Optimal Cryptographic Hardness of Learning Monotone Functions

Dana Dachman-Soled, Homin K. Lee, Tal Malkin,
Rocco A. Servedio, Andrew Wan, and Hoeteck Wee

{dglasner,homin,tal,rocco,atw12,hoeteck}@cs.columbia.edu

Abstract. A wide range of positive and negative results have been established for learning different classes of Boolean functions from uniformly distributed random examples. However, polynomial-time algorithms have thus far been obtained almost exclusively for various classes of *monotone* functions, while the computational hardness results obtained to date have all been for various classes of general (nonmonotone) functions. Motivated by this disparity between known positive results (for monotone functions) and negative results (for nonmonotone functions), we establish strong computational limitations on the efficient learnability of various classes of monotone functions.

We give several such hardness results which are provably almost optimal since they nearly match known positive results. Some of our results show cryptographic hardness of learning polynomial-size monotone circuits to accuracy only slightly greater than $1/2 + 1/\sqrt{n}$; this accuracy bound is close to optimal by known positive results (Blum *et al.*, FOCS '98). 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 '04). Our main tool is a complexity-theoretic approach to hardness amplification via noise sensitivity of monotone functions that was pioneered by O'Donnell (JCSS '04).

1 Introduction

More than two decades ago Valiant introduced the Probably Approximately Correct (PAC) model of learning Boolean functions from random examples [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. 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 which satisfies $\Pr[f(x) \neq h(x)] \leq \epsilon$ 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 [Man94], circuit complexity [LMN93], noise sensitivity and influence of variables in Boolean functions [KKL88,BKS99,KOS04,OS07], coding theory [FGKP06], 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.

Monotonicity makes learning easier. 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 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 to perfect accuracy in poly(n) time, and a more complex algorithm [BT96] learns monotone $\tilde{O}(\log^2(n))$-juntas to any constant accuracy in poly(n) time. In contrast, the fastest known algorithm for learning arbitrary k-juntas runs in time $n^{.704k}$ [MOS04].
2. The fastest known uniform distribution learning algorithm for the general class of s-term DNF, due to Verbeurgt [Ver90], runs in time $n^{O(\log s)}$ to learn to any constant accuracy. In contrast, for s-term monotone DNF [Ser04] gives an algorithm which runs in $s^{O(\log s)}$ time. 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 (this follows from [Ver90]), but poly(n)-size decision trees that compute monotone functions can be learned to any constant accuracy in poly(n) time [OS07].
4. No poly(n)-time algorithm can learn the general class of all Boolean functions on $\{0,1\}^n$ to accuracy better than $\frac{1}{2} + \frac{\mathrm{poly}(n)}{2^n}$, but a simple polynomial-time algorithm can learn the class of all monotone Boolean functions to accuracy $\frac{1}{2} + \frac{\Omega(1)}{\sqrt{n}}$ [BBL98]. We note also that the result of [BT96] 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 which 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 of logarithmic-depth, polynomial-size AND/OR/NOT circuits is hard to learn under the uniform distribution.

Subsequently Kharitonov showed [Kha93] that if factoring Blum integers is 2^{n^ϵ}-hard for some fixed $\epsilon > 0$, then even the class AC^0 of constant-depth, polynomial-size AND/OR/NOT circuits 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 (composed of MAJ and NOT gates) cannot be learned in polynomial time under uniform. 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 above 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? 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.

In [BBL98] Blum *et al.* 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 $\mathrm{poly}(n)$ many membership queries. This is an information-theoretic bound which is proved using randomly generated monotone DNF formulas of size (roughly) $n^{\log n}$. A natural goal is to obtain *computational* lower bounds for learning polynomial-time-computable monotone functions which match, or nearly match, this level of hardness (which is close to optimal by the $(\frac{1}{2} + \frac{\Omega(1)}{\sqrt{n}})$-accuracy algorithm of Blum *et al.* described above). We prove near-optimal hardness for learning polynomial-size monotone circuits:

Theorem 1 (informal). *If one-way functions exist, then there is a class of* $\mathrm{poly}(n)$*-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 subexponential 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.

Hardness assumption	Complexity of f	Accuracy bound	Ref.
none	random $n^{\log n}$-term mono. DNF	$\frac{1}{2} + \frac{\omega(\log n)}{n^{1/2}}$	[BBL98]
OWF (poly)	poly-size monotone circuits	$\frac{1}{2} + \frac{1}{n^{1/2-\epsilon}}$	Thm. 1
OWF (2^{n^α})	poly-size monotone circuits	$\frac{1}{2} + \frac{\mathrm{poly}(\log n)}{n^{1/2}}$	Thm. 3
factoring BI (poly)	monotone NC^1-circuits	$\frac{1}{2} + \frac{1}{n^{1/2-\epsilon}}$	Thm. 4
factoring BI (2^{n^α})	depth-d, size $2^{(\log n)^{O(1)/(d+1)}}$ AND/OR/NOT circuits	$\frac{1}{2} + o(1)$	Thm. 5

Fig. 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^α}" means that the problems are intractable for poly(n)- and 2^{n^α}-time algorithms respectively (for some fixed $\alpha > 0$). Recall that the poly(n)-time algorithm of [BBL98] for learning monotone functions implies that the best possible accuracy bound for monotone functions is $\frac{1}{2} + \frac{\Omega(1)}{n^{1/2}}$.

– 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 in [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 of [Ser04] 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 1.

Proof techniques. A natural first approach is to try to "pseudorandomize" [BBL98]'s construction of random $n^{\log n}$-term monotone DNFs. We were not able to do this directly; indeed, as we discuss in Section 3, pseudorandomizing the [BBL98] construction seems closely related to an open problem of Goldreich *et al.* from [GGN03]. However, it turns out that a closely related approach does yield some results along the desired lines; in Appendix C we present and analyze a simple variant of the [BBL98] 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 [BBL98], 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 "99%"), the $\frac{1}{\mathrm{polylog}(n)}$ factor is rather far from the $\frac{O(\log n)}{\sqrt{n}}$ factor that one might hope for as described above.

In Section 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 which 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 1). We obtain our result for constant depth, sub-polynomial-size circuits (row 5 of Figure 1) by augmenting this approach with an argument which at a high level is similar to one used in [AHM$^+$06], by "scaling down" and modifying our hard-to-learn functions using a variant of Nepomnjaščiĭ's theorem [Nep70].

1.3 Preliminaries

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

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

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

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 $(1) < \frac{1}{2} + \epsilon(n)$ for all sufficiently large n. As noted in [BBL98], it is straightforward to transform a lower bound of this sort into a lower bound for the usual ϵ, δ formulation of PAC learning.

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

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 [O'D04,HVV06], 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, given in Section 2.2, 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 *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. In Section 2.2 we adapt the analysis in [O'D04,HVV06] 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 from [O'D04,MO03] can be used to obtain a "simple" monotone combining function with low noise stability. In Section 2.4 we establish Theorems 2 and 3 (lines 2 and 3 of Figure 1) by making different assumptions about the hardness of the initial pseudorandom functions. Finally, we use more specific assumptions about the hardness of factoring Blum integers to establish Theorems 4 and 5 (lines 4 and 5 of Figure 1), which extend our hardness results to very simple circuit classes; because of space constraints these results are deferred to Appendix B.

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

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

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

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

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

It is immediate that $\mathsf{slice}(g)$ is a monotone Boolean function for any g.

Bias and noise stability. Following the analysis in [O'D04,HVV06], we shall study the bias and noise stability of various Boolean functions. Specifically, we adopt the following notations and definitions from [HVV06]. The *bias* of a 0-1 random variable X is defined to be

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

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

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

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 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*

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

This easily yields Corollary 1, which is an analogue of Lemma 1 with pseudorandom rather than truly random functions, and which we use to obtain our main hardness of learning results.

Proof of Lemma 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 upper bounding

$$\Pr_{g \in \mathcal{G}', x \in \{0,1\}^n}\left[L^f(1^n) \to h;\ h(x) = f(x)\right] \tag{2}$$

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 $f \stackrel{\text{def}}{=} C \circ \mathsf{slice}(g)^{\otimes m}$ 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 poly$(n)$-time machine to simulate oracle access to f if it is given oracle access to g. So (2) is at most

$$\Pr_{g \in \mathcal{G}',\, x \in \{0,1\}^n}\Big[L^g(1^n) \to h;\ h(x) = (C \circ \mathsf{slice}(g)^{\otimes m})(x)\Big]. \tag{3}$$

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) is bounded by

$$\Pr_{g, g' \in \mathcal{G}',\, x \in \{0,1\}n}\Big[L^g(1^n) \to h;\ h(x) = (C \circ \mathsf{slice}(g')^{\otimes m})(x)\Big] + \frac{m \cdot \mathrm{poly}(n)}{2^k}. \tag{4}$$

The first summand in (4) 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 which, for any fixed input x, outputs 1 if and only if the random variable $(C \circ \mathsf{slice}(g')^{\otimes m})(x)$ (we emphasize that the randomness here is over the choice of g') is biased towards 1. The expected accuracy of this h is precisely

$$\frac{1}{2} + \frac{1}{2}\,\mathrm{ExpBias}[C \circ \mathsf{slice}(g')^{\otimes m}]. \tag{5}$$

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 $\mathsf{slice}(g')$ (where g' is truly random) is "δ-random" in the sense of ([HVV06], Definition 3.1), *i.e.,* it is balanced, truly random on inputs in the middle slice, and deterministic on all other inputs. This means that we may apply a technical lemma [HVV06, Lemma 3.7]) to $\mathsf{slice}(g')$ (see also [O'D04]) to obtain

$$\mathrm{ExpBias}[C \circ \mathsf{slice}(g')^{\otimes m}] \leq \sqrt{\mathrm{NoiseStab}_\delta[C]}. \tag{6}$$

Combining (4), (5) and (6) and recalling that $k = \omega(\log n)$, we obtain Lemma 1. □

Corollary 1. *Let $C : \{0,1\}^m \rightarrow \{0,1\}$ be a polynomial-time computable function. Let $\mathcal{G}$ be a pseudorandom family of functions from $\{0,1\}^k$ to $\{0,1\}$ which are secure* *against* $\mathrm{poly}(n)$*-time adversaries, 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*

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

Proof. The corollary follows from the fact that (3) must differ from its pseudorandom counterpart,

$$\Pr_{g \in \mathcal{G},\ x \in \{0,1\}^n}\Big[L^g(1^n) \to h;\ h(x) = (C \circ \mathsf{slice}(g)^{\otimes m})(x)\Big], \tag{7}$$

by less than $1/n^2$ (in fact by less than any fixed $1/\mathrm{poly}(n)$). Otherwise, we would easily obtain a $\mathrm{poly}(n)$-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 1 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 [O'D04,MO03] to obtain:

Proposition 1. *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* $\mathrm{poly}(m)$*-size,* $\log(m)$*-depth monotone circuits such that*

$$\mathrm{NoiseStab}_{\Theta(1/\sqrt{k})}[C] \le O\Big(\frac{k^6 \log m}{m}\Big). \tag{8}$$

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 $\mathrm{NoiseStab}_{\Theta(1/\sqrt{k})}[C]$ for every monotone m-variable function C, so the above upper bound is fairly close to the best possible (within a $\mathrm{polylog}(m)$ factor if $m = 2^{k^{\Theta(1)}}$).

Following [O'D04,HVV06], 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 ([O'D04]). *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* $x^i \in \{0,1\}^{3^{\ell-1}}$*,*

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

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

Lemma 3 ([MO03]). *Let* $\mathrm{Tribes}_d : \{0,1\}^{d2^d} \rightarrow \{0,1\}$ *denote the "tribes" function on* $d2^d$ *variables, i.e., the read-once* 2^d*-term monotone* d*-DNF*

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

Then if $\eta \le O(1/d)$*, we have* $\mathrm{NoiseStab}_{\frac{1-\eta}{2}}[\mathrm{Tribes}_d] \le O\Big(\frac{\eta d^2}{d2^d}\Big) \le O\Big(\frac{1}{2^d}\Big)$.

Lemma 4 ([O'D04]). *If* h *is a balanced Boolean function and* $\psi : \{0,1\}^r \rightarrow \{0,1\}$ *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 1: We take C to be $\mathrm{Tribes}_d \circ \text{Rec-Maj-3}_\ell^{\otimes d2^d}$. Since Rec-Maj-3_ℓ is balanced, by Lemma 4 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 $\delta = \Theta(1/\sqrt{k})$ and recalling that $3^\ell \le k^6$, we have $\ell \ge \log_{1.1}(1/\delta)$ so we may apply Lemma 2 to obtain

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

Since $O(k^{-.35}) \le O(1/d)$, we may apply Lemma 3 with the previous inequalities to obtain

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

The bound (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 $\mathrm{poly}(m)$, and the proposition is proved. □

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 1. Let $\mathcal{G}$ be a pseudorandom family of functions $\{g : \{0,1\}^k \rightarrow \{0,1\}\}$ secure against $\mathrm{poly}(n)$-time adversaries, where $n = mk$. Since we have $k = \omega(\log n)$, we may apply Corollary 1 with the combining function from Proposition 1 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

$$\frac{1}{2} + O\Big(\frac{k^3\sqrt{\log m}}{\sqrt{m}}\Big) + o(1/n) \le \frac{1}{2} + O\Big(\frac{k^{3.5}\sqrt{\log n}}{\sqrt{n}}\Big). \tag{9}$$

We claim that in fact the functions in $\mathcal{C}$ can be computed by $\mathrm{poly}(n)$-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 $\mathsf{slice}(g)$ 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 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 (9) above:

Theorem 2. *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 $\mathrm{poly}(n)$-hard one-way functions exist then we may take $k = n^c$ in Proposition 2 for arbitrarily small constant c; this corresponds to taking $d = C \log k$ for C a large constant in Proposition 1. The claimed bound on (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 3. *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 $\frac{1}{2} + \frac{\mathrm{polylog}(n)}{n^{1/2}}$.*

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

3 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. [BBL98] obtain their $\frac{1}{2} + \frac{\omega(\log n)}{n^{1/2}}$ information-theoretic lower bound by considering random monotone DNF which 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 [BBL98] 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" has a similar flavor to Open Problem 5.4 of [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) 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 $y \leq x$).

In Appendix C we present a variant of the [BBL98] construction in which a typical input satisfies only $\mathrm{poly}(n)$ many candidate terms; this is the key feature enabling us to "pseudorandomize" the construction.

References

[AHM+06] E. Allender, L. Hellerstein, P. McCabe, T. Pitassi, and M. Saks. Minimizing DNF Formulas and AC_d^0 Circuits Given a Truth Table. In *CCC*, pages 237–251, 2006.

[AKS83] M. Ajtai, J. Komlos, and E. Szemeredi. An $O(n \log n)$ sorting network. *Combinatorica*, 3(1):1–19, 1983.

[BBL98] A. Blum, C. Burch, and J. Langford. On learning monotone boolean functions. In *39th FOCS*, pages 408–415, 1998.

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

[BFKL93] A. Blum, M. Furst, M. Kearns, and R. Lipton. Cryptographic Primitives Based on Hard Learning Problems. In *CRYPTO '93*, pages 278–291, 1993.

[BKS99] I. Benjamini, G. Kalai, and O. Schramm. Noise sensitivity of Boolean functions and applications to percolation. *Inst. Hautes Études Sci. Publ. Math.*, 90:5–43, 1999.

[BLR08] Avrim Blum, Katrina Ligett, and Aaron Roth. A learning theory perspective on data privacy: New hope for releasing useful databases privately. Manuscript, 2008.

[BT96] N. Bshouty and C. Tamon. On the Fourier spectrum of monotone functions. *Journal of the ACM*, 43(4):747–770, 1996.

[FGKP06] V. Feldman, P. Gopalan, S. Khot, and A. Ponnuswami. New results for learning noisy parities and halfspaces. In *47th FOCS*, pages 563–576, 2006.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *JACM*, 33(4):210–217, 1986.

[GGN03] O. Goldreich, S. Goldwasser, and A. Nussboim. On the Implementation of Huge Random Objects. Technical report, ECCC Technical Report 045, 2003.

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HVV06] A. Healy, S. Vadhan, and E. Viola. Using Nondeterminism to Amplify Hardness. *SIAM Journal on Computing*, 35(4):903–931, 2006.

[Kha93] M. Kharitonov. Cryptographic hardness of distribution-specific learning. In *25th STOC*, pages 372–381, 1993.

[Kha95] M. Kharitonov. Cryptographic lower bounds for learnability of Boolean functions on the uniform distribution. *JCSS*, 50:600–610, 1995.

[KKL88] J. Kahn, G. Kalai, and N. Linial. The influence of variables on boolean functions. In *29th FOCS*, pages 68–80, 1988.

[KLN+08] Shiva Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? Manuscript, 2008.

[KLV94] Michael J. Kearns, Ming Li, and Leslie G. Valiant. Learning boolean formulas. *J. ACM*, 41(6):1298–1328, 1994.

[KOS04] A. Klivans, R. O'Donnell, and R. Servedio. Learning intersections and thresholds of halfspaces. *JCSS*, 68(4):808–840, 2004.

[LMN93] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform and learnability. *Journal of the ACM*, 40(3):607–620, 1993.

[Man94] Y. Mansour. *Learning Boolean functions via the Fourier transform*, pages 391–424. Kluwer Academic Publishers, 1994.

[MO03] Elchanan Mossel and Ryan O'Donnell. On the noise sensitivity of monotone functions. *Random Struct. Algorithms*, 23(3):333–350, 2003.

[MOS04] E. Mossel, R. O'Donnell, and R. Servedio. Learning functions of k relevant variables. *J. Comput. & Syst. Sci.*, 69(3):421–434, 2004.

[Nep70] V.A. Nepomnjascii. Rudimentary predicates and Turing calculations. *Math Dokl.*, 11:1462–1465, 1970.

[NR04] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM*, 51(2):231–262, 2004.

[O'D04] R. O'Donnell. Hardness amplification within NP. *JCSS*, 69(1):68–94, 2004.

[OS07] R. O'Donnell and R. Servedio. Learning monotone decision trees in polynomial time. *SIAM Journal on Computing*, 37(3):827–844, 2007.

[Raz85] A. Razborov. Lower bounds on the monotone network complexity of the logical permanent. *Mat. Zametki*, 37:887–900, 1985.

[Ser04] R. Servedio. On learning monotone DNF under product distributions. *Information and Computation*, 193(1):57–74, 2004.

[Val84] L. Valiant. A theory of the learnable. *CACM*, 27(11):1134–1142, 1984.

[Ver90] K. Verbeurgt. Learning DNF under the uniform distribution in quasi-polynomial time. In *3rd COLT*, pages 314–326, 1990.

A Preliminaries

Circuit complexity. We shall consider various classes of circuits computing Boolean functions. We assume familiarity with standard circuit classes such as NC^1 (polynomial-size, logarithmic-depth, bounded fan-in circuits of AND/OR/NOT gates), AC^0 (polynomial-size, constant-depth, unbounded fan-in circuits of AND/OR/NOT gates) and TC^0 (polynomial-size, constant-depth unbounded fan-in circuits of MAJ/NOT 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 [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.

Learning. 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 which are allowed to run in poly(n) time, so the learning algorithm L may be viewed as an oracle 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.

Pseudorandom functions. Pseudorandom functions [GGM86] are the main cryptographic primitive that underlie our constructions. Fix $k(n) \leq n$, 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

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

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 2. *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 $\{g : \{0,1\}^{k(n)} \to \{0,1\}\}$ that is $(t(k(n)))^c$-secure.*

B 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 $\{\mathsf{slice}(g) \mid g \in \mathcal{G}^\star\}$ 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 $\{\mathsf{slice}(g) \mid g \in \mathcal{G}^\star\}$ are in fact computable in monotone TC^0, *i.e.,* by polynomial-size, constant-depth circuits composed only of MAJ gates. Since the majority function can be computed by polynomial-size, $O(\log n)$-depth AND/OR circuits, (see e.g. [AKS83]), we easily obtain the following:

Lemma 5. *Let C be the monotone combining function from Proposition 1 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 4. *If factoring Blum integers is hard on average for any poly(n)-time algorithm, then for any constant $\epsilon > 0$ there is a class $\mathcal{C}$ of 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 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 the following

Lemma 6. *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 $\frac{1}{2} + \frac{1}{n^{1/6}}$.*

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 $x_1, \ldots, x_n$. 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 [AHM$^+$06].

Lemma 7. *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 7 with the paragraph that proceeds it, we obtain the following theorem (final line of Figure 1):

Theorem 5. *Suppose that Blum integers are subexponentially hard to factor on average. Then there is a class $\mathcal{C}$ of monotone functions which is hard for any 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 poly(n) time under the uniform distribution (without needing membership queries):

Theorem 6 ([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 $\mathrm{poly}(n)$-time.*

Theorem 5 is thus nearly optimal in terms of the size of the constant-depth circuits for which it establishes hardness of learning.

C A Computational Analogue of [BBL98]'s Lower Bound

In this section we first present a simple variant of the [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 [BBL98], but has the advantage that it can be easily "pseudorandomized". Indeed, as mentioned in Section 3 (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" (terms which 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 poly(n)-size monotone circuits which is hard to learn to accuracy $\frac{1}{2} + \frac{1}{\mathrm{polylog}(n)}$ (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 [BBL98] 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.

C.1 Idea

Recall the [BBL98] information-theoretic lower bound. It works by defining a distribution P_s over monotone functions $\{0,1\}^n \to \{0,1\}$ which is the following: 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 $n/2$ 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. 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 an 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 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 "supervariable" (corresponding to the AND of all k variables in the block). We then construct the DNF by randomly choosing length-t terms over these supervariables. 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 $1/2 + o(1)$.

C.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 supervariables). 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 supervariables 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:

$$(1-p)^{\binom{m/2^k}{t}} = 1/2. \tag{10}$$

  Note that for a uniform $x \in \{0,1\}^n$, we expect the corresponding "superassignment" $X = (X_1, \ldots, X_m)$ to have $m/2^k$ 1's in it. So t 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.

C.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 [BBL98]. To simplify our analysis, we consider an "augmented" oracle, as in [BBL98]. 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. 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 relevant to 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 $s = \mathrm{poly}(n)$, 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 [BBL98], 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 [BBL98]:

Lemma 8 ([BBL98]). *After s queries, with probability $1 - \epsilon^{-s/4}$, 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 [BBL98]:

Lemma 9. *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*

$$1-(1-p)^{[\binom{m/2^k-(m/2^k)^{1/3}}{t}-\frac{2s\sqrt{n}}{p2^{kt}}]} \leq V_s(X) \leq 1-(1-p)^{\binom{m/2^k+(m/2^k)^{1/3}}{t}}. \quad (11)$$

Here

$$\epsilon_1 = s \cdot (\frac{2\sqrt{n}}{p}+1)2^{-kt} + 2e^{-(m/2^k)^{1/3}/3}. \quad (12)$$

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 relevant to X.**
  There are at most s 1-entries in V_s and the probability that any one is relevant to X is 2^{-kt}. Therefore the probability that any 1-entry is relevant to X is at most $s2^{-kt}$ and the probability that none of the 1-entries in V_s are relevant to 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.**
  Using Lemma 8, the expected number of 0-entries in V_s relevant to 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 a multiplicative 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 occuring, 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 [BBL98].

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

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

□

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 $\mathrm{poly}(n)$ candidate terms to be checked for a "typical" input). Moreover, for any $s = \mathrm{poly}(n)$ we want (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 (10) gives $p \gg 1/n$; together with $k = \Theta(\log n)$, for any $s = \mathrm{poly}(n)$ we have $\epsilon_1 = 1/n^{\omega(1)}$.

Now we analyze (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 (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

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

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:

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

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 10. *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 $\frac{1}{2} + \frac{1}{\log n}$.*

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.

C.4 Computational Lower Bound

To obtain a computational analogue of Lemma 10, we "pseudorandomize" the choice of terms in a draw of f_1 from P.

Recall that the constuction P placed each possible term (conjunction of t supervariables) in the target function with probability p, as defined in (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 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 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 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 $\mathrm{poly}(n)$ candidate terms of the monotone DNF 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 10, we can establish the main result of this section:

Theorem 7. *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 accracy* $\frac{1}{2} + \frac{1}{\mathrm{polylog}(n)}$.