On Learning Monotone Boolean Functions

Avrim Blum* Carl Burch† John Langford‡

Abstract

We consider the problem of learning monotone Boolean functions over $\{0,1\}^n$ under the uniform distribution. Specifically, given a polynomial number of uniform random samples for an unknown monotone Boolean function f, and given polynomial computing time, we would like to approximate f as well as possible. We describe a simple algorithm that we prove achieves error at most $1/2 - \Omega(1/\sqrt{n})$, improving on the previous best bound of $1/2 - \Omega((\log^2 n)/n)$. We also prove that no algorithm, given a polynomial number of samples, can guarantee error $1/2 - \omega((\log n)/\sqrt{n})$, improving on the previous best hardness bound of $O(1/\sqrt{n})$. These lower bounds hold even if the learning algorithm is allowed membership queries. Thus this paper settles to an $O(\log n)$ factor the question of the best achievable error for learning the class of monotone Boolean functions with respect to the uniform distribution.

1. Introduction

A *monotone Boolean function* f maps bit vectors $\{0,1\}^n$ to $\{0,1\}$, such that if $f(x) = 1$, then flipping any bit of x from 0 to 1 keeps $f(x) = 1$. (This is sometimes called a *positive Boolean function*, or, in combinatorics, a *monotone increasing set system*.) Because monotone functions encompass a very broad class of Boolean expressions—specifically all circuits including no negations—algorithms to learn them are of special interest.

For a particular Boolean target concept f and a hypothesis function h, we define the *error* of h as the fraction of points x where $h(x) \neq f(x)$; that is, the error is $\Pr[h(x) \neq f(x)]$ for bit vectors x chosen uniformly from $\{0,1\}^n$. (All probabilities in this paper are over the uniform distribution.) Because of the generality of monotone functions, this paper usually discusses errors of nearly $1/2$. To simplify discussion, we sometimes use the closely related concept of *correlation*, which for error γ is defined as $1 - 2\gamma$.

The algorithms we discuss are for learning monotone functions with respect to the uniform distribution. In other words, the algorithm has access to an *example oracle* $SAMPLE$ for a hidden monotone function f, that when invoked produces a pair $(x, f(x))$ where x is chosen uniformly at random from $\{0,1\}^n$. The goal of the learning algorithm is to produce a good approximation to f (a hypothesis with low error over the uniform distribution) using polynomial time and a polynomial number of samples. Achieving error $1/2$ is trivial; achieving error $1/2 - 1/poly(n)$ is called "weak learning"; and achieving arbitrarily low error ϵ in time polynomial in $1/\epsilon$ is called "strong learning". A more powerful oracle than $SAMPLE$ is the *membership query oracle* $MEMBER$ that allows the algorithm to query f at arbitrary points of its choosing. Our upper bounds (algorithms) use only $SAMPLE$ but our lower bounds (hardness results) hold even if the algorithm has access to $MEMBER$ as well.

One reason the problem of learning monotone functions from random examples is interesting is that for several important subclasses of monotone functions, the upper bounds for the general class of monotone functions are the best known. The most prominent such subclass, to which the algorithm of this paper is an improvement, is the class of monotone DNF formulas. (Some restricted subclasses of monotone DNF, such as μDNF, where each variable appears at most once, have known strong-learning algorithms [6, 7].) The study of learning monotone functions under the uniform distribution is also inspired by the fact that they stand at the threshold of what is weakly learnable. Kearns, Li, and Valiant observe on proposing the problem: "Generalization in any direction—uniform distributions to arbitrary distributions, weak learning to strong learning, or monotone functions to arbitrary functions—results in intractability" [6].

The first results on learning monotone functions over the uniform distribution were by Kearns et al [6]. Their algorithm begins by drawing a sample of data and producing the constant-one function ($h(x) = 1$) or the constant-zero function ($h(x) = 0$) if the number of positive examples seen differs significantly from the number of negatives seen.

*Carnegie Mellon University. E-mail: avrim+@cs.cmu.edu. Supported in part by NSF National Young Investigator grant CCR-9357793.

†Carnegie Mellon University. E-mail: cburch+@cs.cmu.edu. Supported in part by a National Science Foundation Graduate Fellowship.

‡Carnegie Mellon University. E-mail: jcl+@cs.cmu.edu.

Otherwise, their algorithm outputs the single-variable function ($f(x) = x_i$) that has highest observed correlation with the data. By results of Aldous [1] there must exist some variable with correlation $\Omega(1/n)$, and thus their error is $1/2 - \Omega(1/n)$. Bshouty and Tamon [4] improve on this guarantee using results of Kahn, Kalai, and Linial [5]. They demonstrate an algorithm which outputs linear-threshold functions and guarantees error at most $1/2 - \Omega((\log^2 n)/n)$. Bshouty and Tamon also describe super-polynomial-time algorithms with better guarantees.

In this paper we present an algorithm that achieves error at most $1/2 - \Omega(1/\sqrt{n})$. The approach is especially simple. In brief, we prove that one of three functions achieves this correlation: the constant-one function, the constant-zero function, or the majority function ($h(x) = 1$ iff $\sum_i x_i > n/2$). By sampling enough times to determine which of these three is best-correlated, we achieve the result.

We complement this result with a lower bound showing that this simple algorithm is nearly the best possible. Specifically, no algorithm, given only a polynomial number of accesses to the target function, can guarantee error $1/2 - \omega((\log n)/\sqrt{n})$, even if it can use both the $SAMPLE$ and $MEMBER$ oracles. The best previous negative result, using "slice" functions, is that no subexponential-time algorithm can guarantee error $O(1/\sqrt{n})$ [6].

In this paper, $\|x\|$ represents the number of 1 bits in x; in other words, $\|x\| = \sum_i x_i$. We use X_k to represent the set of size-k bit vectors: $X_k = \{x \mid \|x\| = k\}$.

2. Learning a fair monotone function

A *fair Boolean function* is a Boolean function that labels exactly half the points with 1; that is, f is fair if $\Pr[f(x) = 1] = 1/2$. In Section 3 (Lemma 7), we prove that a learning algorithm for fair monotone functions implies a learning algorithm for general monotone functions with only a small loss in error; thus, it suffices to assume that the target function is fair. In this section we show that an especially simple algorithm—namely, the algorithm that blindly returns the majority function over all variables—learns fair monotone functions with error at most $1/2 - \Omega(1/\sqrt{n})$. That is, we show that the majority function correlates weakly with every fair monotone function.

The intuition motivating this proposition is that the best fair monotone function for foiling the majority function would be the most lopsided function imaginable, the single-variable function $f(x) = x_1$. (Surprisingly, the converse is not true: Ben-Or and Linial demonstrate that the majority function does not minimize correlation with the best single-variable function [2].) The single-variable function disagrees with the majority function on a

$$\frac{1}{2} - \frac{1}{2} \cdot \frac{\binom{n-1}{(n-1)/2}}{2^n} \approx \frac{1}{2} - \frac{1}{\sqrt{2\pi n}} \approx \frac{1}{2} - \frac{0.4}{\sqrt{n}}$$

fraction of the points. Although we believe that this is the true worst-case error of the majority function, what we prove is a slightly worse approximation guarantee.

Theorem 1 *Say $f : \{0,1\}^n \to \{0,1\}$ is a fair monotone function. Then the majority function*

$$h(x) = \begin{cases} 1 & \text{if } \|x\| > n/2 \\ 0 & \text{otherwise} \end{cases}$$

has error at most $1/2 - 0.1/\sqrt{n}$.

We assume for simplicity in this section that n is odd; this assumption is removed in Section 3 (Lemma 6).

To prove the theorem, we analyze the quantities p_k, which we define as the fraction of the points $x \in X_k$ such that $f(x) = 1$:

$$\begin{aligned} p_k &= \Pr[f(x) = 1 \mid x \in X_k] \\ &= \frac{|\{x \in X_k \mid f(x) = 1\}|}{\binom{n}{k}}. \end{aligned}$$

(Recall that we defined X_k as $\{x \mid \|x\| = k\}$, the set of bit vectors with exactly k ones.)

It is easy to see that the p_k are non-decreasing with k: Imagine placing 1's at random into an example that initially is all 0's; the probability that the example is positive can only increase as more 1's are added. The Kruskal-Katona Theorem (page 39, [3]) implies that in fact these p_k must be increasing at a reasonable rate.

Lemma 2 (Corollary to Kruskal-Katona [3]) *For a monotone increasing function f and for $0 \le i < j \le n$, we have $p_i^i \le p_j^j$.*

We break the proof of Theorem 1 into three lemmas. These lemmas, proven below, examine p_s for a particular s. Define s as the smallest number so that at least $1/4$ of the points have size at most s; that is, s is the minimum number so that $\sum_{j=0}^{s} \binom{n}{j} \ge (1/4)2^n$.

Lemma 3 *If $p_s \le 1/4$, then Theorem 1 is true.*

Lemma 4 *If $p_s > 1/4$, then we have $p_{n-s} \ge p_s + 0.4/\sqrt{n}$.*

Lemma 5 *If $p_{n-s} \ge p_s + 0.4/\sqrt{n}$, then Theorem 1 is true.*

Proof of Theorem 1. The above three lemmas immediately imply the theorem. ■

Proof of Lemma 3. Let α denote the fraction of the points x with $\|x\| \le s$ for which $f(x) = 1$. Since the p_k are increasing and $p_s \leq 1/4$, we know that $\alpha \leq 1/4$. Also, by definition of s, we have that at most an $\alpha/4$ fraction of the points $x \in \{0,1\}^n$ satisfy both $\|x\| \leq s$ and $f(x) = 1$.

Because f is fair, this means that a $1/2 - \alpha/4$ fraction of the points $x \in \{0,1\}^n$ have $f(x) = 1$ and $\|x\| > s$. So $(1/2 - \alpha/4)/(3/4) = (2-\alpha)/3$ of the points x where $\|x\| > s$ must have $f(x) = 1$. Because the p_k increase with k, the proportion of points x with $f(x) = 1$ is less in the range $s < \|x\| < n/2$ than in the range $n/2 \leq \|x\| \leq n$. In the range $s < \|x\| < n/2$, then, where $h(x) = 0$, $f(x)$ is also 0 for at least $1 - (2-\alpha)/3 = (1+\alpha)/3$ of the points. In the range $n/2 \leq \|x\| \leq n$, where $h(x) = 1$, $f(x)$ is also 1 for at least $(2-\alpha)/3$ of the points.

The total fraction of points where $f(x)$ agrees with $h(x)$ is at least

$$\frac{1}{4} \cdot (1-\alpha) + \frac{1}{4} \cdot \frac{(1+\alpha)}{3} + \frac{1}{2} \cdot \frac{(2-\alpha)}{3} = \frac{(2-\alpha)}{3}.$$

The first term represents the points with $\|x\| \leq s$; the second the points with $s < \|x\| < n/2$; and the third the points with $\|x\| \geq n/2$. This $(2-\alpha)/3$ fraction is certainly greater than $\frac{1}{2} + \frac{0.1}{\sqrt{n}}$ for sufficiently large n since $\alpha \leq 1/4$. ■

Proof of Lemma 4. Define c so that $s = n/2 - c\sqrt{n}$. A straightforward calculation shows that $c \geq 5/16$. In particular,

$$\begin{aligned}
\sum_{j=\lceil n/2 - \frac{5}{16}\sqrt{n}\rceil}^{\lfloor n/2 \rfloor} \binom{n}{j} &< \left\lceil \frac{5}{16}\sqrt{n} \right\rceil \left\lfloor \binom{n}{n/2} \right\rfloor \\
&\leq \frac{5}{16}\sqrt{n} \left(\frac{0.8}{\sqrt{n}}\right) 2^n \\
&\qquad \text{(by Stirling's approximation)} \\
&= 1/4,
\end{aligned}$$

which implies $c \geq 5/16$ by definition of s.

By Lemma 2, we know that $p_{n-s} \geq p_s^{s/(n-s)}$. Since $s/(n-s) = 1 - 2c\sqrt{n}/(n/2 + c\sqrt{n})$, we have

$$\begin{aligned}
p_{n-s} &\geq p_s \cdot p_s^{-\frac{2c\sqrt{n}}{n/2+c\sqrt{n}}} = p_s \cdot e^{\frac{2c\sqrt{n}}{n/2+c\sqrt{n}} \ln \frac{1}{p_s}} \\
&\geq p_s \left(1 + \frac{2c\sqrt{n}}{n/2 + c\sqrt{n}} \ln \frac{1}{p_s}\right) \\
&\geq p_s + \frac{3.7c}{\sqrt{n}} p_s \ln \frac{1}{p_s}.
\end{aligned}$$

The lemma's hypothesis requires $p_s > 1/4$, and for f to be fair we must have $p_s < 1/2$. So $p_s \ln(1/p_s)$ is at least $(1/2)\ln 2$. This gives us

$$p_{n-s} \geq p_s + \frac{3.7c\ln 2}{2\sqrt{n}} \geq p_s + \frac{0.4}{\sqrt{n}},$$

which is what we want. ■

Proof of Lemma 5. Note that for $i < s$, we have

$$p_{n-i} \geq p_{n-s} \geq p_s + 0.4/\sqrt{n} \geq p_i + 0.4/\sqrt{n},$$

by our hypothesis and the fact that p_k increases with k. Also note that $\sum_{i=0}^{n} p_i \binom{n}{i} = (1/2) \cdot 2^n$, since f is fair. The number of points x where $h(x) = 1$ and $f(x) = 1$ is

$$\begin{aligned}
&\sum_{i=\lceil n/2\rceil}^{n} p_i \binom{n}{i} \\
&= \frac{1}{2}\left(\sum_{i=0}^{s} p_{n-i}\binom{n}{i} + \sum_{i=s+1}^{\lceil n/2\rceil - 1} p_{n-i}\binom{n}{i} \right. \\
&\qquad \left. + \sum_{i=\lceil n/2\rceil}^{n} p_i \binom{n}{i}\right) \\
&\geq \frac{1}{2}\left(\sum_{i=0}^{s}\left(p_i + \frac{0.4}{\sqrt{n}}\right)\binom{n}{i} \right. \\
&\qquad \left. + \sum_{i=s+1}^{\lceil n/2\rceil - 1} p_i \binom{n}{i} + \sum_{i=\lceil n/2\rceil}^{n} p_i \binom{n}{i}\right) \\
&= \frac{1}{4} \cdot 2^n + \frac{0.4}{2\sqrt{n}} \sum_{i=0}^{s} \binom{n}{i} \\
&\geq \frac{1}{4} \cdot 2^n + \frac{0.4}{8\sqrt{n}} \cdot 2^n.
\end{aligned}$$

Since h and f are both fair functions, the number of points x for which $h(x) = 0$ and $f(x) = 1$ is the same as the number of points x for which $h(x) = 1$ and $f(x) = 0$. Therefore, the number of points x where $h(x) = 0$ and $f(x) = 0$ is also at least

$$\frac{1}{4} \cdot 2^n + \frac{0.4}{8\sqrt{n}} \cdot 2^n.$$

Thus the total number of points where $h(x) = f(x)$ is at least

$$\frac{1}{2} \cdot 2^n + \frac{2 \cdot 0.4}{8\sqrt{n}} \cdot 2^n = 2^n \left(\frac{1}{2} + \frac{0.1}{\sqrt{n}}\right).$$

Therefore the majority function has an error of at most $1/2 - 0.1/\sqrt{n}$. ■

3. Learning monotone and unate functions

In this section we show how the previous algorithm for learning fair monotone functions can be extended to the class of general monotone functions and the broader class of "unate" Boolean functions, with essentially the same guarantees. This implies our main positive result, a learning algorithm achieving error $1/2 - \Omega(1/\sqrt{n})$.

First we show how to work around our earlier assumption that n is odd.

Lemma 6 *If f is a fair monotone target function over an even number of variables, then the majority function h has error at most $1/2 - 0.1/\sqrt{n+1}$.*

Proof. For each $(n+1)$-bit vector x', define $f'(x')$ as the value of f on x' with the last bit removed. Since f is fair and monotone, f' is fair and monotone.

Say we have a random n-bit vector x with label $f(x)$. Let x' be x with a random bit appended, and predict $h(x')$. (This is equivalent to the majority function on n variables, with the label chosen randomly if $\|x\| = n/2$.) Since $f'(x') = f(x)$, by Theorem 1, the probability that $h(x') \neq f(x)$ is at most $1/2 - 0.1/\sqrt{n+1}$. ∎

3.1. Learning monotone functions

To transform an algorithm for learning fair monotone functions into an algorithm for learning monotone functions, we do the following. We sample enough times to determine whether the target concept is approximately fair. If it is far enough away from fair then we can output the constant-zero or constant-one function. If not, then we output what the fair-function algorithm does for the concept. The following lemma makes this argument concrete.

Lemma 7 *Say we have an algorithm A for learning fair monotone functions, which uses no samples and outputs a hypothesis with error at most $1/2 - \epsilon$. Then for any $\alpha, \delta > 0$ we can construct an algorithm A' for learning all monotone functions, finding a hypothesis with error at most $1/2 - \epsilon/(2+\alpha)$ with probability $1 - \delta$. This algorithm A' calls $SAMPLE$*

$$\frac{2(2+\alpha)^2}{\alpha^2\epsilon^2} \ln \frac{1}{\delta}$$

times.

Remark. For simplicity we examine a severely handicapped algorithm A which uses no samples and has no chance of failure, since the particular algorithm we are considering has these properties. The theorem also holds for more general algorithms, at the expense of added complexity and slightly worse bounds.

Proof. Say that the target concept f labels a γ fraction of the points with 1. The new algorithm A' uses the samples to obtain an estimate $\tilde{\gamma}$ of γ. By Hoeffding bounds, with probability at least $1-\delta$ our estimate $\tilde{\gamma}$ is within $\gamma \pm \alpha\epsilon/2(2+\alpha)$. Assume that this happens. If $\tilde{\gamma} \geq 1/2 + \epsilon/2$, then A' outputs the constant-one function $h(x) = 1$. Since in this case $\gamma \geq 1/2 + \epsilon/2 - \alpha\epsilon/2(2+\alpha) = 1/2 + \epsilon/(2+\alpha)$, the error is at most $1/2 - \epsilon/(2+\alpha)$. Similarly, if $\tilde{\gamma} \leq 1/2 - \epsilon/2$, then A' outputs the constant-zero function $h(x) = 0$ with error at most $1/2 - \epsilon/(2+\alpha)$.

Otherwise, if $\tilde{\gamma}$ is within $1/2 \pm \epsilon/2$, then γ is within $1/2 \pm (1+\alpha)\epsilon/(2+\alpha)$, and A' outputs whatever A outputs. This hypothesis may err on the points of the fair function, plus it may err on that $(1+\alpha)\epsilon/(2+\alpha)$ fraction of points that must be relabeled in order to make f fair. Thus the error of this hypothesis is at most $1/2 - \epsilon + (1+\alpha)\epsilon/(2+\alpha) = 1/2 - \epsilon/(2+\alpha)$.

The only possibility that leads to an incorrect hypothesis is if we misestimate $\tilde{\gamma}$ by a wide margin. Since this occurs with probability at most δ, we have the required guarantee. ∎

As a corollary we now have our learning algorithm.

Theorem 8 *In polynomial time we can learn monotone functions guaranteeing error at most $1/2 - 0.04/\sqrt{n}$.*

Proof. Let A be the algorithm that performs no sampling or computation and blindly outputs the majority function. By Theorem 1, this algorithm always has error at most $1/2 - 0.1/\sqrt{n}$ on fair monotone target concepts. We apply Lemma 7 with $\alpha = 1/2$ to get our algorithm. ∎

This algorithm is particularly simple: The amount of time it requires in linear in n; it uses only the labels returned by $SAMPLE$ and not the actual bit vectors; and, the algorithm has only three possible outputs (the constant-one function, the constant-zero function, and the majority function).

3.2. Generalizations

Another transformation generalizes the class of functions further to encompass *unate* functions (in some communities, these are called monotone functions). Say that a variable is a *positive indicator* for a Boolean function f if flipping the variable from zero to one never turns f from one to zero, and say that it is a *negative indicator* for f if flipping the variable from one to zero never turns f from one to zero. In monotone functions, all variables are positive indicators; in a unate function, every variable is either a positive indicator or a negative indicator.

If we have an algorithm for learning monotone functions, then we can construct an algorithm for learning unate functions. The technique is similar to that of Lemma 7. In this case, we determine for each variable whether it is a positive or a negative indicator. We then use this information to transform the unate target concept into a concept that is probably a "mostly" monotone function.

All that remains is to show that variables which individually do not exhibit much correlation do not cause much harm if they are wrongly categorized. Since the algorithm miscategorizes variables only if their correlation is very weak, the fraction of points that must be relabeled in order to make the transformed function monotone is very small. By estimating the correlations accurately enough, the fraction becomes so small that with high probability none of the labeled examples that $SAMPLE$ returns in a subsequent draw are points that must be relabeled. Thus the algorithm provides an good estimate to this function. Though it may err on the small fraction that are relabeled, it is also a good estimate to the original function.

The following lemma gives the precise statement of the result; the proof appears in the appendix.

Lemma 9 *Say we have an algorithm A for weakly learning monotone increasing Boolean functions, which uses at most s calls to $SAMPLE$ to output a hypothesis with error at most $1/2 - \epsilon$ with probability $1 - \delta/4$. Then we can construct an algorithm A' for weakly learning unate functions, finding a hypothesis with error at most $1/2 - \epsilon/2$ with probability $1 - \delta$. This algorithm A' calls $SAMPLE$ at most*

$$s + \frac{2}{\hat{\epsilon}^2} \ln \frac{8n}{\delta}$$

times, where $\hat{\epsilon}$ is defined as

$$\hat{\epsilon} = \frac{\min\{\epsilon, \delta/2s\}}{n + \sqrt{2n \ln(4/\delta)}} \,.$$

4. Hardness of learning

We now prove that no algorithm, given only a polynomial number of calls to $SAMPLE$ or $MEMBER$, can achieve correlation more than an $O(\log n)$ factor better than the algorithm of Theorem 8. This proof does not rely on computational hardness; even if the algorithm has infinite computation time, the information available from the oracles does not permit a better correlation.

Theorem 10 *For sufficiently large n, for any $s \geq n$, there exists a distribution P_s over monotone Boolean functions with the following property: For any algorithm A making at most s calls to $MEMBER$, the expected error of A (the probability over P_s, over any internal random choices of A, and over the choice of a random test example x, that A predicts incorrectly on x) is at least $1/2 - O(\log(sn)/\sqrt{n})$. Thus, no algorithm can guarantee expected error $1/2 - \omega((\log n)/\sqrt{n})$ given a polynomial number of queries.*

Notice that given access to $MEMBER$, the $SAMPLE$ oracle is redundant because a (randomized) learning algorithm can simply call $MEMBER$ on uniform random inputs if it so chooses; thus, we need only consider the $MEMBER$ oracle. Also, Theorem 10 is written in terms of expected error, but it can easily be transformed into the (ϵ, δ) formulation.

Proof. We begin by describing the distribution P_s. Given s, let $t = \lg(3sn)$. The target function is a monotone t-DNF formula in which each possible conjunct of t variables is placed in the target *independently* with probability p, where p is defined such that an example of weight $n/2$ (having exactly $n/2$ 1's in it) has probability $1/2$ of being labeled positive. That is, p is the solution to the equation

$$(1-p)^{\binom{n/2}{t}} = 1/2 \,.$$

Note that we have defined P_s so that each term appears independently with some fixed probability, as opposed to the more common distribution on formulas in which the target is random subject to having a fixed number of terms.

To analyze the learning algorithm A, we want to keep the conditional distribution P_s, given the information gathered by A so far, as "clean" as possible. To do this, we augment the $MEMBER$ oracle so that it provides *more* information to the learning algorithm than the standard oracle. Lower bounds for algorithms using the augmented oracle clearly imply at least the same bound for the standard oracle.

Specifically, we define the augmented $MEMBER$ oracle as follows. Given a query example x, with 1's in bit positions indexed by some set S_x, let us imagine that $MEMBER$ looks at all of the $\binom{|S_x|}{t}$ conjuncts of t variables in S_x in lexicographic order and returns the first such conjunct that appears in the target function (if x is positive), or "0" if x is negative. In other words, for a positive example, the oracle returns a witness (the first one in lexicographic order) to the fact that the example is positive. This augmented oracle is convenient because in the conditional distribution P_s given some set of oracle queries, each term is either known to be present in the target formula, is known to be absent from the target formula, or is still in the target formula independently with probability p.

One way to think of this conditional distribution P_s is as a vector V_s of $\binom{n}{t}$ elements, one for each possible conjunct of size t, in which each element of the vector initially contains the number p, indicating the probability that the conjunct is in the target function. When a query x is made, the oracle examines one by one the entries relevant to x (those corresponding to terms that if present in the target function would make x positive). For each entry having value p, we can think of the oracle as flipping a coin, replacing the entry by 0 with probability $1 - p$ and by 1 with probability p. The oracle announces each result to the learning algorithm and halts when either a 1 is observed (meaning the example is positive) or when the number of relevant entries is exhausted (for a negative example).

At any point in the learning process, by definition of the augmented $MEMBER$ oracle, the vector V_s describes exactly the conditional distribution P_s given the information observed by the learning algorithm so far. Specifically, entries in V_s set to 1 correspond to terms known to be present in the target function, entries set to 0 correspond to terms known to be absent from the target function, and the remaining entries are each in the target function independently with probability p.

Claim 1 *After s queries, at most s of the entries in V_s are set to 1.*

Proof. Immediate by definition of the augmented $MEMBER$ oracle. ∎

Claim 2 *After s queries, with probability $1 - e^{-s/4}$, there are at most $2s/p$ zeros in V_s. (Call this event $\mathcal{E}$.)*

Proof. In the worst case, for each query the oracle continues to flip coins until a 1 is produced (in other words, the oracle is not prematurely interrupted by a previously-seen 1, or by the number of relevant entries being exhausted). Thus, the question of the number of zeros produced is equivalent to: How many times will we flip a coin of bias p before seeing s heads? In $2s/p$ coin flips we expect to see $2s$ heads. By Chernoff bounds, the actual number of heads is at least half this quantity with probability at least $1 - e^{-2s/8}$. ∎

For a given example x, and vector V_s, let $V_s(x)$ denote the probability that x is positive given the distribution over target functions defined by V_s. 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, "If $V_s(x) \geq 1/2$ predict positive, else predict negative." We bound the accuracy of this predictor through the following final claim.

Claim 3 *For* any *vector V_s of size $\binom{n}{t}$ 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, we have that with probability at least $1 - 2/\sqrt{n} - 2e^{-c^2}$, the quantity $V_s(x)$ lies within $1/2 \pm (c+1)t/\sqrt{n}$.*

Remark. Notice that by plugging $c = \sqrt{(\ln n)/2}$ into Claim 3 (and using the definition of t) we have that with probability $1 - 4/\sqrt{n}$, $V_s(x)$ lies within $1/2 \pm O(\log^{3/2}(sn)/\sqrt{n})$. This immediately yields a weaker version Theorem 10 in which $\log(sn)/\sqrt{n}$ is replaced by $\log^{3/2}(sn)/\sqrt{n}$. After proving Claim 3 we give a more refined argument producing the stronger bound.

Proof of Claim 3. Let us say that an entry of V_s is "relevant to" an example x if x satisfies the conjunct corresponding to that entry; that is, the entry is relevant to x if the conjunct's presence in the target function implies $f(x) = 1$.

We begin by showing that for a random example x, with probability at least $1 - 2/\sqrt{n} - 2e^{-c^2}$, the following three events occur.

1. None of the 1-entries in V_s are relevant to x.

   There are at most s 1-entries, and for each one, the probability it is relevant to x is 2^{-t}. Since $s2^{-t} = 1/(3n)$ by the definition of t, this event occurs with probability at least $1 - 1/(3n)$.

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

   The expected number of 0-entries relevant to x is at most $(2s/p)2^{-t}$. By Markov's inequality, the chance that it is more than $\sqrt{n}$ times this is at most $1/\sqrt{n}$.

3. The test example x lies in X_k for k within $n/2 \pm c\sqrt{n/2}$.

   By Hoeffding bounds, this event occurs with probability at least $1 - 2e^{-c^2}$.

The probability that all three events occur is at least

$$1 - \frac{1}{3n} - \frac{1}{\sqrt{n}} - 2e^{-c^2} > 1 - \frac{2}{\sqrt{n}} - 2e^{-c^2} .$$

Given that the above three events occur, we now show that $V_s(x)$ lies in the desired range. 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. Thus $V_s(x)$ is at least

$$\begin{aligned}
V_s(x) &\geq 1 - (1-p)^{\left[\binom{n/2 - c\sqrt{n/2}}{t} - \frac{2s\sqrt{n}}{p2^t}\right]} \\
&\geq 1 - \left[(1-p)^{\binom{n/2 - c\sqrt{n/2}}{t}}\right]\left[e^{3s\sqrt{n}/2^t}\right] \\
&= 1 - \left[(1-p)^{\binom{n/2 - c\sqrt{n/2}}{t}}\right]\left[e^{1/\sqrt{n}}\right] \\
&\qquad \text{(definition of } t\text{)} \\
&= 1 - \left[2^{-\binom{n/2 - c\sqrt{n/2}}{t}/\binom{n/2}{t}}\right]\left[e^{1/\sqrt{n}}\right] . \\
&\qquad \text{(definition of } p\text{)}
\end{aligned}$$

We bound the exponent for sufficiently large n:

$$\begin{aligned}
\frac{\binom{n/2 - c\sqrt{n/2}}{t}}{\binom{n/2}{t}} &\geq \left(\frac{n/2 - c\sqrt{n/2} - t}{n/2}\right)^t \\
&\geq \left(\frac{n/2 - (c+1)\sqrt{n/2}}{n/2}\right)^t \\
&= \left(1 - \frac{\sqrt{2}(c+1)}{\sqrt{n}}\right)^t \\
&\geq 1 - \frac{\sqrt{2}(c+1)t}{\sqrt{n}} .
\end{aligned}$$

Thus our lower bound on $V_s(x)$ is

$$\begin{aligned}
V_s(x) &\geq 1 - \left[2^{-(1 - \sqrt{2}(c+1)t/\sqrt{n})}\right]\left[e^{1/\sqrt{n}}\right] \\
&= 1 - \frac{1}{2}\left[e^{\frac{\sqrt{2}\ln(2)(c+1)\ t+1}{\sqrt{n}}}\right] \\
&\geq 1 - \frac{1}{2}\left[1 + \frac{2\sqrt{2}\ln(2)(c+1)t+1}{\sqrt{n}}\right]
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2} - \frac{\sqrt{2}\ln(2)(c+1)t + 1/2}{\sqrt{n}} \\
&\geq \frac{1}{2} - \frac{(c+1)t}{\sqrt{n}} .
\end{aligned}
$$

We maximize $V_s(x)$ when x contains as many 1's as possible and as few 0-entries as possible. Thus $V_s(x)$ is at most

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

We bound the exponent for sufficiently large n:

$$
\begin{aligned}
\frac{\binom{n/2+c\sqrt{n/2}}{t}}{\binom{n/2}{t}} &\leq \left(\frac{n/2 + c\sqrt{n/2}}{n/2 - t}\right)^t \\
&= \left(1 + \frac{c\sqrt{n/2} + t}{n/2 - t}\right)^t \\
&\leq \left(1 + \frac{\sqrt{2}(c+1)}{\sqrt{n}}\right)^t \\
&\leq 1 + \frac{2\sqrt{2}(c+1)t}{\sqrt{n}} .
\end{aligned}
$$

Thus our upper bound on $V_s(x)$ is

$$
\begin{aligned}
V_s(x) &\leq 1 - 2^{-(1 + \frac{2\sqrt{2}(c+1)t}{\sqrt{n}})} \\
&= 1 - \frac{1}{2} e^{-\frac{2\sqrt{2}\ln(2)(c+1)t}{\sqrt{n}}} \\
&\leq 1 - \frac{1}{2}\left[1 - \frac{2\sqrt{2}\ln(2)(c+1)t}{\sqrt{n}}\right] \\
&\leq \frac{1}{2} + \frac{(c+1)t}{\sqrt{n}} .
\end{aligned}
$$

■

Given the above three claims, we now complete the proof of Theorem 10. Claim 2's event $\mathcal{E}$ fails with probability $e^{-s/4}$. Given $\mathcal{E}$, we would like to know the probability that the Bayes-optimal prediction is correct on a random example. Define P_c for $1 \leq c \leq \sqrt{n}$ as the probability for a random example x that $|V_s(x) - 1/2| \leq (c+1)t/\sqrt{n}$. By Claim 3, we know that $1 - 2/\sqrt{n} - 2e^{-c^2} \leq P_c \leq 1$. The probability that the Bayes-optimal prediction is correct for a random example, then, is at most

$$
\begin{aligned}
&P_1\left(\frac{1}{2} + \frac{2t}{\sqrt{n}}\right) \\
&\quad + (P_2 - P_1)\left(\frac{1}{2} + \frac{3t}{\sqrt{n}}\right) \\
&\quad + (P_3 - P_2)\left(\frac{1}{2} + \frac{4t}{\sqrt{n}}\right) \\
&\quad + \cdots \\
&\quad + (P_{\sqrt{n}} - P_{\sqrt{n}-1})\left(\frac{1}{2} + \frac{(\sqrt{n}+1)t}{\sqrt{n}}\right) .
\end{aligned}
$$

By telescoping this series, we get a bound of

$$
\begin{aligned}
&P_{\sqrt{n}}\left(\frac{1}{2} + \frac{(\sqrt{n}+1)t}{\sqrt{n}}\right) - \sum_{c=1}^{\sqrt{n}-1} P_c \frac{t}{\sqrt{n}} \\
&\leq \frac{1}{2} + \frac{t}{\sqrt{n}}\Bigg((\sqrt{n}+1) \\
&\qquad - \sum_{c=1}^{\sqrt{n}-1}\left(1 - \frac{2}{\sqrt{n}} - 2e^{-c^2}\right)\Bigg) \\
&\leq \frac{1}{2} + \frac{t}{\sqrt{n}}\Bigg((\sqrt{n}+1) - (\sqrt{n}-1) \\
&\qquad + \frac{2(\sqrt{n}-1)}{\sqrt{n}} + 2\sum_{c=1}^{\infty} e^{-2c}\Bigg) \\
&= \frac{1}{2} + \frac{4t}{\sqrt{n}} - \frac{2t}{n} + \frac{2t}{\sqrt{n}} \cdot \frac{e^{-2}}{1 - e^{-2}} \\
&= \frac{1}{2} + O\left(\frac{t}{\sqrt{n}}\right) .
\end{aligned}
$$

Thus the best a predictor can do is to achieve a $1/2 + O(t/\sqrt{n})$ probability of agreeing with the target function, given that $\mathcal{E}$ occurs. Since $\mathcal{E}$ fails with $o(t/\sqrt{n})$ probability, and $t = O(\log sn)$, we have the theorem. ■

5. Conclusions

This paper closes to within an $O(\log n)$ factor the question of how well algorithms can learn the class of monotone Boolean functions on the uniform distribution, given a polynomial number of accesses to the target function. It is natural to suppose that one might guarantee an error $1/2 - \Omega((\log n)/\sqrt{n})$ using a more sophisticated algorithm than that of Theorem 8. In particular, the following approach appears promising: First, order the variables by their observed individual correlations with a sufficiently large sample of data. Then, look at the n hypotheses $h_1, \ldots, h_n$ where h_i is the majority function over just the first i variables in this ordering. Finally, choose the h_i of highest observed correlation with the data (or the constant-zero or constant-one hypotheses if the target function is sufficiently non-fair).

The most interesting open question related to this work is that of the learnability of monotone DNF formulas over the uniform distribution, where the algorithm's time and samples used may be polynomial in the number of terms in the formula. The proof of Theorem 10 uses a target concept including $\Theta(sn)$ terms, and so it does not apply directly to this problem. A number of algorithms have been given for special cases of this problem (i.e., when the target function is further restricted to be a special kind of monotone DNF formula) but we know of no positive results better than the guarantee of Theorem 8 for the general case.

References

[1] D. Aldous. On the Markov chain simulation method for uniform combinatorial distributions and simulated annealing. Technical Report 60, Univ California at Berkeley, 1986.

[2] M. Ben-Or and N. Linial. Collective coin flipping, robust voting schemes, and minima of Banzhaf values (preliminary report). In *FOCS*, pages 408–416, 1985.

[3] B. Bollobás. *Combinatorics*. Cambridge University, 1986.

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

[5] J. Kahn, G. Kalai, and N. Linial. The influence of variables on Boolean functions (extended abstract). In *FOCS*, pages 68–80, 1988.

[6] M. Kearns, M. Li, and L. Valiant. Learning Boolean formulas. *JACM*, 41(6):1298–1328, Nov 1994.

[7] R. Schapire. *The Design and Analysis of Efficient Learning Algorithms*. MIT Press, Cambridge MA, 1992.

Appendix

Proof of Lemma 9. For each variable i, our new algorithm A' computes an estimate $\tilde{r}_i$ of the relevance r_i of that variable

$$\begin{aligned} r_i &= \Pr\left[f(x) = 1 \mid x_i = 1\right] - \Pr\left[f(x) = 1 \mid x_i = 0\right] \\ &= 1 - 2\Pr\left[f(x) \neq x_i\right] \ . \end{aligned}$$

For each i, after $(2/\hat{\epsilon}^2)\ln(8n/\delta)$ examples, with probability $\delta/4n$ the estimate of $\Pr\left[f(x) \neq x_i\right]$ is within $\hat{\epsilon}/2$ of the true value. Thus with probability $1-\delta/4$ we estimate all of the $\tilde{r}_i$ within $r_i \pm \hat{\epsilon}$.

When f is a unate function, the ith variable is a positive indicator exactly when $r_i \geq 0$. We define $t : \{0,1\}^n \to \{0,1\}^n$ to transform bit vectors so that $f \circ t$ is a monotone function if we have the correct values for all the $\tilde{r}_i$:

$$t(x)_i = \begin{cases} x_i & \text{if } \tilde{r}_i \geq 0 \\ 1 - x_i & \text{otherwise} \end{cases} .$$

To compute its return value, A' defines for A a new oracle $SAMPLE'$, which works by receiving $(x, f(x))$ from $SAMPLE$ and returning $(t(x), f(x))$. Since $(f\circ t)(t(x)) = f(x)$, $SAMPLE'$ is an oracle to $f \circ t$, and the distribution of its returned vectors is still uniform. So A' can use this oracle to call $A(SAMPLE', \delta/4)$, which returns some hypothesis function h. The return value of A' is $h \circ t$.

Since t is based on the $\tilde{r}_i$, however, $f \circ t$ may not be monotone. In particular, the transformation t may transform variables incorrectly if r_i is within $0 \pm \hat{\epsilon}$. But in this case we can relabel a small fraction of the points to make $f \circ t$ monotone. Let $x \oplus e_i$ denote the bitwise exclusive-OR of x with e_i, the bit vector that is 0 except in the ith position. If t mistransforms the ith variable, we relabel the $|r_i| < \hat{\epsilon}$ fraction of points x where $x_i = 1$, $(f \circ t)(x) = 0$, and $(f \circ t)(x \oplus e_i) = 1$.

With probability $1 - \delta/4$, the number of variables i for which we must do this relabeling is at most $n/2 + \sqrt{(n/2)\ln(4/\delta)}$, because the chance that $\tilde{r}_i$ and r_i have different signs is at most $1/2$. Assuming this occurs, we may need to relabel as much as an $(n/2 + \sqrt{(n/2)\ln(4/\delta)})\hat{\epsilon}$ fraction of the points to make $f \circ t$ monotone. But A' need not compute these to generate $SAMPLE'$: The probability that none of the s examples seen by A fall in these relabeled points is at least

$$\begin{aligned} \left(1 - \left(\frac{n}{2} + \sqrt{\frac{n}{2}\ln\frac{4}{\delta}}\right)\hat{\epsilon}\right)^s &\geq 1 - \left(\frac{n}{2} + \sqrt{\frac{n}{2}\ln\frac{4}{\delta}}\right)\hat{\epsilon}s \\ &\geq 1 - \delta/4 \ . \end{aligned}$$

We assume, then, that A sees a monotone function through $SAMPLE'$.

If A succeeds, its hypothesis h has error at most $1/2 - \epsilon$. This hypothesis may also be wrong on the $(n/2 + \sqrt{(n/2)\ln(4/\delta)})\hat{\epsilon}$ relabeled points, so its error on $f \circ t$ is at most $1/2 - \epsilon + (n/2 + \sqrt{(n/2)\ln(4/\delta)})\hat{\epsilon} \leq 1/2 - \epsilon/2$. This is the error of $h \circ t$ (the hypothesis returned by A') on $f \circ t \circ t = f$.

Four events may occur to prevent A' from returning a hypothesis of error at most $1/2 - \epsilon/2$. One of the estimates of $\tilde{r}_i$ may be outside $r_i \pm \hat{\epsilon}$. The number of variables for which we must do relabeling may exceed $n/2 + \sqrt{(n/2)\ln(4/\delta)}$. One of the samples A sees may be in the relabeled points. Or the h returned by A may have error more than $1/2 - \epsilon$. Each of these occurs with probability at most $\delta/4$, so A' succeeds with probability at least $1 - \delta$. ∎