THEORY OF COMPUTING, Volume 5 (2009), pp. 257–282
www.theoryofcomputing.org

Optimal Cryptographic Hardness of Learning Monotone Functions*

Dana Dachman-Soled ‡ † Homin K. Lee ‡ Tal Malkin §
Rocco A. Servedio ¶ Andrew Wan ‡ Hoeteck Wee

Received: October 9, 2008; published: December 12, 2009.

Abstract: 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. Prior to our work, however, the only negative result for learning monotone functions in this model has been an information-theoretic lower bound showing that certain super-polynomial-size monotone circuits cannot be learned to accuracy $1/2+\omega(\log n/\sqrt{n})$ (Blum, Burch, and Langford, *FOCS'98*). This is in contrast with the situation for non-monotone functions, where a wide range of cryptographic hardness results establish that various "simple" classes of polynomial-size circuits are not learnable by polynomial-time algorithms.

In this paper we establish cryptographic hardness results for learning various "simple" classes of monotone circuits, thus giving a computational analogue of the information-

*A extended abstract of this paper appeared in the Proceedings of ICALP 2008 [10]. The results and proofs in Section 4 are new to this article.

†Supported in part by an FFSEAS Presidential Fellowship.

‡Supported in part by NSF Cybertrust CNS-0716245.

§Supported by NSF CAREER award CCF-03-047839 and NSF award SBE-0245014.

¶Supported by DARPA grant HR0011-07-1-0012, NSF CAREER award CCF-0347282, NSF award CCF-0523664, and a Sloan Foundation Fellowship.

ACM Classification: F.1.3

AMS Classification: 68Q17

Key words and phrases: computational learning theory, hardness amplification, lower bounds, cryptography, analysis of Boolean functions

© 2009 Dana Dachman-Soled, Homin K. Lee, Tal Malkin, Rocco A. Servedio, Andrew Wan, and Hoeteck Wee
Licensed under a Creative Commons Attribution License DOI: 10.4086/toc.2009.v005a013

theoretic hardness results of Blum et al. mentioned above. Some of our results show the cryptographic hardness of learning polynomial-size monotone circuits to accuracy only slightly greater than $1/2+1/\sqrt{n}$, which is close to the optimal accuracy bound by positive results of Blum et al. Other results show that under a plausible cryptographic hardness assumption, a class of constant-depth, sub-polynomial-size circuits computing monotone functions is hard to learn. This result is close to optimal in terms of the circuit-size parameter by known positive results as well (Servedio, *Information and Computation* 2004). Our main tool is a complexity-theoretic approach to hardness amplification via noise sensitivity of monotone functions that was pioneered by O'Donnell (*JCSS* 2004).

1 Introduction

More than two decades ago Valiant introduced the Probably Approximately Correct (PAC) model of learning Boolean functions from random examples [34]. 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. This paper addresses a discrepancy between known positive and negative results for uniform distribution learning by establishing strong computational hardness results for learning various classes of *monotone* functions.

1.1 Background and motivation

In the uniform distribution PAC learning model, a learning algorithm is given access to a source of independent random examples $(x, f(x))$ where each x is drawn uniformly from the n-dimensional Boolean cube and f is the unknown Boolean function to be learned. The goal of the learner is to construct a high-accuracy hypothesis function h, i. e., one that satisfies $\Pr[f(x) \neq h(x)] \leq \varepsilon$, where the probability is with respect to the uniform distribution and ε is an error parameter given to the learning algorithm. Algorithms and hardness results in this framework have interesting connections with topics such as discrete Fourier analysis [23], circuit complexity [22], noise sensitivity and influence of variables in Boolean functions [16, 4, 21, 29], coding theory [11], privacy [8, 17], and cryptography [7, 20]. 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.

Monotonicity makes learning easier For many classes of functions, uniform distribution learning algorithms have been devised that substantially improve on a naive exponential-time approach to learning via brute-force search. However, despite intensive efforts, researchers have not yet obtained poly(n)-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 poly(n)-time—algorithms. We list some examples:

1. A simple algorithm learns monotone $O(\log n)$-juntas[1] to perfect accuracy in poly(n) time, and a more complex algorithm [9] learns monotone $\tilde{O}(\log^2(n))$-juntas to any constant accuracy in

[1] A k-junta $f(x_1,\ldots,x_n)$ is a Boolean function that only depends on k of the n input variables.

poly(n) time. In contrast, the fastest known algorithm for learning arbitrary k-juntas runs in time $n^{.704k}$ [25].

2. The fastest known uniform distribution learning algorithm for the general class of s-term DNF, due to Verbeurgt [36], runs in time $n^{O(\log s)}$ to learn to any constant accuracy. In contrast, [31] gives an algorithm that runs in time $s^{O(\log s)}$ for s-term monotone DNF. Thus, the class of $2^{O(\sqrt{\log n})}$-term monotone DNF can be learned to any constant accuracy in poly(n) time, but no such result is known for $2^{O(\sqrt{\log n})}$-term general DNF.

3. The fastest known algorithm for learning poly(n)-size general decision trees to constant accuracy takes $n^{O(\log n)}$ time (following from [36]), but poly(n)-size decision trees that compute monotone functions can be learned to any constant accuracy in poly(n) time [29].

4. No poly(n)-time algorithm can learn the general class of all Boolean functions on $\{0,1\}^n$ to accuracy better than $1/2+\text{poly}(n)/2^n$, but a simple polynomial-time algorithm can learn the class of all monotone Boolean functions to accuracy $1/2+\Omega(1/\sqrt{n})$ [6]. We note also that the result of [9] mentioned above follows from a $2^{\tilde{O}(\sqrt{n})}$-time algorithm for learning arbitrary monotone functions on n variables to constant accuracy. (It is easy to see that no comparable algorithm can exist for learning arbitrary Boolean functions to constant accuracy.)

Cryptography and hardness of learning 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 [20] showed that the class AC^1 of logarithmic-depth, polynomial-size Boolean circuits (circuits with AND, OR, and NOT gates) is hard to learn under the uniform distribution. Subsequently Kharitonov [19] showed that if factoring Blum integers is $2^{n^{\varepsilon}}$-hard for some fixed $\varepsilon > 0$, then even the class AC^0 of constant-depth, polynomial-size Boolean circuits similarly cannot be learned in polynomial time under the uniform distribution. In later work, Naor and Reingold [26] 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. (TC^0 circuits are Boolean circuits that can also use MAJ gates. The value of a MAJ gate is one if at least half of its inputs are one, and zero otherwise.) We note that all of these hardness results apply even to algorithms that 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 above, a fairly broad range of polynomial-time learning results have 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? As far as we are aware, 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 paper (which, as we describe below, lets us answer "no" to the above question in a strong sense).

1.2 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 [6] showed that arbitrary monotone functions cannot be learned to accuracy better than $1/2 + O(\log n/\sqrt{n})$ by any algorithm that makes $\mathrm{poly}(n)$ many membership queries. This is an information-theoretic bound that is proved using randomly generated monotone DNF formulas of size (roughly) $n^{\log n}$ that are not polynomial-time computable. A natural goal is to obtain *computational* lower bounds for learning polynomial-time computable monotone functions that match, or nearly match, this level of hardness (which is close to optimal by the $(1/2 + \Omega(1/\sqrt{n}))$-accuracy algorithm of Blum et al. described above). We prove near-optimal hardness for learning polynomial-size monotone Boolean circuits (circuits with AND and OR gates):

Theorem 1.1 (informal statement)**.** *If one-way functions exist, then there is a class of* $\mathrm{poly}(n)$*-size monotone Boolean circuits that cannot be learned to accuracy* $1/2 + 1/n^{1/2-\varepsilon}$ *for any fixed* $\varepsilon > 0$.

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 $1/2 + \mathrm{polylog}(n)/\sqrt{n}$.
- Assuming the hardness of factoring Blum integers, our hard-to-learn functions may be computed in monotone NC^1.
- Assuming that Blum integers are $2^{n^{\varepsilon}}$-hard to factor on average (which is the same hardness assumption used in Kharitonov's work [19]), we obtain a lower bound for learning constant-depth circuits of sub-polynomial size that almost matches the positive result from [31]. More precisely, we show that for any (sufficiently large) constant d, the class of monotone functions computed by depth-d Boolean circuits of size $2^{(\log n)^{O(1)/(d+1)}}$ cannot be learned to accuracy 51% under the uniform distribution in $\mathrm{poly}(n)$ time. In contrast, the positive result of [31] shows that monotone functions computed by depth-d Boolean circuits of size $2^{O((\log n)^{1/(d+1)})}$ can be learned to any constant accuracy in $\mathrm{poly}(n)$ time.

These results are summarized in Figure 1.

Proof techniques A natural first approach is to try to replace the random $n^{\log n}$-term monotone DNFs constructed in [6] by pseudorandom DNFs of polynomial size. We were not able to do this directly;

Hardness assumption	Complexity of f	Accuracy bound	Ref.
none	random $n^{\log n}$-term monotone DNF	$\frac{1}{2}+\frac{\omega(\log n)}{n^{1/2}}$	[6]
OWF (poly)	poly-size monotone circuits	$\frac{1}{2}+\frac{1}{n^{1/2-\varepsilon}}$	Thm. 1.1
OWF ($2^{n^{\alpha}}$)	poly-size monotone circuits	$\frac{1}{2}+\frac{\mathrm{poly}(\log n)}{n^{1/2}}$	Thm. 2.9
factoring BI (poly)	monotone NC^1-circuits	$\frac{1}{2}+\frac{1}{n^{1/2-\varepsilon}}$	Thm. 3.2
factoring BI ($2^{n^{\alpha}}$)	depth-d, size $2^{(\log n)^{O(1)/(d+1)}}$ Boolean circuits	$\frac{1}{2}+o(1)$	Thm. 3.5

Figure 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 that no $\mathrm{poly}(n)$-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 "$2^{n^{\alpha}}$" mean that the problems are intractable for $\mathrm{poly}(n)$- and $2^{n^{\alpha}}$-time algorithms, respectively (for some fixed $\alpha > 0$). Recall that the $\mathrm{poly}(n)$-time algorithm of [6] for learning monotone functions implies that the best possible accuracy bound for monotone functions is $1/2+\Omega(1)/n^{1/2}$.

indeed, as we discuss in Section 5, constructing such DNFs seems closely related to an open problem of Goldreich, Goldwasser, and Nussboim [13]. However, it turns out that a closely related approach does yield some results along the desired lines; in Section 4 we present and analyze a simple variant of the information-theoretic construction from [6] and then show how to replace random choice by pseudorandom in this the variant. Because our variant gives a weaker quantitative bound on the information-theoretic hardness of learning than [6], this gives a construction of polynomial-time-computable monotone functions that, assuming the existence of one-way functions, cannot be learned to accuracy $1/2+1/\mathrm{polylog}(n)$ under the uniform distribution. While this answers the question posed above (even with "51%" in place of "99%"), the $1/\mathrm{polylog}(n)$ factor is rather far from the $O(\log n/\sqrt{n})$ factor that one might hope for as described above.

In Section 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 $1-\Omega(1/\sqrt{n})$. We then use the elegant machinery of hardness amplification of monotone functions pioneered by O'Donnell [28] to amplify the hardness of this construction to near-optimal levels (as summarized in rows 2–4 of Figure 1). We obtain our result for constant-depth, sub-polynomial-size circuits (row 5 of Figure 1) by augmenting this approach with an argument that, at a high level, is similar to one used in [2], by "scaling down" and modifying our hard-to-learn functions using a variant of Nepomnjaščiĭ's theorem [27].

1.3 Preliminaries

We consider Boolean functions of the form $f : \{0,1\}^n \to \{0,1\}$. We view $\{0,1\}^n$ as endowed with the natural partial order: $x \leq y$ if and only if $x_i \leq y_i$ for all $i = 1, \ldots, n$. A Boolean function f is *monotone* if $x \leq y$ implies $f(x) \leq f(y)$.

Our work uses various standard definitions from the fields of circuit complexity, learning, and cryptographic pseudorandomness; and for completeness we recall this material below.

Learning As described earlier, all of our hardness results apply even to learning algorithms that may make *membership queries*, i. e., black-box queries to an oracle that 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, as 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 $\mathrm{poly}(n)$ time, so the learning algorithm L may be viewed as a probabilistic polynomial-time oracle machine that 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 $\mathrm{poly}(n)$-time learning algorithm L is close to $1/2$ when run with f as the target function. Thus we bound the quantity

$$\Pr_{f\in\mathcal{C}, x\in\{0,1\}^n}\left[L^f(1^n) \to h; h(x) = f(x)\right] \tag{1.1}$$

where the probability is also taken over any internal randomization of the learning algorithm L. We say that a *class* $\mathcal{C}$ *is hard to learn to accuracy* $1/2 + \varepsilon(n)$ if, for every $\mathrm{poly}(n)$-time membership query learning algorithm L (i. e., probabilistic polynomial-time oracle algorithm), we have that the above quantity (1.1) is smaller than $1/2 + \varepsilon(n)$ for all sufficiently large n. As noted in [6], 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 We shall consider various classes of circuits computing Boolean functions, including the classes NC^1 (polynomial-size, logarithmic-depth, bounded fan-in Boolean circuits, AC^0 (polynomial-size, constant-depth, unbounded fan-in Boolean circuits), and TC^0 (polynomial-size, constant-depth unbounded fan-in Boolean circuits with MAJ gates).

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 [30] shows that there are natural monotone Boolean functions (such as the perfect matching function) that can be computed by polynomial-size circuits but cannot be computed by quasi-polynomial-size monotone circuits, and Tardos [32] observed that the separation can be increased to an exponential gap.

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 [12] are the main cryptographic primitive that underlie our constructions. Fix $k(n) \le n$, and let $\mathcal{G}$ be a family of functions $\{g : \{0,1\}^{k(n)} \to \{0,1\}\}$ 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

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

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 [12, 14]. We shall use the following folklore quantitative variant that relates the hardness of the one-way function to the security of the resulting pseudorandom function:

Proposition 1.2. *Fix $t(n) \ge \mathrm{poly}(n)$ 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 $k(n) \le n$, there is a pseudorandom family $\mathcal{G}$ of functions $\{g : \{0,1\}^{k(n)} \to \{0,1\}\}$ that is $(t(k(n)))^c$-secure.*

2 Lower bounds via hardness amplification of monotone functions

In this section we prove our main hardness results, summarized in Figure 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 [28, 33, 15], we study constructions of the form

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

where C is a Boolean "combining function" with low noise stability (we give precise definitions later) that 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 [28]).

Roughly speaking, we establish an analogue of O'Donnell's result for learning. Our analogue, given in Section 2.2, essentially states that for certain well-structured[2] 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 $1/2$. As 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

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

are easily obtained by a slight extension of an observation of Kearns, Li, and Valiant [18]. They noted that the middle slice f' of a random Boolean function on $\{0,1\}^k$ is hard to learn to accuracy greater than $1-\Theta(1/\sqrt{k})$ [6, 18]; by taking the middle slice of a *pseudorandom* function instead, we obtain an f' with the desired properties. In fact, by a result of Berkowitz [5] (see also [35, 3]), 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

In Section 2.2 we adapt the analysis from [28, 33, 15] to reduce the problem of constructing hard-to-learn monotone Boolean functions to constructing monotone combining functions C with low noise stability. In Section 2.3 we show how constructions and analyses in [28, 24] can be used to obtain a "simple" monotone combining function with low noise stability. In Section 2.4 we establish Theorems 2.8 and 2.9 (lines 2 and 3 of Figure 1) by making different assumptions about the hardness of the initial pseudorandom functions. Finally, in Section 3 we establish Theorems 3.2 and 3.5 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.

2.1 Preliminaries

Functions 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

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

For $g:\{0,1\}^k \to \{0,1\}$, we write $\mathsf{slice}(g)$ to denote the "middle slice" function:

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

where $|x|$ denotes the number of ones in the string x. It is immediate that $\mathsf{slice}(g)$ is a monotone Boolean function for any g.

Bias and noise stability Following the analysis in [28, 33, 15], we shall study the bias and noise stability of various Boolean functions. Specifically, we adopt the following notations and definitions from [15]. The *bias* of a 0-1 random variable X is defined to be

$$\mathrm{Bias}[X] \stackrel{\text{def}}{=} \left|\Pr[X=0]-\Pr[X=1]\right|.$$

Recall that a probabilistic Boolean function h on $\{0,1\}^k$ is a probability distribution over Boolean functions on $\{0,1\}^k$ (so for each input x, the output $h(x)$ is a 0-1 random variable). The *expected bias* of a probabilistic Boolean function h is

$$\mathrm{ExpBias}[h] \stackrel{\text{def}}{=} \mathrm{E}_x[\mathrm{Bias}[h(x)]].$$

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

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

where $x \in \{0,1\}^m$ is uniform random, $\eta \in \{0,1\}^m$ is a vector whose bits are each independently 1 with probability δ, and $\oplus$ denotes bitwise XOR.

2.2 Hardness amplification for learning

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

Lemma 2.1. *Let $C : \{0,1\}^m \to \{0,1\}$ 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*

$$\mathcal{C} = \{f = C \circ \mathsf{slice}(g)^{\otimes m} \mid g \in \mathcal{G}'\}$$

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 2.3, which is an analogue of Lemma 2.1 with pseudorandom rather than truly random functions, and which we use to obtain our main hardness of learning results.

Proof of Lemma 2.1. Let k, m be such that $mk = n$, and let $C : \{0,1\}^m \to \{0,1\}$ be a Boolean combining function. We prove the lemma by establishing an upper bound on the probability

$$\Pr_{g \in \mathcal{G}', x \in \{0,1\}^n}\Big[L^f(1^n) \to h;\ h(x) = f(x)\Big] \tag{2.1}$$

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

We first observe that because C is computed by a uniform family of circuits of size $\mathrm{poly}(m) \le \mathrm{poly}(n)$, 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, the probability in (2.1) is at most

$$\Pr_{g \in \mathcal{G}',\, x \in \{0,1\}^n}\Big[L^g(1^n) \to h;\ h(x) = (C \circ \mathsf{slice}(g)^{\otimes m})(x)\Big]\,. \tag{2.2}$$

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 the queries of L 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 $x=(x_1,\ldots,x_m)$.) As L runs in $\mathrm{poly}(n)$ 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 $m\cdot\mathrm{poly}(n)/2^k$. Hence (2.2) is bounded by

$$\Pr_{g,g'\in\mathcal{G}',\,x\in\{0,1\}^n}\Big[L^g(1^n)\to h;\ h(x)=(C\circ\mathsf{slice}(g')^{\otimes m})(x)\Big]+\frac{m\cdot\mathrm{poly}(n)}{2^k}\,. \tag{2.3}$$

The first summand in (2.3) is the probability that L correctly predicts the value $C\circ\mathsf{slice}(g')^{\otimes m}(x)$, 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 that, for any fixed input x, outputs 1 if and only if the random variable $(C\circ\mathsf{slice}(g')^{\otimes m})(x)$ (which we emphasize has randomness over the choice of g') is biased towards 1. The expected accuracy of this h is precisely

$$\frac{1}{2}+\frac{1}{2}\mathrm{ExpBias}[C\circ\mathsf{slice}(g')^{\otimes m}]\,. \tag{2.4}$$

Now fix

$$\delta\stackrel{\text{def}}{=}\frac{1}{2^k}\binom{k}{\lfloor k/2\rfloor}=\Theta\left(\frac{1}{\sqrt{k}}\right)$$

to be the fraction of inputs in the "middle slice" of $\{0,1\}^k$. We observe that the probabilistic function $\mathsf{slice}(g')$ (where g' is truly random) is "δ-random" in the sense of Definition 3.1 of [15], 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 [15], see also [28]):

Lemma 2.2. *Let $h:\{0,1\}^n\to\{0,1\}$ be a function that is δ-random. Then*

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

Applying this lemma to the function $\mathsf{slice}(g')$ we obtain

$$\mathrm{ExpBias}[C\circ\mathsf{slice}(g')^{\otimes m}]\le\sqrt{\mathrm{NoiseStab}_\delta[C]}\,. \tag{2.5}$$

Combining (2.3), (2.4) and (2.5) and recalling that $k=\omega(\log n)$, we obtain Lemma 2.1. □

Corollary 2.3. *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 $\mathrm{poly}(n)$-time adversaries, where $n=mk$ and $k=\omega(\log n)$. Then the class*

$$\mathcal{C}=\left\{f=C\circ\mathsf{slice}(g)^{\otimes m}\mid g\in\mathcal{G}\right\}$$

of Boolean functions over $\{0,1\}^n$ is hard to learn to accuracy

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

Proof. The corollary follows from the fact that (2.2) must differ from its pseudorandom counterpart,

$$\Pr_{g\in\mathcal{G},\, x\in\{0,1\}^n}\left[L^g(1^n)\to h;\ h(x)=(C\circ\mathsf{slice}(g)^{\otimes m})(x)\right], \tag{2.6}$$

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

$$h(x)=(C\circ\mathsf{slice}(g)^{\otimes m})(x)$$

for a random x to determine whether g is drawn from $\mathcal{G}$ or $\mathcal{G}'$. □

By instantiating Corollary 2.3 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.

2.3 A simple monotone combining function with low noise stability

In this section we combine known results of [28, 24] to obtain:

Proposition 2.4. *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*

$$\operatorname{NoiseStab}_{\Theta(1/\sqrt{k})}[C]=O\left(\frac{k^6\log m}{m}\right). \tag{2.7}$$

Note that in this proposition we may have m as large as $2^{\Theta\left(k^{.35}\right)}$ but not larger. O'Donnell [28] established the lower bound

$$\operatorname{NoiseStab}_{\Theta\left(1/\sqrt{k}\right)}[C]=\Omega\left(\frac{\log^2 m}{m}\right)$$

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 [28, 15], we use the "recursive majority of 3" function and the "tribes" function in our construction. We require the following results on noise stability:

Lemma 2.5 ([28])**.** *Let* $\text{Rec-Maj-3}_\ell:\{0,1\}^{3^\ell}\to\{0,1\}$ *be defined as follows: for $x=(x^1,x^2,x^3)$ where each $x^i\in\{0,1\}^{3^{\ell-1}}$,*

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

Then for $\ell\ge\log_{1.1}(1/\delta)$, we have

$$\operatorname{NoiseStab}_\delta[\text{Rec-Maj-3}_\ell]\le\delta^{-1.1}(3^\ell)^{-.15}.$$

Lemma 2.6 ([24]). *Let* $\mathrm{Tribes}_d : \{0,1\}^{d2^d} \to \{0,1\}$ *denote the "tribes" function on* $d2^d$ *variables, i. e., the read-once* 2^d*-term monotone* d*-DNF*

$$\mathrm{Tribes}_d(x_1,\ldots,x_{d2^d}) \stackrel{def}{=} (x_1 \wedge \cdots \wedge x_d) \vee (x_{d+1} \wedge \cdots \wedge x_{2d}) \vee \cdots .$$

Then if $\eta = O(1/d)$*, we have*

$$\mathrm{NoiseStab}_{\frac{1-\eta}{2}}[\mathrm{Tribes}_d] = O\Big(\frac{\eta d^2}{d2^d}\Big) = O\Big(\frac{1}{2^d}\Big).$$

Lemma 2.7 ([28]). *If* h *is a balanced Boolean function and* $\psi : \{0,1\}^r \to \{0,1\}$ *is arbitrary, then for any* δ *we have*

$$\mathrm{NoiseStab}_\delta[\psi \circ h^{\otimes r}] = \mathrm{NoiseStab}_{\frac{1}{2}-\frac{\mathrm{NoiseStab}_\delta[h]}{2}}[\psi].$$

Proof of Proposition 2.4. We take C to be $\mathrm{Tribes}_d \circ \text{Rec-Maj-3}_\ell^{\otimes d2^d}$. Given that Rec-Maj-3_ℓ is balanced, by Lemma 2.7 we have

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

Setting $\delta = \Theta\big(1/\sqrt{k}\big)$ and recalling that $3^\ell \leq k^6$, we have $\ell \geq \log_{1.1}(1/\delta)$ so we may apply Lemma 2.5 to obtain

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

As $O(k^{-.35}) \leq O(1/d)$, we may apply Lemma 2.6 with the previous inequalities to obtain

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

The bound (2.7) follows from a 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 $\mathrm{poly}(m)$. This completes the proof. $\square$

2.4 Nearly optimal hardness of learning polynomial-size monotone circuits

Given a value of k, let $m = 3^\ell d2^d$ for ℓ, d as in Proposition 2.4. Let $\mathcal{G}$ be a pseudorandom family of functions $\big\{g : \{0,1\}^k \to \{0,1\}\big\}$ secure against $\mathrm{poly}(n)$-time adversaries, where $n = mk$. Given that we have $k = \omega(\log n)$, we may apply Corollary 2.3 with the combining function from Proposition 2.4 and conclude that the class $\mathcal{C} = \{C \circ \mathsf{slice}(g)^{\otimes m} \mid g \in \mathcal{G}\}$ is hard to learn to accuracy

$$\frac{1}{2} + O\left(\frac{k^3\sqrt{\log m}}{\sqrt{m}}\right) + o(1/n) \leq \frac{1}{2} + O\left(\frac{k^{3.5}\sqrt{\log n}}{\sqrt{n}}\right). \tag{2.8}$$

We claim that the functions in $\mathcal{C}$ can, in fact, be computed by $\mathrm{poly}(n)$-size monotone circuits. This follows from a result of Berkowitz [5] that states that if a k-variable slice function is computed by a Boolean circuit of size s and depth d, then it is also computed by a monotone Boolean circuit with

MAJ gates of size $O(s+k)$ and depth $d+1$. Combining these monotone circuits for $\mathsf{slice}(g)$ with the monotone circuit for C, we obtain a $\mathrm{poly}(n)$-size monotone circuit for each function in $\mathcal{C}$.

By making various different assumptions on the hardness of one-way functions, Proposition 1.2 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 (2.8) above:

Theorem 2.8. *Suppose that standard one-way functions exist. Then for any constant $\varepsilon > 0$ there is a class $\mathcal{C}$ of $\mathrm{poly}(n)$-size monotone circuits that is hard to learn to accuracy $1/2+1/n^{1/2-\varepsilon}$.*

Proof. If $\mathrm{poly}(n)$-hard one-way functions exist then we may take $k = n^c$ in Proposition 1.2 for an arbitrarily small constant c; this corresponds to taking $d = \gamma \log k$ for γ a large constant in Proposition 2.4. The claimed bound on (2.8) 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 2.9. *Suppose that sub-exponentially hard (2^{n^α} for some fixed $\alpha > 0$) 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 $1/2+\mathrm{polylog}(n)/\sqrt{n}$.*

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 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 [26] 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 $\{\mathsf{slice}(g) \mid g \in \mathcal{G}^\star\}$ are also computable in TC^0.

We now observe that the result of Berkowitz [5] mentioned earlier for converting slice circuits into monotone circuits applies not only to Boolean circuits, but also to TC^0 circuits.

This means that the functions in $\{\mathsf{slice}(g) \mid g \in \mathcal{G}^\star\}$ are in fact computable in monotone TC^0, i. e., by polynomial-size, constant-depth circuits composed only of AND/OR/MAJ gates. As the majority function can be computed by polynomial-size, $O(\log n)$-depth monotone Boolean circuits, (see, e. g., [1]), the functions in $\{\mathsf{slice}(g) \mid g \in \mathcal{G}^\star\}$ are computable by $O(\log n)$-depth monotone Boolean circuits. Finally, using the parameters in Theorem 2.8 we have a combining function C that is a $O(\log n)$-depth poly-size monotone Boolean circuit, which implies the following lemma:

Lemma 3.1. *Let C be the monotone combining function from Proposition 2.4 and $\mathcal{G}^\star$ be a family of pseudorandom functions computable in TC^0. Then every function in $\{C \circ \mathsf{slice}(g)^{\otimes m} \mid g \in \mathcal{G}^\star\}$ is computable in monotone NC^1.*

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

Theorem 3.2. *If factoring Blum integers is hard on average for any* poly(n)*-time algorithm, then for any constant* $\varepsilon > 0$ *there is a class* $\mathcal{C}$ *of* poly(n)*-size monotone* NC^1 *circuits that is hard to learn to accuracy* $1/2 + 1/n^{1/2-\varepsilon}$.

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 that is very close to a class known to be learnable to any constant accuracy in poly(n) time. Suppose that n-bit Blum integers are 2^{n^α}-hard to factor on average for some fixed $\alpha > 0$ (which is the same hardness assumption that was earlier used by Kharitonov [19]). This means there exist $2^{n^{\alpha/2}}$-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 $\varepsilon = 1/3$, we obtain:

Lemma 3.3. *Assume that Blum integers are* 2^{n^α}*-hard to factor on average. Then there is a class* $\mathcal{C}$ *of* poly(n)*-size monotone* NC^1 *circuits that is hard for any* $2^{n^{\alpha/20}}$*-time algorithm to learn to accuracy* $1/2 + 1/n^{1/6}$.

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 substitute n' for 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 $\{0,1\}^n$ that 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 $x_1, \ldots, x_n$. 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 [2].

Lemma 3.4. *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* $\mathcal{L}$.

As every function in $\mathcal{C}'$ can be computed in NC^1, which is contained in NL, combining Lemma 3.4 with the paragraph that proceeds it, we obtain the following theorem (final line of Figure 1):

Theorem 3.5. *Suppose that Blum integers are sub-exponentially hard to factor on average. Then there is a class* $\mathcal{C}$ *of monotone functions that is hard for any* poly(n)*-time algorithm to learn to accuracy* $1/2 + o(1)$ *and that, 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 poly(n) time under the uniform distribution (without needing membership queries):

Theorem 3.6 ([31] 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* $1 - \varepsilon$ *in* poly(n)*-time.*

Theorem 3.5 is thus nearly optimal in terms of the size of the constant-depth circuits for which it establishes hardness of learning.

4 A computational analogue of the Blum-Burch-Langford lower bound

In this section we first present a simple variant of the lower bound construction in [6], 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 [6], but has the advantage that it can be easily derandomized. Indeed, as mentioned in Section 5 (and further discussed below), our construction uses a certain probability distribution over monotone DNFs, such that a typical random input x satisfies only $\mathrm{poly}(n)$ many "candidate terms" (which are terms that may be present in a random DNF drawn from our distribution). By selecting terms for inclusion in the DNF in a pseudorandom rather than truly random way, we obtain a class of $\mathrm{poly}(n)$-size monotone circuits that is hard to learn to accuracy $1/2+1/\mathrm{polylog}(n)$ (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 [6] using the pseudorandom 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.

4.1 Idea

Recall the information-theoretic lower bound from [6]. It works by defining a distribution P_s over monotone functions of the form $\{0,1\}^n \to \{0,1\}$ 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 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 $n/2$ ones). The naive idea for derandomizing 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. 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 DNFs. 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. 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 $\mathrm{poly}(n)$ 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 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 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) 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 $1/2+o(1)$.

4.2 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 fix the values of k,t later):

- Construct a monotone DNF 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:
$$(1-p)^{\binom{m/2^k}{t}} = \frac{1}{2}\,. \tag{4.1}$$
Note that for a uniform choice of $x \in \{0,1\}^n$, we expect $m/2^k$ ones in the corresponding "super-assignment" $X=(X_1,\ldots,X_m)$, and any superassignment with this many ones 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
$$f(x) = \begin{cases} f_1(x) & \text{if the number of supervariables satisfied in } x \text{ is at most } m/2^k+(m/2^k)^{2/3}, \\ 1 & \text{otherwise.} \end{cases}$$

Note that because of the final step of the construction, the function f is not actually a DNF (though it is a monotone function). Intuitively, the final step is there because if too many supervariables were satisfied in x, there could be too many (more than $\mathrm{poly}(n)$) 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 supervariables 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 DNFs f_1 carry over to the actual functions f. For most of our discussion we shall refer to P as a distribution over DNFs, meaning the functions f_1.

4.3 Information-theoretic lower bound

As discussed previously, we view the distribution P defined above as a distribution over DNFs of terms of size t over the supervariables. Each possible combination of t supervariables appears in f_1 independently with probability p and the supervariables 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 supervariables 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 "supervariable" examples and vice versa. Thus we henceforth analyze the product distribution.

We follow the proof technique of [6]. To simplify our analysis, we consider an "augmented" oracle, as in [6]. Given a query X, with ones 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 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 $1/2 \pm 1/\log n$. This is easily seen to give a lower bound of the type we require.

Following [6], 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 from [6]:

Lemma 4.1 ([6]). *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 [6]:

Lemma 4.2. *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 $1-\varepsilon_1$, the quantity $V_s(X)$ lies in the range*

$$1-(1-p)^{\left[\binom{m/2^k-(m/2^k)^{1/3}}{t}-\frac{2s\sqrt{n}}{p2^{kt}}\right]} \leq V_s(X) \leq 1-(1-p)^{\binom{m/2^k+(m/2^k)^{1/3}}{t}}. \tag{4.2}$$

Here

$$\varepsilon_1 = s \cdot \left(\frac{2\sqrt{n}}{p}+1\right)2^{-kt} + 2e^{-(m/2^k)^{1/3}/3}. \tag{4.3}$$

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

  Because 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 ones in X lies in the range $m/2^k \pm (m/2^k)^{2/3}$.**

  Using a multiplicative Chernoff bound, we have that this occurs with probability at least $1-2e^{-(m/2^k)^{1/3}/3}$. Note that for any X 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 $1-\varepsilon_1$ where

$$\varepsilon_1 = s\cdot\left(\frac{2\sqrt{n}}{p}+1\right)2^{-kt}+2e^{-(m/2^k)^{1/3}/3}.$$

Given that these events all occur, we show that $V_s(X)$ lies in the desired range. We follow the approach of [6].

For the lower bound, $V_s(X)$ is minimized when X has as few ones as possible and when as many of the 0-entries in V_s are satisfied by X as possible. So $V_s(X)$ is at least

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

For the upper bound, $V_s(X)$ is maximized when X has as many ones as possible and as few zeros as possible. So, $V_s(X)$ is at most

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

which completes the proof. □

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 $\mathrm{poly}(n)$ (so that there are at most $\mathrm{poly}(n)$ candidate terms to be checked for a "typical" input). Moreover, for any $s=\mathrm{poly}(n)$ we want both sides of (4.2) to 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 (4.1) gives $p \gg 1/n$; together with $k=\Theta(\log n)$, for any $s=\mathrm{poly}(n)$ we have $\varepsilon_1 = 1/n^{\omega(1)}$.

Now we analyze (4.2). First the lower bound:

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

(In the last step we are using the definition of p from (4.1).) 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:

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

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 4.3. *Let L be any 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 $1/2 + 1/\log n$.*

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 $1/\log n$ with any $1/\mathrm{polylog}\, n$ in the above lemma.

4.4 Computational lower bound

To obtain a computational analogue of Lemma 4.3, we make a pseudorandom choice of terms in a draw of f_1 from P.

Recall that the construction of P placed each possible term (conjunction of t supervariables) in the target function with probability p, as defined in (4.1). We first consider a distribution that uses uniform bits to approximate the probability p. This can be done by approximating $\log(p^{-1})$ with poly(n) bits, associating each term with independent uniform 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 that 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 the paper show that a 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 poly(n)-size circuit. It is obviously easy to count the number of supervariables 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" supervariables satisfied. But by construction, such an input will satisfy only poly(n) candidate terms of the monotone DNF f_1 and thus a 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 4.3, we can establish the main result of this section:

Theorem 4.4. *Suppose that standard one-way functions exist. Then there is a class $\mathcal{C}$ of poly(n)-size monotone circuits that is hard to learn to accuracy* $1/2 + 1/\mathrm{polylog}(n)$.

5 Discussion and future work

An obvious goal for future work is to establish even sharper quantitative bounds on the cryptographic hardness of learning monotone functions. Blum, Burch, and Langford [6] obtained their

$$\frac{1}{2}+\frac{\omega(\log n)}{\sqrt{n}}$$

information-theoretic lower bound by considering random monotone DNF 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 1, it is natural to consider a pseudorandom variant of the construction in [6] 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 $\log n$ that x satisfies. Indeed, the question of obtaining an efficient implementation of these "huge pseudorandom monotone DNF" has a similar flavor to Open Problem 5.4 of [13]. In that question the goal is to construct pseudorandom functions that support "subcube queries" that 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 that 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) 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 $y \leq x$).

References

[1] M. AJTAI, J. KOMLÓS, AND E. SZEMERÉDI: An $O(n\log n)$ sorting network. *Combinatorica*, 3(1):1–19, 1983. [doi:10.1007/BF02579338]. 269

[2] E. ALLENDER, L. HELLERSTEIN, P. MCCABE, T. PITASSI, AND M. SAKS: Minimizing DNF formulas and AC^0_d circuits given a truth table. In *Proc. 21st IEEE Conf. Comput. Complex. (CCC)*, pp. 237–251. IEEE Comp. Soc. Press, 2006. [CCC:10.1109/CCC.2006.27]. 261, 270

[3] R. BEALS, T. NISHINO, AND K. TANAKA: On the complexity of negation-limited Boolean networks. *SIAM J. Comput.*, 27(5):1334–1347, 1998. [SICOMP:10.1137/S0097539794275136]. 264

[4] I. BENJAMINI, G. KALAI, AND O. SCHRAMM: Noise sensitivity of Boolean functions and applications to percolation. *Publ. Math. Inst. Hautes Etudes Sci.*, 90:5–43, 1999. 258

[5] S. J. BERKOWITZ: On some relationships between monotone and non-monotone circuit complexity. Technical report, Technical Report, University of Toronto, 1982. 264, 268, 269

[6] A. BLUM, C. BURCH, AND J. LANGFORD: On learning monotone Boolean functions. In *Proc. 39th FOCS*, pp. 408–415. IEEE Comp. Soc. Press, 1998. [FOCS:10.1109/SFCS.1998.743491]. 259, 260, 261, 262, 264, 271, 273, 274, 277

[7] A. BLUM, M. FURST, M. KEARNS, AND R. LIPTON: Cryptographic primitives based on hard learning problems. In *Proc. Advances in Cryptology – CRYPTO'93*, number 773 in LNCS, pp. 278–291. Springer, 1993. [doi:10.1007/3-540-48329-2]. 258

[8] A. BLUM, K. LIGETT, AND A. ROTH: A learning theory perspective on data privacy: New hope for releasing useful databases privately. In *Proc. 40th STOC*, pp. 609–618. ACM Press, 2008. [STOC:10.1145/1374376.1374464]. 258

[9] N. BSHOUTY AND C. TAMON: On the Fourier spectrum of monotone functions. *J. ACM*, 43(4):747–770, 1996. [JACM:10.1145/234533.234564]. 258, 259

[10] D. DACHMAN-SOLED, H. K. LEE, T. MALKIN, R. A. SERVEDIO, A. WAN, AND H. WEE: Optimal cryptographic hardness of learning monotone functions. In *Proc. 35th Intern. Colloq. Automata, Languages and Programming, Part I*, pp. 36–47. Springer-Verlag, 2008. [doi:10.1007/978-3-540-70575-8_4]. 257

[11] V. FELDMAN, P. GOPALAN, S. KHOT, AND A. PONNUSWAMI: New results for learning noisy parities and halfspaces. In *Proc. 47th FOCS*, pp. 563–576. IEEE Comp. Soc. Press, 2006. [FOCS:10.1109/FOCS.2006.51]. 258

[12] O. GOLDREICH, S. GOLDWASSER, AND S. MICALI: How to construct random functions. *J. ACM*, 33(4):792–807, 1986. [JACM:10.1145/6490.6503]. 263

[13] O. GOLDREICH, S. GOLDWASSER, AND A. NUSSBOIM: On the implementation of huge random objects. In *Proc. 44th FOCS*, pp. 68–79. IEEE Comp. Soc. Press, 2003. [FOCS:10.1109/SFCS.2003.1238182]. 261, 277

[14] J. HÅSTAD, R. IMPAGLIAZZO, L. LEVIN, AND M. LUBY: A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999. [SICOMP:10.1137/S0097539793244708]. 263

[15] A. HEALY, S. VADHAN, AND E. VIOLA: Using nondeterminism to amplify hardness. *SIAM J. Comput.*, 35(4):903–931, 2006. [SICOMP:10.1137/S0097539705447281]. 263, 264, 266, 267

[16] J. KAHN, G. KALAI, AND N. LINIAL: The influence of variables on Boolean functions. In *Proc. 29th FOCS*, pp. 68–80. IEEE Comp. Soc. Press, 1988. [FOCS:10.1109/SFCS.1988.21923]. 258

[17] S. KASIVISWANATHAN, H. K. LEE, K. NISSIM, S. RASKHODNIKOVA, AND A. SMITH: What can we learn privately? In *Proc. 49th FOCS*, pp. 531–540. IEEE Comp. Soc. Press, 2008. [FOCS:10.1109/FOCS.2008.27]. 258

[18] M. J. KEARNS, M. LI, AND L. G. VALIANT: Learning Boolean formulas. *J. ACM*, 41(6):1298–1328, 1994. [JACM:10.1145/195613.195656]. 264

[19] M. KHARITONOV: Cryptographic hardness of distribution-specific learning. In *Proc. 25th FOCS*, pp. 372–381. IEEE Comp. Soc. Press, 1993. [FOCS:10.1109/SFCS.1993.366850]. 259, 260, 270

[20] M. KHARITONOV: Cryptographic lower bounds for learnability of Boolean functions on the uniform distribution. *J. Comput. System Sci.*, 50(3):600–610, 1995. [JCSS:10.1006/jcss.1995.1046]. 258, 259

[21] A. KLIVANS, R. O'DONNELL, AND R. SERVEDIO: Learning intersections and thresholds of halfspaces. *J. Comput. System Sci.*, 68(4):808–840, 2004. [JCSS:10.1016/j.jcss.2003.11.002]. 258

[22] N. LINIAL, Y. MANSOUR, AND N. NISAN: Constant depth circuits, Fourier transform and learnability. *J. ACM*, 40(3):607–620, 1993. [JACM:10.1145/174130.174138]. 258

[23] Y. MANSOUR: Learning Boolean functions via the Fourier transform. In *Theoretical Advances in Neural Computation and Learning*, pp. 391–424. Kluwer Academic Publishers, 1994. 258

[24] E. MOSSEL AND R. O'DONNELL: On the noise sensitivity of monotone functions. *Random Structures Algorithms*, 23(3):333–350, 2003. [Wiley:10.1002/rsa.10097]. 264, 267, 268

[25] E. MOSSEL, R. O'DONNELL, AND R. SERVEDIO: Learning functions of k relevant variables. *J. Comput. System Sci.*, 69(3):421–434, 2004. [JCSS:10.1016/j.jcss.2004.04.002]. 259

[26] M. NAOR AND O. REINGOLD: Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, 2004. [JACM:10.1145/972639.972643]. 259, 269

[27] V. A. NEPOMNJAŠČIĬ: Rudimentary predicates and Turing computations (in Russian). *Dokl. Akad. Nauk SSSR*, 195:282–284, 1970. English translation in *Soviet Mathematics Doklady*, Vol. 11, pp. 1642–1645, 1970. *Math. Reviews* MR02811611 (43#7326). 261

[28] R. O'DONNELL: Hardness amplification within NP. *J. Comput. System Sci.*, 69(1):68–94, 2004. [JCSS:10.1016/j.jcss.2004.01.001]. 261, 263, 264, 266, 267, 268

[29] R. O'DONNELL AND R. SERVEDIO: Learning monotone decision trees in polynomial time. *SIAM J. Comput.*, 37(3):827–844, 2007. [SICOMP:10.1137/060669309]. 258, 259

[30] A. RAZBOROV: Lower bounds on the monotone network complexity of the logical permanent. *Mat. Zametki*, 37:887–900, 1985. English translation in *Math. Notes of the Academy of Sciences of the USSR*, Vol 37, pp. 485-493, 1985. 262

[31] R. SERVEDIO: On learning monotone DNF under product distributions. *Inform. Comput.*, 193(1):57–74, 2004. 259, 260, 270

[32] É. TARDOS: The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica*, 8(1):141–142, 1988. [10.1007BF02122563]. 262

[33] L. TREVISAN: List decoding using the XOR lemma. In *Proc. 44th FOCS*, pp. 126–135. IEEE Comp. Soc. Press, 2003. [FOCS:10.1109/SFCS.2003.1238187]. 263, 264

[34] L. G. VALIANT: A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. [ACM:10.1145/1968.1972]. 258

[35] L. G. VALIANT: Negation is powerless for Boolean slice functions. *SIAM J. Comput.*, 15:531–535, 1986. [SICOMP:10.1137/0215037]. 264

[36] K. VERBEURGT: Learning DNF under the uniform distribution in quasi-polynomial time. In *Proc. 3rd Ann. Workshop Comput. Learn. Theory*, pp. 314–326. Morgan Kaufman, 1990. 259

AUTHORS

Dana Dachman-Soled [About the author]
Department of Computer Science
Columbia University, New York, NY
dglasner@cs.columbia.edu

Homin K. Lee [About the author]
Department of Computer Science
University of Texas at Austin
homin@cs.utexas.edu
http://www.cs.utexas.edu/~homin

Tal Malkin [About the author]
Assistant Professor
Department of Computer Science
Columbia University, New York, NY
tal@cs.columbia.edu
http://www.cs.columbia.edu/~tal

Rocco A. Servedio [About the author]
Associate Professor
Columbia University, New York, NY
rocco@cs.columbia.edu
http://www.cs.columbia.edu/~rocco

Andrew Wan [About the author]
Department of Computer Science
Columbia University, New York, NY
atw12@columbia.edu
http://www.cs.columbia.edu/~atw12

Hoeteck Wee [About the author]
Assistant Professor
Department of Computer Science
Queens College, City University of New York
hoeteck@cs.qc.cuny.edu
http://www.cs.qc.edu/~hoeteck/

ABOUT THE AUTHORS

DANA DACHMAN-SOLED (a. k. a. DANA GLASNER) is a Ph. D. candidate in the Department of Computer Science at Columbia University, supervised by Tal Malkin and Rocco Servedio. She received her undergraduate degree in Computer Science from Yeshiva University. Dana's research interests include property testing and cryptography. She enjoys life in New York City with her husband and their 17-month-old[3] son.

HOMIN K. LEE is a postdoctoral researcher under the direction of Adam Klivans at the University of Texas at Austin[4]. He received his Ph. D. in 2009 from Columbia University where he was advised by Tal Malkin and Rocco A. Servedio. His thesis was titled *On the Learnability of Monotone Functions*, and his research interests include computational learning theory and the analysis of Boolean function. He is often hungry.

TAL MALKIN received her Ph. D. in Computer Science from MIT in 2000, under the supervision of Shafi Goldwasser. After spending three years at AT&T Labs Research, she joined the Department of Computer Science at Columbia University, where she heads the Cryptography Lab. Her research interests are in cryptography, security, complexity theory, and related areas, with foundations of cryptography being closest to her heart. She enjoys living in New York City and spending time with (and without) her husband, two sons, and cat. In theory she loves theater, ice hockey, editing the journal "Theory of Computing," and sleep, but in practice she doesn't get to enjoy these activities too often.

ROCCO A. SERVEDIO is an associate professor in the Department of Computer Science at Columbia University. He graduated from Harvard University in 2001 where his Ph. D. was supervised by Leslie Valiant. He is interested in computational learning theory, computational complexity, and other topics. He enjoys spending time with his family and hopes to drink a quart of stout with Herman Melville in the afterlife.

[3]At the time of submission.
[4]The author performed this work as a Ph. D. candidate at Columbia University.

ANDREW WAN is a Ph. D. candidate at Columbia University, advised by Tal Malkin and Rocco Servedio. His interests include complexity theory, cryptography, and computational learning theory. Before graduate school, he was a student of philosophy at Columbia University and enjoyed playing the piano, the trumpet, and the accordion. Although he still enjoys playing music, the PAC model rarely affords him the time.

HOETECK WEE is an assistant professor at Queens College, CUNY[5]. He received his Ph. D. from UC Berkeley under the supervision of Luca Trevisan and his B. S. from MIT. He was previously a postdoc at Columbia University and a visiting student at Tsinghua University and IPAM. Hoeteck currently lives in Manhattan close to the cafés in order to cut down on his commute. He's working to convince more people that "black box is the new black."

[5]This work was performed while the author was a postdoc at Columbia University.