer than $2^{\ell-k}$ different partitions. Since by assumption there are at most $K \leq 2^{k-1}$ terms close to T_1, there are fewer than $2^{k-1} \cdot 2^{\ell-k} = 2^{\ell-1}$ different $Q_T(z)$ eliminated, and T must have a cogent antipodal pair left. ■

By Proposition 78, we know that if $A(T_1)$ occurs (T_1 has no cogent antipodal pairs) and $C(T_1)$ occurs (T_1 is close to no more than $K = 25$ other terms), then there must be some term close to T_1 that has at most $k = 6$ variables not in T_1. Thus we have that $\Pr[A(T_1) \wedge B(T_1) \wedge C(T_1) \mid (T_1 = \mathcal{T})]$ is at most the probability there exist two terms close to T_1, one of which contains at most $k = 6$ variables not in T_1. We saw earlier that the probability that a randomly chosen term is close to T_1 is at most $2\alpha n(2/3)^{\alpha n}$. However, the probability that a randomly chosen term is close to T_1 *and* contains at most 6 variables not in T_1 is much lower (because almost all of the $(1-\alpha)n$ variables not in T_1 are constrained to be absent from the term); more precisely this probability is at most $2\alpha n\binom{(1-\alpha)n}{6}\left(\frac{2}{3}\right)^{\alpha n}\left(\frac{1}{3}\right)^{(1-\alpha)n-6}$. A union bound over all possible pairs of terms gives us that the second term of (7.1) is at most $2\alpha n\binom{s}{2}\binom{(1-\alpha)n}{6}3^6\left(\frac{2}{3}\right)^{2\alpha n}\left(\frac{1}{3}\right)^{(1-\alpha)n}$. It is straightforward to check that this is at most $O(1)\cdot 2^{-n}$ for all $1 \leq s \leq (9/8)^{n/25}$ and all $\alpha \in [0,1]$.

Thus, we have bounded $\Pr[A(T_1) \wedge B(T_1)]$ by $O(1)\cdot 2^{-n}$. A union bound over the s terms gives that $\Pr[\exists T_i \in f : A(T_i) \wedge B(T_i)]$ is at most $O(s)2^{-n}$, and the lemma is proved. ■

Theorem 79. *Let $s \leq (9/8)^{n/25}$. The average teaching dimension of $\mathcal{D}_{\leq s}$, the class of DNF formulas over n variables with at most s terms, is $O(ns)$.*

Proof. Theorem 74 gives us that the teaching dimension of any concept in $\mathcal{S} \subset \mathcal{D}_s$ is $O(ns)$. By Lemma 75, we have that $|\mathcal{D}_{\leq s-1}| \leq \frac{4s}{3^n}|\mathcal{D}_s|$. This leaves us with $\mathcal{D}_s \setminus \mathcal{S}$, whose size we bounded by $\frac{O(s)}{2^n}|\mathcal{D}_s|$ in Corollary 77. Combining these bounds, we are ready to bound the average teaching number of $|\mathcal{D}_{\leq s}|$. Since we can teach any bad concept with at most 2^n

examples, the average teaching dimension is at most

$$\begin{aligned}\frac{O(ns)|\mathcal{S}| + 2^n(|\mathcal{D}_{\leq s-1}| + |\mathcal{D}_s \setminus \mathcal{S}|)}{|\mathcal{D}_s| + |\mathcal{D}_{\leq s-1}|} &\leq \frac{O(ns)|\mathcal{D}_s| + 2^n(\frac{4s}{3^n}|\mathcal{D}_s| + \frac{O(s)}{2^n}|\mathcal{D}_s|)}{|\mathcal{D}_s| + |\mathcal{D}_{\leq s-1}|} \\ &\leq O(ns) + (2/3)^n \cdot 4s + O(s) = O(ns)\end{aligned}$$

and the theorem is proved. ■

As in Corollary 66, we have $2^n \leq \text{poly}(s)$ if $s > (9/8)^{n/25}$, and thus the worst-case teaching dimension 2^n is actually $\text{poly}(n, s)$ for such large s. This gives the following corollary:

Corollary 80. *Let s be any value $1 \leq s \leq 2^n$. The class $\mathcal{D}_{\leq s}$ of at-most-s-term DNF formulas has average teaching dimension* $\text{poly}(n, s)$.

7.5 Teaching Dimension of k-Juntas

A Boolean function f over n variables depends on its i-th variable if there are two inputs $x, x' \in \{0,1\}^n$ that differ only in the i-th coordinate and that have $f(x) \neq f(x')$. Recall that a *k-junta* is a Boolean function which depends on at most k of its n input variables. The class of k-juntas is well studied in computational learning theory, see *e.g.,* [Blu03, MOS03, ABF$^+$04]. We write $\mathcal{J}_k$ to denote the class of Boolean functions $f : \{0,1\} \rightarrow \{0,1\}^n$ that depend on exactly k variables, and we write $\mathcal{J}_{\leq k}$ to denote the class $\mathcal{J}_{\leq k} = \cup_{k' \leq k} \mathcal{J}_{k'}$ of Boolean functions over $\{0,1\}^n$ that depend on at most k variables, *i.e.,* $\mathcal{J}_{\leq k}$ is the class of all k-juntas.

We analyze the worst-case and average-case teaching dimensions of the class of k-juntas, and show that while the worst-case teaching dimension has a logarithmic dependence on n, the average-case dimension has no dependence on n. Thus k-juntas are another natural concept class where there is a substantial asymptotic difference between the worst-case and average teaching dimensions.

Worst-Case teaching dimension of k-juntas. We recall the following:

Definition 81. *Let $k \leq n$. A set $S \subseteq \{0,1\}^n$ is said to be an (n, k)-*universal set *if for any $1 \leq i_1 < i_2 \ldots < i_k \leq n$, it holds that $\forall y \in \{0,1\}^k, \exists x \in S$ satisfying $(x_{i_1}, \ldots, x_{i_k}) = (y_1, \ldots, y_k)$*

Nearly matching upper and lower bounds are known for the size of (n, k)-universal sets:

Theorem 82 ([BS88])**.** *Let $k \le n$. Any (n, k)-universal set has size $\Omega(2^k \log n)$, and there exists an (n, k)-universal set of size $O(k2^k \log n)$.*

This straightforwardly yields:

Theorem 83. *The teaching dimension of the class $\mathcal{J}_{\le k}$ is at least $\Omega(2^k \log n)$ and at most $O(k2^k \log n)$.*

Proof. For the lower bound, we show that any teaching set for the identically-0 concept $c \equiv \mathbf{0}$ (which is a k-junta for any $k \ge 0$) must be an (n, k)-universal set. Suppose $S \subseteq \{0,1\}^n$ is not an (n, k)-universal set, *i.e.,* there is some $i_1 < \cdots < i_k$ and some $y \in \{0,1\}^k$ such that for every $x \in S$ we have $(x_{i_1}, \ldots, x_{i_k}) \ne (y_1 \ldots y_k)$. Then the k-junta defined as

$$c'(x) = \begin{cases} 1 & \text{if } (x_{i_1} \ldots x_{i_k}) = (y_1 \ldots y_k) \\ 0 & \text{otherwise} \end{cases}$$

labels S the same way as c.

Now we prove the upper bound. Let c be any k-junta that has $R = \{i_1, \ldots, i_r\}$ as its set of relevant variables (so $r \le k$). We describe a teaching set for c. For each relevant variable $i_j \in R$, there is a pair of examples $x, x' \in \{0,1\}^n$ that disagree only in their i_j-th bit and have $c(x) \ne c(x')$. Let the set S consist of these $2r$ examples together with an (n, k)-universal set; we will argue that S is a teaching set for c and thus prove the theorem.

Suppose that c' is some k-junta that is consistent with S. Clearly c' must depend on every variable in R or else it would label one of the first $2r$ examples differently from c. We claim that c' cannot depend on any additional variables. Suppose to the contrary that c' depends on exactly q additional variables $j_1, \ldots, j_q$. Given $a \in \{0,1\}^r$ and $b \in \{0,1\}^q$, let $V(a, b) = \{x \in \{0,1\}^n : (x_{i_1} \ldots x_{i_r}) = a \text{ and } (x_{j_1} \ldots x_{j_q}) = b\}$. Since c' depends on $j_1, \ldots, j_q$, there must be some $a \in \{0,1\}^r$ and $b \ne b' \in \{0,1\}^q$ such that all the examples in $V(a, b)$ take one value under c' while all the examples in $V(a, b')$ take the other value under c'. Furthermore, since S is an (n, k)-universal set and $|a| + |b| \le k$, S must contain some example x^1 from $V(a, b)$ and some example x^2 from $V(a, b')$. But c only depends on variables $i_1, \ldots, i_r$, so c assigns x^1 and x^2 the same label while c' does not. Thus c

and c' must have the exact same set of $r \le k$ relevant variables. Since they agree on an (n,k)-universal set, they agree for every setting of those r variables, and thus they agree on all of $\{0,1\}^n$. ■

Average-case teaching dimension of k-juntas. The idea is similar to the case of monotone DNF formulas: we show that k-juntas with exactly k relevant variables can be taught with 2^k examples (independent of n), and then use the fact that the overwhelming majority of k-juntas have exactly k relevant variables.

Lemma 84. *Let c be any concept in $\mathcal{J}_k$. Then the teaching dimension of c with respect to $\mathcal{J}_{\le k}$ is at most 2^k.*

Proof. Given any k-junta c with exactly k relevant variables, let S be the set of 2^k examples in which all irrelevant variables are always set to 0 and the relevant variables range over all 2^k possible settings. It is straightforward to see that S is a teaching set for c. ■

We now claim that $\frac{1}{2}\binom{n}{k}2^{2^k} \le |\mathcal{J}_k| \le \binom{n}{k}2^{2^k}$. The upper bound is clear since any k-junta can be specified by presenting k variables ($\binom{n}{k}$ possibilities) and a Boolean function on those k variables (2^{2^k} possibilities). The lower bound (which is very crude but sufficient for our purposes) follows from the easily verified fact that at least half of all 2^{2^k} functions on $\{0,1\}^k$ in fact depend on all k variables. It is easy to see from these bounds that $|\mathcal{J}_k|$ strictly increases with k for all k, and thus we have $|\mathcal{J}_{\le k-1}| \le (k-1)|\mathcal{J}_{k-1}| \le (k-1)\binom{n}{k-1}2^{2^{k-1}}$.

By Lemma 84 we can specify any function in $\mathcal{J}_k$ with at most 2^k examples, and by Theorem 82 we can specify any of the other functions in $\mathcal{J}_{\le k}$ (*i.e.*, any function in $\mathcal{J}_{\le k-1}$) with at most $O(k2^k \log n)$ many examples. It follows that the average teaching dimension of $\mathcal{J}_{\le k}$ is at most

$$\frac{2^k|\mathcal{J}_k| + O(k2^k\log n)\cdot|\mathcal{J}_{\le k-1}|}{|\mathcal{J}_k| + |\mathcal{J}_{\le k-1}|} \le 2^k + \frac{O(k2^k\log n)\cdot(k-1)\binom{n}{k-1}2^{2^{k-1}}}{\frac{1}{2}\binom{n}{k}2^{2^k}}.$$

The second term on the right simplifies to

$$\frac{O(k2^k\log n)\cdot k(k-1)}{2^{2^{k-1}}(n-k+1)}$$

which is easily seen to be $o(1)$ for any k. We have thus proved:

Theorem 85. *The average teaching dimension of the class $\mathcal{J}_{\leq k}$ of k-juntas is at most $2^k + o(1)$.*

7.6 Sparse GF_2 Polynomials

A GF_2 *polynomial* is a multilinear polynomial with 0/1 coefficients that maps $\{0,1\}^n$ to $\{0,1\}$ where all arithmetic is done modulo 2. Since addition mod 2 corresponds to parity and multiplication corresponds to AND, a GF_2 polynomial can be viewed as a parity of monotone conjunctions. It is well known, and not hard to show, that every Boolean function $f : \{0,1\} \to \{0,1\}^n$ has a unique GF_2 polynomial representation. (For example, the parity function has $x_1 \oplus \cdots \oplus x_n$ as its GF_2 polynomial, and $x_1 \vee x_2$ has $x_1 \oplus x_2 \oplus x_1x_2$.)

A natural measure of the size of a GF_2 polynomial is the number of monomials that it contains. In keeping with our usual notation, let $\mathcal{G}_s$ denote the class of all Boolean functions $f : \{0,1\} \to \{0,1\}^n$ that have GF_2 polynomial representations with exactly s monomials and let $\mathcal{G}_{\leq s}$ denote $\cup_{s' \leq s} \mathcal{G}_{s'}$. We sometimes refer to functions in $\mathcal{G}_{\leq s}$ as being *s-sparse GF_2* polynomials. The class of s-sparse GF_2 polynomials has been studied by several researchers in learning theory and complexity theory, see *e.g.,* [RB91, BM02, SS96].

Roth and Benedek [RB91] showed that any $f \in \mathcal{G}_{\leq s}$ is uniquely determined by the values it assumes on those $x \in \{0,1\}^n$ that contain at least $n - (1 + \lfloor \log_2 s \rfloor)$ many 1s. They also showed that it is in fact necessary to specify the value of f on every such point even in order to uniquely determine the parity (even or odd) of $|f^{-1}(1)|$ where f ranges over all of $\mathcal{G}_{\leq s}$. We thus have:

Theorem 86 ([RB91])**.** *Fix any $1 \leq s \leq 2^n$. The (worst-case) teaching dimension of $\mathcal{G}_{\leq s}$ is:*

$$\sum_{i=0}^{1+\lfloor \log_2 s \rfloor} \binom{n}{i}$$

(which is $n^{\Theta(\log s)}$ for s subexponential in n).

In contrast, we show that if s is sufficiently small, the average-case teaching dimension of $\mathcal{G}_{\leq s}$ is $O(ns)$:

Theorem 87. *Fix $1 \leq s \leq (1-\epsilon)\log_2 n$, where $\epsilon > 0$ is any constant. Then the average-case teaching dimension of $\mathcal{G}_{\leq s}$ is at most $ns + 2s$.*

For $s = \omega(1)$, $s < (1-\epsilon)\log_2 n$, this gives a superpolynomial separation between the worst-case and average-case teaching dimension of s-sparse GF_2 polynomials.

Proof of Theorem 87. We now define the "nice" (easy-to-teach) subset of $\mathcal{G}_{\leq s}$, in analogy with $\mathcal{S}$ in Section 7.4. We say that a function $f = M_1 \oplus \cdots \oplus M_s \in \mathcal{G}_s$ is *individuated* if for each $i = 1, \ldots, s$ there is some $j \in \{1, \ldots, n\}$ such that the variable x_j occurs in monomial M_i and does not occur in any of the other $s - 1$ monomials. Let $\mathcal{I} \subseteq \mathcal{G}_s$ denote the set of all functions in $\mathcal{G}_s$ that are individuated.

We first show that any function in $\mathcal{I}$ can be specified using few examples:

Lemma 88. *For any $f \in \mathcal{I}$, the teaching dimension of f with respect to $\mathcal{G}_{\leq s}$ is at most $ns + 2s - 1$.*

Proof. Given $x^1, \ldots, x^r \in \{0,1\}^n$, we write $\text{join}(x^1, \ldots, x^r)$ to denote the string $z \in \{0,1\}^n$ that has for all $i = 1, \ldots, n$, $z_i = \max\{x^1_i, \ldots, x^r_i\}$.

Let $f = M_1 \oplus \cdots \oplus M_s \in \mathcal{I}$ be any individuated GF_2 polynomial. For $i = 1, \ldots, s$ let y^i denote the minimal (with respect to bitwise $\leq$ ordering described above) assignment that satisfies M_i, *i.e.*, y^i has 1s in precisely the variables contained in M_i. Note that since f is individuated the points $y^1, \ldots, y^s$ are all pairwise incomparable with respect to the bitwise partial ordering. Thus we have $f(y^i) = 1$ but $f(x) = 0$ for any x such that $x < y^i$ for some i. We sometimes say that y is above x if $x \leq y$.

Let $S \subset \{0,1\}^n$ be the set which contains: (a) each y^i (which is a positive example) and all of its neighbors that can be obtained by flipping a single 1 to 0 (all of these are negative examples); and (b) the $(s-1)$ additional points $z^2 = \text{join}(y^1, y^2)$, $z^3 = \text{join}(y^1, y^2, y^3)$, $\ldots$, $z^s = \text{join}(y^1, y^2, \ldots, y^s)$ (it is not hard to see that z^i is a positive example for i odd and a negative example for i even, since z^i satisfies precisely the monomials $M_1, \ldots, M_i$ and $M_1 \oplus \cdots \oplus M_s$ is individuated). There are at most $(n+1)s$ points from (a) and $s - 1$ points from (b) so we have $|S| \leq ns + 2s - 1$.

We will show that S is a teaching set for f and thus prove the lemma. So suppose that $\hat{f} = \widehat{M}_1 \oplus \cdots \oplus \widehat{M}_r$ is some GF_2 polynomial that is consistent with S where $r \leq s$. Let us write $\hat{y}^j$ for the minimal assignment that satisfies $\widehat{M_j}$.

We first observe that since y^1 is a positive example, there must be at least one $\hat{y}^j$ such that $\hat{y}^j \leq y^1$. Since $y^1 < z^2$ and z^2 is a negative example, there must be at least two $\hat{y}^j$ such that $\hat{y}^j \leq z^2$. Since the labels of $z^2, z^3, \ldots$ always alternate, proceeding in this fashion there must be at least s many $\hat{y}^j$ such that $\hat{y}^j \leq z^s$. It follows that $r = s$, that y^1 is above precisely one $\hat{y}^1$, and that in fact each z^i is above precisely i of the $\hat{y}^j$'s (call them $\hat{y}^1, \ldots, \hat{y}^i$).

Now the negative examples below y^1 show that in fact we must have $\hat{y}^1 = y^1$. Since z^2 is above exactly one other $\hat{y}^j$ besides $\hat{y}^1$ (namely $\hat{y}^2$), and z^2 is above y^2 which is labeled positive, it must be the case that $\hat{y}^2 \leq y^2$ (for if $\hat{y}^2 \not\leq y^2$, there would be no $\hat{y}^j$ beneath y^2 and consequently y^2 would be labeled negative). But since all of y^2's downward neighbors are labeled negative, it must be the case that $\hat{y}^2 = y^2$. Similar logic applied successively to $z^3, \ldots, z^s$ shows that each of $\hat{y}^3, \ldots, \hat{y}^s$ must equal the corresponding $y^3, \ldots, y^s$. Thus we have $\widehat{M_i} = M_i$ for $i = 1, \ldots, s$, so $\hat{f} = f$ and the lemma is proved. ■

Now observe that $|\mathcal{G}_s| = \binom{2^n}{s} < \frac{2^{ns}}{s!}$, and thus $(\frac{2^n}{s})^s \leq |\mathcal{G}_{\leq s}| = |\mathcal{G}_s| + |\mathcal{G}_{\leq s-1}| < \frac{2^{ns}}{s!} + (s-1)\frac{2^{ns-n}}{(s-1)!} = \frac{2^{ns}}{s!} + \frac{2^{ns-n}}{(s-2)!}$. Our next lemma shows that almost every function in $\mathcal{G}_s$ (and thus almost every function in $\mathcal{G}_{\leq s}$) is in fact individuated:

Lemma 89. *Recall that* $1 \leq s \leq (1-\epsilon)\log_2 n$*, where* $\epsilon > 0$ *is any constant. We have* $|\mathcal{I}| \geq \frac{2^{ns}}{s!}(1 - s \cdot e^{-n^\epsilon})$*, and thus there are at most* $s \cdot e^{-n^\epsilon} \cdot \frac{2^{ns}}{s!} + \frac{2^{ns-n}}{(s-2)!}$ *many functions in* $\mathcal{G}_{\leq s} \setminus \mathcal{I}$*.*

Proof. Let $(M_1, \ldots, M_s)$ be a sequence of s monomials obtained by drawing each one uniformly from all 2^n possible monomials. We will show that of the 2^{ns} possible outcomes for $(M_1, \ldots, M_s)$, at most an $s \cdot e^{-n^\epsilon}$ fraction have the property that the corresponding GF_2 polynomial $M_1 \oplus \cdots \oplus M_s$ is not individuated, and consequently the number of sequences for which the corresponding GF_2 polynomial *is* individuated is at least $2^{ns}(1 - s \cdot e^{-n^\epsilon})$. Each such sequence clearly consists of s distinct monomials (since no sequence in which some monomial occurs more than once can be individuated), so accounting for the $s!$ different orderings of s distinct elements, we have that there are at least $2^{ns}(1 - s \cdot e^{-n^\epsilon})/s!$ many individuated GF_2 polynomials.

We say that a variable individuates a monomial M_i if it occurs in M_i but in no other M_j. For any fixed variable x_j, and fixed index $1 \le i \le s$, the probability (over the random choice of $(M_1, \dots, M_s)$) that x_j individuates M_i is precisely $1/2^s$, since x_j must occur in M_i (probability $1/2$) and must be absent from each of the other $s-1$ terms (probability $1/2^{s-1}$). By independence, the probability that none of the n variables individuates M_i is $\left(1 - \frac{1}{2^s}\right)^n \le e^{-n/2^s} \le e^{-n^\epsilon}$, where we have used the fact that $s \le (1-\epsilon)\log_2 n$. A union bound now gives that the probability that any of the s monomials $M_1, \dots, M_s$ is not individuated by any variable is at most $s \cdot e^{-n^\epsilon}$. ■

By Lemma 88 we can specify any function in $\mathcal{I}$ with at most $N := ns + 2s - 1$ examples, and by Theorem 86 we can specify any of the other functions in $\mathcal{G}_{\le s}$ with at most $n^{O(\log s)}$ many examples. It follows from Lemma 89 that the average teaching dimension of $\mathcal{G}_{\le s}$ is at most

$$\frac{N|\mathcal{I}| + n^{O(\log s)} \cdot |\mathcal{G}_{\le s} \setminus \mathcal{I}|}{|\mathcal{G}_{\le s}|} \le N + \frac{n^{O(\log s)} \cdot (s \cdot e^{-n^\epsilon} \cdot \frac{2^{ns}}{s!} + \frac{2^{ns-n}}{(s-2)!})}{(\frac{2^n}{s})^s}.$$

The second term on the right simplifies to $s^s \cdot n^{O(\log s)} \cdot (s \cdot e^{-n^\epsilon}/s! + 2^{-n}/(s-2)!)$, which is easily seen to be $o(1)$ since ϵ is a constant greater than 0 and $s \le (1-\epsilon)\log n$. This proves Theorem 87.

While our proof technique does not extend to s that are larger than $\log n$, it is possible that different methods could establish a $\text{poly}(n, s)$ upper bound on the average teaching dimension for the class $\mathcal{G}_{\le s}$ of s-sparse GF_2 polynomials for a much larger range of values of s. This is an interesting goal for future work.

Chapter 8

Conclusions and Future Directions

While we have made significant progress on understanding the learnability of succinctly representable monotone functions, the main question still remains: Are polynomial-size monotone DNF formulas learnable?

On the impossibility side, one could try to extend our results from Chapter 4 to lower bound the strong statistical query dimension of monotone DNF formulas. Unfortunately, the idea of embedding parities in the middle slice of the Boolean hypercube is unlikely to result in depth-2 monotone circuits.

On the algorithmic side, one could try to extend our algorithm from Chapter 5 to learn the entire class of monotone DNF formulas. Until a strong statistical query lower bound is shown, there is no reason to believe that monotone DNF formulas cannot be learned by Fourier analytic techniques.

As for the general question of learning succinctly representable monotone functions, an obvious goal for future work is to establish even sharper quantitative bounds on the cryptographic hardness of learning monotone functions. Blum *et al.* [BBL98] obtain their $\frac{1}{2} + \frac{\omega(\log n)}{n^{1/2}}$ information-theoretic lower bound by considering random monotone DNF formulas that are constructed by independently including each of the $\binom{n}{\log n}$ possible terms of length $\log n$ in the target function. Can we match this hardness with a class of polynomial-size circuits?

As mentioned in Section 3.1.1, it is natural to consider a pseudorandom variant of the Blum *et al.* construction in which a pseudorandom rather than truly random function is used to decide whether or not to include each of the $\binom{n}{\log n}$ candidate terms. However, we have not been able to show that a function f constructed in this way can be computed by a poly(n)-size circuit. Intuitively, the problem is that for an input x with (typically) $n/2$ bits set to 1, to evaluate f we must check the pseudorandom function's value on all of the $\binom{n/2}{\log n}$ potential "candidate terms" of length $\log n$ which x satisfies. Indeed, the question of obtaining an efficient implementation of these "huge pseudorandom monotone DNF formulas" has a similar flavor to Open Problem 5.4 of a paper by Goldreich *et al.* [GGN03]. In that question the goal is to construct pseudorandom functions that support "subcube queries" which give the parity of the function's values over all inputs in a specified subcube of $\{0,1\}^n$. In our scenario we are interested in functions f which are pseudorandom only over the $\binom{n}{\log n}$ inputs with precisely $\log n$ ones (these inputs correspond to the "candidate terms" of the monotone DNF formula) and are zero everywhere else, and we only need to support "monotone subcube queries" (*i.e.,* given an input x, we want to know whether $f(y) = 1$ for any $y \leq x$).

Showing that the candidate algorithm in Section 6.1.1 for properly learning monotone decision trees works, even only for constant accuracy, would be a huge advance. The intermediate results obtained in Chapter 6 suggest other possible avenues of research. For instance, Theorem 54 implies:

Theorem 90 ([OSSS05] Theorem 1.1)**.** *Let* $f : \{0,1\}^n \to \{+1,-1\}$*. Then,*

$$\max_{i\in[n]} \mathrm{Inf}_i(f) = \Omega\left(\frac{\mathrm{Var}[f]}{\log(\text{DT-size}(f))}\right).$$

Can we show analogous results for different complexity measures such as subcube partition-size, CDNF-size, and the ℓ_1-norm?

Bibliography

[ABF$^+$04] Michael Alekhnovich, Mark Braverman, Vitaly Feldman, Adam R. Klivans, and Toniann Pitassi. Learnability and automatizability. In *Proc. 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 621–630. IEEE Computer Society Press, October 2004.

[ABST95] M. Anthony, G. Brightwell, and J. Shawe-Taylor. On specifying Boolean functions by labelled examples. *Discrete Applied Mathematics*, 61(1):1–25, 1995.

[AHM$^+$06] Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael Saks. Minimizing DNF formulas and AC^0_d circuits given a truth table. In *Proc. 21st Annual IEEE Conference on Computational Complexity (CCC)*. IEEE Computer Society Press, 2006.

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. An $o(n \log n)$ sorting network. *Combinatorica*, 3(1):1–19, 1983.

[AM02] Kazuyuki Amano and Akira Maruoka. On learning monotone boolean functions under the uniform distribution. In *Proc. Algorithmic Learning Theory, 13th International Conference (ALT)*, volume 2533 of *Lecture Notes in Artificial Intelligence*, pages 57–68. Springer-Verlag, 2002.

[AP95] H. Aizenstein and L. Pitt. On the learnability of disjunctive normal form formulas. *Machine Learning*, 19:183–208, 1995.

[Bal05] Frank J. Balbach. Teaching classes with high teaching dimension using few examples. In Peter Auer and Ron Meir, editors, *Proc. of the 18th Annual Con-*

ference on Computational Learning Theory (COLT), volume 3559 of *Lecture Notes in Computer Science*, pages 637–651. Springer-Verlag, 2005.

[BBL98] Avrim Blum, Carl Burch, and John Langford. On learning monotone boolean functions. In *Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 408–415. IEEE Computer Society Press, 1998.

[BEHW89] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(84):929–965, 1989.

[Ber82] S. J. Berkowitz. On some relationships between monotone and non-monotone circuit complexity. Technical report, University of Toronto, 1982.

[BF02] Nader H. Bshouty and Vitaly Feldman. On using extended statistical queries to avoid membership queries. *Journal of Machine Learning Research*, 2:359–395, 2002. Prelim. ver. in *Proc. of COLT'01*.

[BFJ$^+$94] Avrim Blum, Merrick L. Furst, Jeffrey Jackson, Michael J. Kearns, Yishay Mansour, and Steven Rudich. Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In *Proc. 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 253–262. ACM Press, 1994.

[BFKL93] Avrim Blum, Merrick Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *Advances in Cryptology (CRYPTO)*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291. Springer-Verlag, 1993.

[BFKV98] Avrim Blum, Alan M. Frieze, Ravi Kannan, and Santosh Vempala. A polynomial-time algorithm for learning noisy linear threshold functions. *Algorithmica*, 22(1/2):35–52, 1998.

[BFSO84] Leo Breiman, Jerome Friedman, Charles J. Stone, and RA Olshen. *Classification and Regression Trees.* Wadsworth and Brooks, 1984.

[BKS99] Itai Benjamini, Gil Kalai, and Oded Schramm. Noise sensitivity of boolean functions and applications to percolation. *Publications Mathématiques de l'I.H.E.S.*, 90:5–43, 1999.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM*, 50(4):506–519, July 2003. Prelim. ver. in *Proc. of STOC'00*.

[BLR08] Avrim Blum, Katrina Ligett, and Aaron Roth. A learning theory approach to non-interactive database privacy. In *Proc. 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 609–618. ACM Press, 2008.

[Blu03] Avrim Blum. Learning a function of r relevant variables. In *Proc. of the 16th Annual Conference on Computational Learning Theory (COLT)*, volume 2777 of *Lecture Notes in Computer Science*, pages 731–733. Springer-Verlag, 2003.

[BM02] Nader H. Bshouty and Yishay Mansour. Simple learning algorithms for decision trees and multivariate polynomials. *SIAM Journal on Computing*, 31(6):1909–1925, 2002.

[BMOS03] Nader H. Bshouty, Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning DNF from random walks. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 189–198. IEEE Computer Society Press, October 2003.

[BOL87] Michael Ben-Or and Nathan Linial. Collective coin flipping. Technical report, Hebrew University of Jerusalem, 1987. Prelim. ver. in *Proc. of FOCS'85*.

[BS88] N. Bshouty and G. Seroussi. Vector sets for exhaustive testing of logic circuits. *IEEE Transactions on Information Theory*, 34(3):513–522, 1988.

[BT06] Nader H. Bshouty and Christino Tamon. On the Fourier spectrum of monotone functions. *Journal of the ACM*, 43(4):747–770, 2006.

[Byl94] Tom Bylander. Learning linear threshold functions in the presence of classification noise. In *Proc. of the 7th Annual Conference on Computational Learning Theory (COLT)*, pages 340–347, 1994.

[Che52] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–509, 1952.

[CS98] J. Cherniavsky and R. Statman. Testing: An abstract approach. In *Proceedings of the 2nd Workshop on Software Testing*, 1998.

[Dec93] Scott E. Decatur. Statistical queries and faulty PAC oracles. In *Proc. of the 6th Annual Conference on Computational Learning Theory (COLT)*, pages 262–268. ACM Press, 1993.

[DSLM$^+$08] Dana Dachman-Soled, Homin K. Lee, Tal Malkin, Rocco A. Servedio, Andrew Wan, and Hoeteck Wee. Optimal cryptographic hardness of learning monotone functions. In *Proc. 35th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 36–47. Springer-Verlag, 2008.

[DV04] John Dunagan and Santosh Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. In *Proc. 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 315–320. ACM Press, 2004.

[EH89] Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82:231–246, 1989.

[EHKV89] Andrzej Ehrenfeucht, David Haussler, Michael J. Kearns, and Leslie Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82:246–261, 1989.

[Fel08] Vitaly Feldman. Evolvability from learning algorithms. In *Proc. 40th Annual ACM Symposium on Theory of Computing (STOC)*. ACM Press, 2008.

[Fel09] Vitaly Feldman. A complete characterization of statistical query learning with applications to evolvability. Manuscript, 2009.

[FGKP07] Vitaly Feldman, Parikshit Gopalan, Subhash Khot, and Ashok Kumar Ponnuswami. On agnostic learning of parities, monomials and halfspaces. *SIAM Journal on Computing*, 2007. Prelim. ver. in *Proc. of FOCS'06 & CCC'06*.

[FKN02] Ehud Friedgut, Gil Kalai, and Assaf Naor. Boolean functions whose fourier transform is concentrated on the first two levels. *Advances in Applied Mathematics*, 29:427–437, 2002.

[Fre95] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–185, 1995. Prelim. ver. in *COLT'90*.

[FS92] Paul Fischer and Hans-Ulrich Simon. On learning ring-sum-expansions. *SIAM Journal on Computing*, 21(1):181–192, 1992. Prelim. ver. in *COLT'90*.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986. Prelim. ver. in *FOCS'84*.

[GGN03] Oded Goldreich, Shafi Goldwasser, and Asaf Nussboim. On the implementation of huge random objects. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 68–. IEEE Computer Society Press, 2003.

[GK92] Sally A. Goldman and Michael J. Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50(1):20–31, February 1992.

[Gol05] Oded Goldreich. Foundations of cryptography – a primer. *Foundations and Trends in Theoretical Computer Science*, 1(1):1–116, 2005.

[GRS93] S. Goldman, R. Rivest, and R. Schapire. Learning binary relations and total orders. *SIAM Journal on Computing*, 22(5):1006–1034, 1993.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4), 1999.

[HM91] Thomas Hancock and Yishay Mansour. Learning monotone k-μ DNF formulas on product distributions. In *Proc. of the 4th Annual Conference on Computational Learning Theory (COLT)*, pages 179–183, 1991.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, 58:13–30, 1963.

[HSW92] David Helmbold, Robert Sloan, and Manfred K. Warmuth. Learning integer lattices. *SIAM Journal on Computing*, 21(2):240–266, 1992. Prelim. ver. in *COLT'90*.

[HVV04] Alexander Healy, Salil Vadhan, and Emanuele Viola. Using nondeterminism to amplify hardness. *SIAM Journal on Computing*, 35(4):903–931, 2004. Prelim. ver. in *STOC'04*.

[Hyl04] K. Hyland. Graduates' gratitude: the generic structure of dissertation acknowledgements. *English for Specific Purposes*, 23:303–324, 2004.

[Jac97] Jeffrey C. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *Journal of Computer and System Sciences*, 55(3):414–440, 1997. Prelim. ver. in *Proc. of FOCS'94*.

[JKS02] Jeffrey C. Jackson, Adam Klivans, and Rocco A. Servedio. Learnability beyond ac^0. In *Proc. 34th Annual ACM Symposium on Theory of Computing (STOC)*. ACM Press, 2002.

[JLSW08] Jeffrey C. Jackson, Homin K. Lee, Rocco A. Servedio, and Andrew Wan. Learning random monotone DNF. In *11th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 12th International Workshop on Randomization and Computation (RANDOM-APPROX)*, pages 483–497. Springer-Verlag, 2008.

[JS05a] Jeffrey C. Jackson and Rocco A. Servedio. Learning random log-depth decision trees under uniform distribution. *SIAM Journal on Computing*, 34(5):1107–1128, 2005. Prelim. ver. in *Proc. of COLT'03*.

[JS05b] Jeffrey C. Jackson and Rocco A. Servedio. On learning random DNF formulas under the uniform distribution. In *8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 9th International Workshop on Randomization and Computation (RANDOM-APPROX)*, volume 3624 of *Lecture Notes in Computer Science*, pages 342–353. Springer-Verlag, 2005.

[JT97] Jeffrey C. Jackson and C. Tamon. Fourier analysis in machine learning. ICML/COLT 1997 tutorial slides, available at: http://learningtheory.org/resources.html, 1997.

[Kea98] Michael J. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45(6):983–1006, 1998. Prelim. ver. in *Proc. of STOC'93*.

[Kha93] Michael Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 372–381. ACM Press, 1993.

[Kha95] Michael Kharitonov. Cryptographic lower bounds for learnability of Boolean functions on the uniform distribution. *Journal of Computer and System Sciences*, 50(3):600–610, June 1995.

[KKL88] Jeff Kahn, Gil Kalai, and Nathan Linial. The influence of variables on Boolean functions. In *Proc. 29th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 68–80. IEEE Computer Society Press, 1988.

[KLN$^+$08] Shiva Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? In *Proc. 49th IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, 2008. To appear.

[KLRS96] Eyal Kushilevitz, Nathan Linial, Yuri Rabinovich, and Michael Saks. Witness sets for families of binary vectors. *Journal of Combinatorial Theory*, 73(2):376–380, February 1996.

[KLV94] Michael J. Kearns, Ming Li, and Leslie G. Valiant. Learning Boolean formulas. *Journal of the ACM*, 41(6):1298–1328, 1994. Prelim. ver. in *Proc. of STOC'87*.

[KM93] Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, December 1993. Prelim. ver. in *Proc. of STOC'91*.

[KM96] Michael J. Kearns and Yishay Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proc. 28th Annual ACM Symposium on Theory of Computing (STOC)*. ACM Press, 1996.

[KMSP94] Ludek Kucera, Alberto Marchetti-Spaccamela, and Marco Protasi. On learning monotone DNF formulae under uniform distributions. *Information and Computation*, 110(1):84–95, 1994.

[KOS04] Adam Klivans, Ryan O'Donnell, and Rocco A. Servedio. Learning intersections and thresholds of halfspaces. *Journal of Computer and System Sciences*, 68(4):808–840, 2004. Prelim. ver. in *Proc. of FOCS'02*.

[KPPY84] Maria M. Klawe, Wolfgang J. Paul, Nicholas Pippenger, and Mihalis Yannakakis. On monotone formulae with restricted depth. In *Proc. 16th Annual ACM Symposium on Theory of Computing (STOC)*, pages 480–487. ACM Press, 1984.

[KS06] Adam R. Klivans and Alexander A. Sherstov. Unconditional lower bounds for learning intersections of halfspaces. In *Proc. of the 19th Annual Conference on Computational Learning Theory (COLT)*, 2006.

[Kuh99] Christian Kuhlmann. On teaching and learning intersection-closed concept classes. In *Proc. 4th EUROCOLT*, pages 168–182, 1999.

[KV94] M. Kearns and U. Vazirani. *An introduction to computational learning theory.* MIT Press, Cambridge, MA, 1994.

[LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform, and learnability. *Journal of the ACM*, 40(3):607–620, 1993. Prelim. ver. in *Proc. of FOCS'89*.

[LSW07] Homin K. Lee, Rocco A. Servedio, and Andrew Wan. DNF are teachable in the average case. *Machine Learning*, 69(2-3):79–96, 2007. Prelim. ver. in *Proc. of COLT'06*.

[Man94] Yishay Mansour. Learning Boolean functions via the Fourier transform. In *Theoretical Advances in Neural Computation and Learning*, chapter 11, pages 391–424. Kluwer Academic Publishers, 1994.

[MO03] Elchanan Mossel and Ryan O'Donnell. On the noise sensitivity of monotone functions. *Random Structures and Algorithms*, 23(3):333–350, 2003.

[MOS03] Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning juntas. In *Proc. 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 206–212. ACM Press, 2003.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, New York, NY, 1995.

[Nep70] V. Nepomnjaščiĭ. Rudimentary predicates and turing calculations. *Soviet Mathematics Doklady*, 11:1462–1465, 1970.

[NR04] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM*, 51(2):231–262, March 2004.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994. Prelim. ver. in *Proc. of FOCS'98*.

[O'D04] Ryan O'Donnell. Hardness amplification within NP. *Journal of Computer and System Sciences*, 69(1):68–94, 2004. Prelim. ver. in *Proc. of STOC'02*.

[OS07] Ryan O'Donnell and Rocco A. Servedio. Learning monotone decision trees in polynomial time. *SIAM Journal on Computing*, 37(3):827–844, 2007. Prelim. ver. in *Proc. of CCC'06*.

[OSSS05] Ryan O'Donnell, Michael Saks, Oded Schramm, and Rocco A. Servedio. Every decision tree has an influential variable. In *Proc. 46th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 31–39. IEEE Computer Society Press, 2005.

[OW09] Ryan O'Donnell and Karl Wimmer. KKL, Kruskal-Katona, and monotone nets. Manuscript, 2009.

[Qui93] J. Ross Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., 1993.

[Raz85] Alexander A. Razborov. Lower bounds on the monotone network complexity of the logical permanent. *Matematicheskie Zametki*, 37:887–900, 1985.

[RB91] R. Roth and G. Benedek. Interpolation and approximation of sparse multivariate polynomials over $GF(2)$. *SIAM Journal on Computing*, 20(2):291–314, 1991.

[Sch90] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990. Prelim. ver. in *Proc. of FOCS'1989*.

[Sch01] Robert E. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2001.

[Sel08] Linda Sellie. Learning random monoton DNF under the uniform distribution. In *Proc. of the 21th Annual Conference on Computational Learning Theory (COLT)*, pages 181–192, 2008.

[Sel09] Linda Sellie. Exact learning of random dnf over the uniform distribution. In *Proc. 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 45–54, 2009.

[Ser04] Rocco A. Servedio. On learning monotone DNF under product distributions. *Information and Computation*, 193:57–74, 2004. Prelim. ver. in *Proc. of COLT'01*.

[She07] A. Sherstov. Halfspace matrices. In *Proc. 22nd Annual IEEE Conference on Computational Complexity (CCC)*, pages 83–95. IEEE Computer Society Press, 2007.

[Sim07] Hans Ulrich Simon. A characterization of strong learnability in the statistical query model. In *Proc. 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 393–404, 2007.

[SM90] A. Shinohara and S. Miyano. Teachability in computational learning. In *Proc. Algorithmic Learning Theory, 1st International Workshop (ALT)*, pages 247–255, 1990.

[SM00] Y. Sakai and A. Maruoka. Learning monotone log-term DNF formulas under the uniform distribution. *Theory of Computing Systems*, 33:17–33, 2000.

[SS96] Robert E. Schapire and Linda M. Sellie. Learning sparse multivariate polynomials over a field with queries and counterexamples. *Journal of Computer and System Sciences*, 52:201–213, 1996.

[Tre03] Luca Trevisan. List decoding using the XOR lemma. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 126–135. IEEE Computer Society Press, 2003.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984. Prelim. ver. in *Proc. of STOC'84*.

[Ver90] Karsten A. Verbeurgt. Learning DNF under the uniform distribution in quasi-polynomial time. In *Proc. of the 3rd Annual Conference on Computational Learning Theory (COLT)*, pages 314–326, 1990.

[Ver98] Karsten A. Verbeurgt. Learning sub-classes of monotone DNF on the uniform distribution. In *Proc. Algorithmic Learning Theory, 9th Annual Conference (ALT)*, pages 385–399. spver, 1998.

[Yan01] Ke Yang. On learning correlated Boolean functions using statistical query. Technical Report 98, Electronic Colloquium on Computational Complexity (ECCC), 2001. Prelim. ver. in *Proc. of ALT'01*.

[Yan05] Ke Yang. New lower bounds for statistical query learning. *Journal of Computer and System Sciences*, 70(4):485–509, 2005. Prelim. ver. in *Proc. of COLT'02*.