On Learning Monotone DNF under Product Distributions

Rocco A. Servedio

Division of Engineering and Applied Sciences, Harvard University
Cambridge, MA 02138
rocco@eecs.harvard.edu

Abstract. We show that the class of monotone \(2^c(\sqrt{n \log n})\)-term DNF formulae can be PAC learned in polynomial time under the uniform distribution from random examples only. This is an exponential improvement over the best previous polynomial-time algorithms in this model, which could learn monotone \(o(\log^2 n)\)-term DNF, and is the first efficient algorithm for monotone \((\log n)^{1/3}\)-term DNF in any nontrivial model of learning from random examples. We also show that various classes of small constant-depth circuits which compute monotone functions on few input variables are PAC learnable in polynomial time under the uniform distribution. All of our results extend to learning under any constant-bounded product distribution.

1 Introduction

A disjunctive normal form formula, or DNF, is a disjunction of conjunctions of Boolean literals. The size of a DNF is the number of conjunctions (also known as terms) which it contains. In a seminal 1984 paper [30] Valiant introduced the distribution-free model of Probably Approximately Correct (PAC) learning from random examples and posed the question of whether polynomial-size DNF are PAC learnable in polynomial time. Over the past fifteen years the DNF learning problem has been widely viewed as one of the most important – and challenging – open questions in computational learning theory. This paper substantially improves the best previous results for a well-studied restricted version of the DNF learning problem.

1.1 Previous Work

The lack of progress on Valiant’s original question – are polynomial-size DNF learnable from random examples drawn from an arbitrary distribution in polynomial time? – has led many researchers to study restricted versions of the DNF learning problem. As detailed below, the restrictions which have been considered include

- allowing the learner to make membership queries for the value of the target function at points selected by the learner;
— requiring that the learner succeed only under restricted distributions on examples, such as the uniform distribution, rather than all distributions;
— requiring that the learner succeed only for restricted subclasses of DNF formulae such as monotone DNF with a bounded number of terms.

A SAT-$k$ DNF is a DNF in which each truth assignment satisfies at most $k$ terms. Khardon [22] gave a polynomial time membership query algorithm for learning polynomial-size SAT-1 DNF under the uniform distribution; this result was later strengthened by Blum et al. [4] to SAT-$k$ DNF for any constant $k$. Bellare [6] gave a polynomial time membership query algorithm for learning $O(\log n)$-term DNF under the uniform distribution. This result was strengthened by Blum and Rudich [7] who gave a polynomial time algorithm for exact learning $O(\log n)$-term DNF using membership and equivalence queries; several other polynomial-time algorithms for $O(\log n)$-term DNF have since been given in this model [3, 9, 10, 25]. Mansour [27] gave a $n^{O(\log \log n)}$-time membership query algorithm which learns polynomial-size DNF under the uniform distribution. In a celebrated result, Jackson [18] gave a polynomial-time membership query algorithm for learning polynomial-size DNF under constant-bounded product distributions. His algorithm, the efficiency of which was subsequently improved by several authors [11, 23], is the only known polynomial time algorithm for learning the unrestricted class of polynomial size DNF in any learning model.

In the standard PAC model without membership queries positive results are known for various subclasses of DNF under restricted distributions. A read-$k$ DNF is one in which each variable appears at most $k$ times. Kearns et al. [20, 21] showed that read-once DNF are PAC learnable under the uniform distribution in polynomial time. Hancock [15] extended this result to read-$k$ DNF for any constant $k$. Verbeuert [31] gave an algorithm for learning arbitrary polynomial-size DNF under the uniform distribution in time $n^{O(\log n)}$, and Linial et al. [26] gave an algorithm for learning any $AC^0$ circuit (constant depth, polynomial size, unbounded fanin AND/OR gates) under the uniform distribution in $n^{\text{poly}(\log n)}$ time.

A monotone DNF is a DNF with no negated variables. It is well known that in the distribution-independent setting, learning monotone DNF is equivalent to learning general DNF [20]. However this equivalence does not hold for restricted distributions such as the uniform distribution, and many researchers have studied the problem of learning monotone DNF under restricted distributions. Hancock and Mansour [16] gave a polynomial time algorithm for learning monotone read-$k$ DNF under constant-bounded product distributions. Verbeuert [32] gave a polynomial time uniform distribution algorithm for learning poly-disjoint one-read-once monotone DNF and read-once factorable monotone DNF. Kucera et al. [24] gave a polynomial-time algorithm which learns monotone $k$-term DNF under the uniform distribution using hypotheses which are monotone $k$-term DNF. This was improved by Sakai and Maruoka [29] who gave a polynomial-time algorithm for learning monotone $O(\log n)$-term DNF under the uniform distribution using hypotheses which are monotone $O(\log n)$-term DNF. In [9] Bshouty gave a polynomial-time uniform-distribution algorithm for learning a
class which includes monotone $O(\log n)$-term DNF. Later Bshouty and Tamon [12] gave a polynomial-time algorithm for learning a class which includes monotone $O(\log^2 n/(\log \log n)^3)$-term DNF under constant-bounded product distributions.

1.2 Our Results

We give an algorithm for learning monotone DNF under the uniform distribution. If the desired accuracy level $\epsilon$ is constant as a function of $n$ (the number of variables), then the algorithm learns $2^{O(\sqrt{\log n})}$-term monotone DNF over $n$ variables in $\text{poly}(n)$ time. (We note that the algorithm of [12] for learning monotone DNF with $O((\log n)^2/(\log \log n)^3)$ terms also requires that $\epsilon$ be constant in order to achieve $\text{poly}(n)$ runtime.) This is the first polynomial time algorithm which uses only random examples and successfully learns monotone DNF with more than a polylogarithmic number of terms. We also show that essentially the same algorithm learns various classes of small constant-depth circuits which compute monotone functions on few variables. All of our results extend to learning under any constant-bounded product distribution.

Our algorithm combines ideas from Linial et al.'s influential paper [26] on learning $AC^0$ functions using the Fourier transform and Bshouty and Tamon’s paper [12] on learning monotone functions using the Fourier transform. By analyzing the Fourier transform of $AC^0$ functions, Linial et al. showed that almost all of the Fourier “power spectrum” of any $AC^0$ function is contained in “low” Fourier coefficients, i.e. coefficients which correspond to small subsets of variables. Their learning algorithm estimates each low Fourier coefficient by sampling and constructs an approximation to $f$ using these estimated Fourier coefficients. If $c$ is the size bound for low Fourier coefficients, then since there are $\binom{n}{c}$ Fourier coefficients corresponding to subsets of $c$ variables the algorithm requires roughly $n^c$ time steps. Linial et al. showed that for $AC^0$ circuits $c$ is essentially $\text{poly}(\log n)$; this result was later sharpened for DNF formulae by Mansour [27].

Our algorithm extends this approach in the following way: Let $C \subseteq AC^0$ be a class of Boolean functions which we would like to learn. Suppose that $C$ has the following properties:

1. For every $f \in C$ there is a set $S_f$ of “important” variables such that almost all of the power spectrum of $f$ is contained in Fourier coefficients corresponding to subsets of $S_f$.
2. There is an efficient algorithm which identifies the set $S_f$ from random examples.

(Such an algorithm, which we give in Section 3.1, is implicit in [12] and requires only that $f$ be monotone.) We can learn an unknown function $f$ from such a class $C$ by first identifying the set $S_f$, then estimating the low Fourier coefficients which correspond to small subsets of $S_f$ and using these estimates to construct an approximation to $f$. To see why this works, note that since $f$ is in $AC^0$ almost all of the power spectrum of $f$ is in the low Fourier coefficients; moreover, property
(1) implies that almost all of the power spectrum of $f$ is in the Fourier coefficients which correspond to subsets of $S_f$. Consequently it must be the case that almost all of the power spectrum of $f$ is in low Fourier coefficients which correspond to subsets of $S_f$. Thus in our setting we need only estimate the $(|S_f|)$ Fourier coefficients which correspond to “small” subsets of variables in $S_f$. If $|S_f| \ll n$ then this is much more efficient than estimating all $\binom{n}{2}$ low Fourier coefficients.

In Section 2 we formally define the learning model and give some necessary facts about Fourier analysis over the Boolean cube. In Section 3 we give our learning algorithm for the uniform distribution, and in Section 4 we describe how the algorithm can be modified to work under any constant-bounded product distribution.

2 Preliminaries

We write $[n]$ to denote the set $\{1, \ldots, n\}$ and use capital letters for subsets of $[n]$. We write $|A|$ to denote the number of elements in $A$. Barred lowercase letters denote bitstrings, i.e. $\overline{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$. In this paper Boolean circuits are composed of AND/OR/NOT gates where AND and OR gates have unbounded fanin and negations occur only on inputs. We view Boolean functions on $n$ variables as real valued functions which map $\{0,1\}^n$ to $\{-1, 1\}$. A Boolean function $f : \{0,1\}^n \rightarrow \{-1, 1\}$ is monotone if changing the value of an input bit from 0 to 1 never causes the value of $f$ to change from 1 to $-1$.

If $D$ is a distribution and $f$ is a Boolean function on $\{0,1\}^n$, then as in [12, 16] we say that the influence of $x_i$ on $f$ with respect to $D$ is the probability that $f(\overline{x})$ differs from $f(\overline{y})$, where $\overline{y}$ is $\overline{x}$ with the $i$-th bit flipped and $\overline{x}$ is drawn from $D$. For ease of notation let $f_{i,0}$ denote the function obtained from $f$ by fixing $x_i$ to 0 and let $f_{i,1}$ be defined similarly. We thus have

$$I_{D,i}(f) = \Pr_D[f_{i,0}(\overline{x}) \neq f_{i,1}(\overline{x})] = \frac{1}{2}E_D[|f_{i,1} - f_{i,0}|].$$

For monotone $f$ this can be further simplified to

$$I_{D,i}(f) = \frac{1}{2}E_D[f_{i,1} - f_{i,0}] = \frac{1}{2}(E_D[f_{i,1}] - E_D[f_{i,0}]).$$

We frequently use Chernoff bounds on sums of independent random variables [14]:

**Theorem 1.** Let $x_1, \ldots, x_m$ be independent identically distributed random variables with $E[x_i] = p$, $|x_i| \leq B$, and let $s_m = x_1 + \cdots + x_m$. Then

$$m \geq \frac{2B^2}{\epsilon^2} \ln \frac{2}{\delta} \quad \text{implies that} \quad \Pr\left[\left|\frac{s_m}{m} - p\right| > \epsilon\right] \leq \delta.$$
2.1 The Learning Model

Our learning model is a distribution-specific version of Valiant’s Probably Approximately Correct (PAC) model [30] which has been studied by many researchers, e.g. [4, 6, 11–13, 16, 18, 22, 24, 26, 27, 31, 32]. Let $C$ be a class of Boolean functions over $\{0, 1\}^n$, let $D$ be a probability distribution over $\{0, 1\}^n$, and let $f \in C$ be an unknown target function. A learning algorithm $A$ for $C$ takes as input an accuracy parameter $0 < \epsilon < 1$ and a confidence parameter $0 < \delta < 1$. During its execution the algorithm has access to an example oracle $E_X(f, D)$ which, when queried, generates a random labeled example $(\overline{x}, f(\overline{x}))$ where $\overline{x}$ is drawn according to $D$. The learning algorithm outputs a hypothesis $h$ which is a Boolean function over $\{0, 1\}^n$; the error of this hypothesis is defined to be $\text{error}(h, f) = \Pr_D[h(\overline{x}) \neq f(\overline{x})]$. We say that $A$ learns $C$ under $D$ if for every $f \in C$ and $0 < \epsilon, \delta < 1$, with probability at least $1 - \delta$ algorithm $A$ outputs a hypothesis $h$ which has error $\text{error}(h, f) \leq \epsilon$.

2.2 The Discrete Fourier Transform

Let $U$ denote the uniform distribution over $\{0, 1\}^n$. The set of all real valued functions on $\{0, 1\}^n$ may be viewed as a $2^n$-dimensional vector space with inner product defined as

$$\langle f, g \rangle = 2^{-n} \sum_{\overline{x} \in \{0, 1\}^n} f(\overline{x})g(\overline{x}) = E_U[f g]$$

and norm defined as $\|f\| = \sqrt{\langle f, f \rangle}$. Given any subset $A \subseteq [n]$, the Fourier basis function $\chi_A : \{0, 1\}^n \rightarrow \{-1, 1\}$ is defined by $\chi_A(\overline{x}) = (-1)^{\sum_{i \in A} x_i}$, where $X$ is the subset of $[n]$ defined by $i \in X$ iff $x_i = 1$. It is well known that the $2^n$ basis functions $\chi_A$ form an orthonormal basis for the vector space of real valued functions on $\{0, 1\}^n$; we refer to this basis as the $\chi$ basis. In particular, any function $f$ can be uniquely expressed as $f(\overline{x}) = \sum_A \hat{f}(A)\chi_A(\overline{x})$, where the values $\hat{f}(A)$ are known as the Fourier coefficients of $f$ with respect to the $\chi$ basis.

Since the functions $\chi_A$ form an orthonormal basis, the value of $\hat{f}(A)$ is $\langle f, \chi_A \rangle$; also by linearity we have that $f(\overline{x}) + g(\overline{x}) = \sum_A (\hat{f}(A) + \hat{g}(A))\chi_A(\overline{x})$. Another easy consequence of orthonormality is Parseval’s identity

$$E_U[f^2] = \|f\|^2 = \sum_{A \subseteq [n]} |\hat{f}(A)|^2.$$

If $f$ is a Boolean function then this value is exactly 1. Finally, for any Boolean function $f$ and real-valued function $g$ we have [12, 26]

$$\Pr_U[f \neq \text{sign}(g)] \leq E_U[(f - g)^2]$$

where $\text{sign}(z)$ takes value 1 if $z \geq 0$ and takes value $-1$ if $z < 0$. 

(2)
3 Learning under Uniform Distributions

3.1 Identifying Relevant Variables

The following lemma, which is implicit in [12], gives an efficient algorithm for identifying the important variables of a monotone Boolean function. We refer to this algorithm as FindVariables.

**Lemma 1.** Let $f : \{0,1\}^n \to \{\pm 1\}$ be a monotone Boolean function. There is an algorithm which has access to $EX(f,\mathcal{U})$, runs in $\text{poly}(n,1/\epsilon,\log 1/\delta)$ time steps for all $\epsilon, \delta > 0$, and with probability at least $1 - \delta$ outputs a set $S_f \subseteq [n]$ such that

$$i \in S_f \implies \sum_{A : i \in A} \hat{f}(A)^2 \geq \epsilon/2 \quad \text{and} \quad i \notin S_f \implies \sum_{A : i \in A} \hat{f}(A)^2 \leq \epsilon.$$  

**Proof.** Kahn et al. ([19] Section 3) have shown that

$$I_{\mathcal{U},i}(f) = \sum_{A \ni i \subseteq A} \hat{f}(A)^2.$$  

(3)

To prove the lemma it thus suffices to show that $I_{\mathcal{U},i}(f)$ can be estimated to within accuracy $\epsilon/4$ with high probability. By Equation (1) from Section 2 this can be done by estimating $E_{\mathcal{U}}[f_{i,1}]$ and $E_{\mathcal{U}}[f_{i,0}]$. Two applications of Chernoff bounds finish the proof: the first is to verify that with high probability a large sample drawn from $EX(f,\mathcal{U})$ contains many labeled examples which have $x_i = 1$ and many which have $x_i = 0$, and the second is to verify that a collection of many labeled examples with $x_i = b$ with high probability yields an accurate estimate of $E_{\mathcal{U}}[f_{i,b}]$. \hfill \Box

3.2 The Learning Algorithm

Our learning algorithm, which we call LearnMonotone, is given below:

- Use FindVariables to identify a set $S_f$ of important variables.
- Draw $m$ labeled examples $(\overline{x^1}, f(\overline{x^1})), \ldots, (\overline{x^m}, f(\overline{x^m}))$ from $EX(f,\mathcal{U})$. For every $A \subseteq S_f$ with $|A| \leq c$ set $\alpha_A = \frac{1}{m} \sum_{i=1}^{m} f(\overline{x^i})\chi_A(\overline{x^i})$. For every $A$ such that $|A| > c$ or $A \not\subseteq S_f$ set $\alpha_A = 0$.
- Output the hypothesis $\text{sign}(g(\overline{x}))$, where $g(\overline{x}) = \sum_A \alpha_A \chi_A(\overline{x})$.

The algorithm thus estimates $\hat{f}(A)$ for $A \subseteq S_f, |A| \leq c$ by sampling and constructs a hypothesis using these approximate Fourier coefficients. The values of $m$ and $c$ and the parameter settings for FindVariables are specified below.
3.3 Learning Monotone $2^{O(\sqrt{\log n})}$-term DNF

Let $f : \{0, 1\}^n \rightarrow \{-1, 1\}$ be a monotone $t$-term DNF. The proof that algorithm LearnMonotone learns $f$ uses a DNF called $f_1$ to show that FindVariables identifies a small set of variables $S_f$ and uses another DNF called $f_2$ to show that $f$ can be approximated by approximating Fourier coefficients which correspond to small subsets of $S_f$.

Let $f_1$ be the DNF which is obtained from $f$ by removing every term which contains more than $\log \frac{32tn}{\epsilon}$ variables. Since there are at most $t$ such terms each of which is satisfied by a random example with probability less than $\epsilon/32tn$, we have $\Pr_{\mathcal{U}}[\hat{f}(\mathcal{X}) \neq \hat{f}_1(\mathcal{X})] < \frac{\epsilon}{32tn}$ (this type of argument was first used by Verbeurgt [31]). Let $R \subseteq [n]$ be the set of variables which $f_1$ depends on; it is clear that $|R| \leq t \log \frac{32tn}{\epsilon}$. Moreover, since $I_{i \in R}$, equation (3) from Section 3.1 implies that $\hat{f}_1(A) = 0$ for $A \not\subseteq R$.

Since $f$ and $f_1$ are Boolean functions, $f - f_1$ is either 0 or 2, so $E_{\mathcal{U}}[(f - f_1)^2] = 4 \Pr_{\mathcal{U}}[f \neq f_1] < \epsilon/8n$. By Parseval’s identity we have

$$E_{\mathcal{U}}[(f - f_1)^2] = \sum_{A} (\hat{f}(A) - \hat{f}_1(A))^2$$

$$= \sum_{A \subseteq R} (\hat{f}(A) - \hat{f}_1(A))^2 + \sum_{A \not\subseteq R} (\hat{f}(A) - \hat{f}_1(A))^2$$

$$= \sum_{A \subseteq R} (\hat{f}(A) - \hat{f}_1(A))^2 + \sum_{A \not\subseteq R} (\hat{f}(A))^2$$

$$< \epsilon/8n.$$

Thus $\sum_{A \not\subseteq R} \hat{f}(A)^2 < \frac{\epsilon}{8n}$, and consequently we have

$$i \not\subseteq R \text{ implies } \sum_{A \ni i \not\subseteq A} \hat{f}(A)^2 < \frac{\epsilon}{8n}. \quad (4)$$

We set the parameters of FindVariables so that with high probability

$$i \in S_f \text{ implies } \sum_{A \ni i \in A} \hat{f}(A)^2 \geq \frac{\epsilon}{8n} \quad (5)$$

$$i \not\in S_f \text{ implies } \sum_{A \ni i \not\subseteq A} \hat{f}(A)^2 \leq \frac{\epsilon}{4n}. \quad (6)$$

Inequalities (4) and (5) imply that $S_f \subseteq R$, so $|S_f| \leq t \log \frac{32tn}{\epsilon}$. Furthermore, since $A \not\subseteq S_f$ implies $i \not\in A$ for some $i \not\in S_f$, inequality (6) implies

$$\sum_{A \not\subseteq S_f} \hat{f}(A)^2 \leq \frac{\epsilon}{4}. \quad (7)$$

The following lemma is due to Mansour ([27] Lemma 3.2):
Lemma 2 (Mansour). Let $f$ be a DNF with terms of size at most $d$. Then for all $\epsilon > 0$ 
\[ \sum_{|A| > 10d \log(2/\epsilon)} \hat{f}(A)^2 \leq \epsilon/2. \]

One approach at this point is to use Mansour's lemma to approximate $f$ by approximating the Fourier coefficients of all subsets of $S_f$ which are smaller than $20d \log(2/\epsilon)$, where $d = \log \frac{2m}{d}$ is the maximum size of any term in $f_1$. However, this approach does not give a good overall running time because $d$ is too large. Instead we consider another DNF with smaller terms than $f_1$ which also closely approximates $f$. By using this stronger bound on term size in Mansour's lemma we get a better final result.

More precisely, let $f_2$ be the DNF obtained from $f$ by removing every term which contains at least $\log \frac{2m}{\epsilon}$ variables. Let $c = 20 \log \frac{12m}{\epsilon} \log \frac{2}{\epsilon}$. Mansour’s lemma implies that
\[ \sum_{|A| > c} \hat{f}_2(A)^2 \leq \epsilon/8. \] (8)

Moreover, we have $\Pr_{\mathcal{U}}[f \neq f_2] \leq \epsilon/32$ and hence
\[ 4 \Pr_{\mathcal{U}}[f \neq f_2] = \mathbb{E}_{\mathcal{U}}[(f - f_2)^2] = \sum_A (\hat{f}(A) - \hat{f}_2(A))^2 \leq \epsilon/8. \] (9)

Let $\alpha_A$ and $g(x)$ be as defined in LearnMonotone. Using inequality (2) from Section 2.2, we have
\[ \Pr[\text{sign}(g) \neq f] \leq \mathbb{E}_{\mathcal{U}}[(g - f)^2] = \sum_A (\alpha_A - \hat{f}(A))^2 = X + Y + Z, \]
where
\[ X = \sum_{|A| \leq c, A \notin S_f} (\alpha_A - \hat{f}(A))^2, \]
\[ Y = \sum_{|A| > c} (\alpha_A - \hat{f}(A))^2, \]
\[ Z = \sum_{|A| \leq c, A \in S_f} (\alpha_A - \hat{f}(A))^2. \]

To bound $X$, we observe that $\alpha_A = 0$ for $A \notin S_f$, so by (7) we have
\[ X = \sum_{|A| \leq c, A \notin S_f} \hat{f}(A)^2 = \sum_{A \notin S_f} \hat{f}(A)^2 \leq \epsilon/4. \]
To bound $Y$, we note that $\alpha_A = 0$ for $|A| > c$ and hence $Y = \sum_{|A| > c} \hat{f}(A)^2$. Since $\hat{f}(A)^2 \leq 2(\hat{f}(A) - \hat{f}_2(A))^2 + 2\hat{f}_2(A)^2$, we have
\[ Y \leq 2 \sum_{|A| > c} (\hat{f}(A) - \hat{f}_2(A))^2 + 2 \sum_{|A| > c} \hat{f}_2(A)^2 \]
\[ \leq 2 \sum_{A} (\hat{f}(A) - \tilde{f}_i(A))^2 + \epsilon/4 \]
\[ \leq \epsilon/2 \]

by inequalities (8) and (9) respectively.

It remains to bound \( Z = \sum_{|A| \leq c, \alpha_A \leq S_f} (\alpha_A - \hat{f}(A))^2 \). As in Linial et al. [26] this sum can be made less than \( \epsilon/4 \) by taking \( m \) sufficiently large so that with high probability each estimate \( \alpha_A \) differs from the true value \( \hat{f}(A) \) by at most \( \sqrt{\epsilon/4}|S_f| \). A straightforward Chernoff bound argument shows that taking \( m = \text{poly}(|S_f|^c, 1/\epsilon, \log(1/\delta)) \) suffices.

Thus, we have \( X + Y + Z \leq \epsilon \). Recalling our bounds on \( |S_f| \) and \( c \), we have proved:

**Theorem 2.** Under the uniform distribution, for any \( \epsilon, \delta > 0 \), the algorithm \textbf{LearnMonotone} can be used to learn \( t \)-term monotone DNF in time polynomial in \( n \), \( (t \log \frac{12}{\epsilon}) \frac{\log \frac{12}{\epsilon}}{2 \log \frac{12}{\epsilon}} \) and \( \log(1/\delta) \).

Taking \( t = 2^{O(\sqrt{\log n})} \) we obtain the following corollary:

**Corollary 1.** For any constant \( \epsilon \) algorithm \textbf{LearnMonotone} learns \( 2^{O(\sqrt{\log n})} \)-term monotone DNF in \( \text{poly}(n, \log(1/\delta)) \) time under the uniform distribution.

As noted earlier, Bshouty and Tamon’s algorithm [12] for learning monotone DNF with \( O((\log n)^2/\log \log n)^3 \) terms also requires that \( \epsilon \) be constant in order to achieve \( \text{poly}(n) \) runtime.

### 3.4 Learning Small Constant-Depth Monotone Circuits on Few Variables

Let \( C \) be the class of depth \( d \), size \( M \) circuits which compute monotone functions on \( r \) out of \( n \) variables. An analysis similar to that of the last section (but simpler since we do not need to introduce auxiliary functions \( f_1 \) and \( f_2 \)) shows that algorithm \textbf{LearnMonotone} can be used to learn \( C \). As in the last section the \textbf{FindVariables} procedure is used to identify the “important” relevant variables, of which there are now at most \( r \). Instead of using Mansour’s lemma, we use the main lemma of Linial et al. [26] to bound the total weight of high-order Fourier coefficients for constant-depth circuits:

**Lemma 3 (Linial et al.).** Let \( f \) be a Boolean function computed by a circuit of depth \( d \) and size \( M \) and let \( c \) be any integer. Then

\[ \sum_{|A| = c} \hat{f}(A)^2 \leq 2M2^{-c^{d/2}/20} \]

Taking \( m = \text{poly}(r^c, 1/\epsilon, \log(1/\delta)) \) and \( c = \Theta((\log(M/\epsilon))^d) \) in \textbf{LearnMonotone} we obtain:
Theorem 3. Fix $d \geq 1$ and let $C_{d,M,r}$ be the class of depth $d$, size $M$ circuits which compute monotone functions on $r$ out of $n$ variables. Under the uniform distribution, for any $c, \delta > 0$, algorithm LearnMonotone learns class $C_{d,M,r}$ in time polynomial in $n$, $r^{\log(\log M/c)} d^d$ and $\log(1/\delta)$.

One interesting corollary is the following:

Corollary 2. Fix $d \geq 1$ and let $C_d$ be the class of depth $d$, size $2^{O((\log n)^{1/(d+1)})}$ circuits which compute monotone functions on $2^{O((\log n)^{1/(d+1)})}$ variables. Then for any constant $\epsilon$ algorithm LearnMonotone learns class $C_d$ in $\text{poly}(n, \log(1/\delta))$ time.

While this class $C_d$ is rather limited from the perspective of Boolean circuit complexity, from a learning theory perspective it is fairly rich. We note that $C_d$ strictly includes the class of depth $d$, size $2^{O((\log n)^{1/(d+1)})}$ circuits on $2^{O((\log n)^{1/(d+1)})}$ variables which contain only unbounded fanin AND and OR gates. This follows from results of Okolnischev [28] and Ajtai and Gurevich [1] (see also [8] Section 3.6) which show that there are monotone functions which can be computed by $AC^0$ circuits but are not computable by $AC^0$ circuits which have no negations.

4 Product Distributions

A product distribution over $\{0,1\}^n$ is characterized by parameters $\mu_1, \ldots, \mu_n$ where $\mu_i = \Pr[x_i = 1]$. Such a distribution $D$ assigns values independently to each variable, so for $\pi \in \{0,1\}^n$ we have $D(\pi) = \left(\prod_{i=1}^{n} \mu_i \right) \left(\prod_{i=0}^{n} (1 - \mu_i) \right)$. The uniform distribution is a product distribution with each $\mu_i = 1/2$. The standard deviation of $x_i$ under a product distribution is $\sigma_i = \sqrt{\mu_i (1 - \mu_i)}$. A product distribution $D$ is constant-bounded if there is some constant $c \in (0,1)$ independent of $n$ such that $\mu_i \in [c, 1 - c]$ for all $i = 1, \ldots, n$. We let $\beta$ denote $\max_{i=1,\ldots,n}(1/\mu_i, 1/(1 - \mu_i))$. Throughout the rest of this paper $D$ denotes a product distribution.

Given a product distribution $D$ we define a new inner product over the vector space of real valued functions on $\{0,1\}^n$ as

$$\langle f, g \rangle_D = \sum_{\pi \in \{0,1\}^n} D(\pi) f(\pi) g(\pi) = E_D[fg]$$

and a corresponding norm $\|f\|_D = \sqrt{\langle f, f \rangle_D}$. We refer to this norm as the $D$-norm. For $i = 1, \ldots, n$ let $z_i = (x_i - \mu_i)/\sigma_i$. Given $A \subseteq [n]$, let $\phi_A$ be defined as $\phi_A(\pi) = \prod_{i \in A} z_i$. As noted by Bahadur [5] and Furst et al. [13], the $2^n$ functions $\phi_A$ form and orthonormal basis for the vector space of real valued functions on $\{0,1\}^n$ with respect to the $D$-norm, i.e. $\langle \phi_A, \phi_B \rangle_D$ is 1 if $A = B$ and is 0 otherwise We refer to this basis as the $\phi$ basis. The following fact is useful:
**Fact 1 (Bahadur; Furst et. al)** The $\phi$ basis is the basis which would be obtained by Gram-Schmidt orthonormalization (with respect to the $D$-norm) of the $\chi$ basis performed in order of increasing $|A|$.

By the orthonormality of the $\phi$ basis, any real function on $\{0, 1\}^n$ can be uniquely expressed as $f(\mathbf{x}) = \sum_A \hat{f}(A) \phi_A(\mathbf{x})$ where $\hat{f}(A) = \langle f, \phi_A \rangle_D$ is the Fourier coefficient of $A$ with respect to the $\phi$ basis. Note that we write $\hat{f}(A)$ for the $\phi$ basis Fourier coefficient and $\hat{f}(A)$ for the $\chi$ basis Fourier coefficient. Also by orthonormality we have Parseval’s identity

$$E_D[f^2] = \|f\|_D^2 = \sum_{A \in \mathcal{X}} \hat{f}(A)^2$$

which is 1 for Boolean $f$. Finally, for Boolean $f$ and real-valued $g$ we have ([13] Lemma 10)

$$P_D[f \neq \text{sign}(g)] \leq E_D[(f - g)^2]. \quad (10)$$

Furst et al. [13] analyzed the $\phi$ basis Fourier spectrum of $AC^0$ functions and gave product distribution analogues of Linial et al’s results on learning $AC^0$ circuits under the uniform distribution. In Section 4.1 we sharpen and extend some results from [13], and in Section 5 we use these sharpened results together with techniques from [13] to obtain product distribution analogues of our algorithms from Section 3.

### 4.1 Some $\phi$ Basis Fourier Lemmas

A random restriction $\rho_{p, D}$ is a mapping from $\{x_1, \ldots, x_n\}$ to $\{0, 1, \ast\}$ where $x_i$ is mapped to $\ast$ with probability $p$, to 1 with probability $(1-p)\mu_i$, and to 0 with probability $(1-p)(1-\mu_i)$. If $f$ is a Boolean function then $f[p]$ represents the function $f(\rho_{p, D}(\mathbf{x}))$ whose variables are those $x_i$ which are mapped to $\ast$ and whose other $x_i$ are instantiated as 0 or 1 according to $\rho_{p, D}$.

The following is a variant of Håstad’s well known switching lemma [17]:

**Lemma 4.** Let $D$ be a product distribution with parameters $\mu_i$ and $\beta$ as defined above, let $f$ be a CNF formula where each clause has at most $d$ literals, and let $\rho_{p, D}$ be a random restriction. Then with probability at least $1 - (4\beta^d p)^d$, the following hold:

1. the function $f[p]$ can be expressed as a DNF formula where each term has at most $s$ literals;
2. the terms of such a DNF all accept disjoint sets of inputs.

**Proof sketch:** The proof is a minor modification of arguments given in Section 4 of [2].

The following corollary is a product distribution analogue of ([26] Corollary 1):
Corollary 3. Let \( D \) be a product distribution with parameters \( \mu \) and \( \beta \), let \( f \) be a CNF formula where each clause has at most \( d \) literals, and let \( \rho_{p,D} \) be a random restriction. Then with probability at least \( 1 - (4\beta pd)^s \) we have that \( \tilde{f}(\rho(A)) = 0 \) for all \( |A| > s \).

Proof. Linial et al. [26] show that if \( f[\rho \) satisfies properties (1) and (2) of Lemma 4 then \( \tilde{f}(\rho(A)) = 0 \) for all \( |A| > s \). Hence such a \( f[\rho \) is in the space spanned by \( \{X_A : |A| \leq s \} \). By Fact 1 and the nature of Gram-Schmidt orthonormalization, this is the same space which is spanned by \( \{\phi_A : |A| \leq s \} \), and the corollary follows.

Corollary 3 is a sharpened version of a similar lemma, implicit in [13], which states that under the same conditions with probability at least \( 1 - (5\beta pd/2)^s \) we have \( \tilde{f}(\rho(A)) = 0 \) for all \( |A| > s^2 \). Armed with the sharper Corollary 3, using arguments from [13] it is straightforward to prove

Lemma 5. For any Boolean function \( f \), for any integer \( t \),

\[
\sum_{|A| > t} \tilde{f}(A)^2 \leq 2\Pr_{\rho_{p,D}}[\tilde{f}(\rho(A)) \neq 0 \text{ for some } |A| > tp/2].
\]

Boolean duality implies that the conclusion of Corollary 3 also holds if \( f \) is a DNF with each term of length at most \( d \). Taking \( p = 1/8\beta d \) and \( s = \log \frac{4}{\epsilon} \) in this DNF version of Corollary 3 and \( t = 16\beta d \log \frac{4}{\epsilon} \) in Lemma 5, we obtain the following analogue of Mansour’s lemma (Lemma 2) for the \( \phi \) basis:

Lemma 6. Let \( f \) be a DNF with terms of size at most \( d \). Then for all \( \epsilon > 0 \)

\[
\sum_{|A| > 16\beta d \log(4/\epsilon)} \tilde{f}(A)^2 \leq \epsilon/2.
\]

Again using arguments from [13], Corollary 3 can also be used to prove the following version of the main lemma from [13]:

Lemma 7. Let \( f \) be a Boolean function computed by a circuit of depth \( d \) and size \( M \) and let \( c \) be any integer. Then

\[
\sum_{|A| > \epsilon} \tilde{f}(A)^2 \leq 2M2^{-c^{1/4}/8\beta}.
\]

The version of this lemma given in [13] has \( 1/(d + 2) \) instead of \( 1/d \) in the exponent of \( c \). This new tighter bound will enable us to give stronger guarantees on our learning algorithm’s performance under product distributions than we could have obtained by simply using the lemma from [13].
5 Learning under Product Distributions

5.1 Identifying Relevant Variables

We have the following analogue to Lemma 2 for product distributions:

**Lemma 8.** Let \( f : \{0, 1\}^n \rightarrow \{-1, 1\} \) be a monotone Boolean function. There is an algorithm which has access to \( EX(f, \mathcal{D}) \), runs in \( \text{poly}(n, \beta, 1/\epsilon, \log 1/\delta) \) time steps for all \( \epsilon, \delta > 0 \), and with probability at least \( 1 - \delta \) outputs a set \( S_f \subseteq [n] \) such that

\[
i \in S_f \text{ implies } \sum_{A : i \in A} \hat{f}(A)^2 \geq \epsilon/2 \quad \text{and} \quad i \notin S_f \text{ implies } \sum_{A : i \notin A} \hat{f}(A)^2 \leq \epsilon.
\]

The proof uses the fact ([12] Lemma 4.1) that \( 4\sigma_i^2 I_{D, i}(f) = \sum_{A : i \in A} \hat{f}(A)^2 \) for any Boolean function \( f \) and any product distribution \( \mathcal{D} \). The algorithm uses sampling to approximate each \( \mu_i \) (and thus \( \sigma_i \)) and to approximate \( I_{D, i}(f) \). We call this algorithm \textbf{FindVariables2}.

5.2 The Learning Algorithm

We would like to modify \textbf{LearnMonotone} so that it uses the \( \phi \) basis rather than the \( \chi \) basis. However, as in [13] the algorithm does not know the exact values of \( \mu_i \) so it cannot use exactly the \( \phi \) basis; instead it approximates each \( \mu_i \) by a sample value \( \mu'_i \) and uses the resulting basis, which we call the \( \phi' \) basis. In more detail, the algorithm is as follows:

- Use \textbf{FindVariables2} to identify a set \( S_f \) of important variables.
- Draw \( m \) labeled examples \( (x^1, f(x^1)), \ldots, (x^m, f(x^m)) \) from \( EX(f, \mathcal{D}) \). Compute \( \mu'_i = \frac{1}{m} \sum_{j=1}^{m} x^j_i \) for \( 1 \leq i \leq n \). Define \( z'_i = (x_i - \mu'_i)/\sqrt{\mu'_i(1 - \mu'_i)} \) and \( \phi'_A = \prod_{i \in A} z'_i \).
- For every \( A \subseteq S_f \) with \( |A| \leq c \) set \( \alpha'_A = \frac{1}{m} \sum_{j=1}^{m} f(x^j) \phi'_A(x^j) \). If \( |\alpha'_A| > 1 \) set \( \alpha'_A = \text{sign}(\alpha'_A) \). For every \( A \) such that \( |A| > c \) or \( A \not\subseteq S_f \) set \( \alpha'_A = 0 \).
- Output the hypothesis \( g(\mathcal{F}) \), where \( g(\mathcal{F}) = \sum_A \alpha'_A \chi_A(\mathcal{F}) \).

We call this algorithm \textbf{LearnMonotone2}. As in [13] we note that setting \( \alpha'_A \) to \( \pm 1 \) if \( |\alpha'_A| > 1 \) can only bring the estimated value closer to the true value of \( \hat{f}(A) \).

5.3 Learning Monotone \( 2^{O(\sqrt{n})} \)-term DNF

For the most part only minor changes to the analysis of Section 3.3 are required. Since a term of size greater than \( d \) is satisfied by a random example from \( \mathcal{D} \) with probability less than \( (\frac{\beta}{2})^d \), we now take \( \log \frac{2^{2nm}}{\epsilon} = \Theta(\beta \log \frac{2m}{\epsilon}) \) as the term size bound for \( f_1 \). Proceeding as in Section 3.3 we obtain \( |S_f| = O(\beta \log \frac{2m}{\epsilon}) \). We similarly set a term size bound of \( \Theta(\beta \log \frac{1}{\epsilon}) \) for \( f_2 \). We use the \( \phi \) basis.
Parseval identity and inequality (10) in place of the \( \chi \) basis identity and inequality (2) respectively. Lemma 6 provides the required analogue of Mansour’s lemma for product distributions; using the new term size bound on \( f_2 \) we obtain 
\[
c = \Theta(\beta^2 \log\frac{1}{\epsilon} \log\frac{1}{\delta}).
\]

The one new ingredient in the analysis of LearnMonotone2 comes in bounding the quantity 
\[
Z = \sum_{|A| \leq c, A \subseteq S} (\alpha'_{A} - \tilde{f}(A))^2.
\]
In addition to the sampling error which would be present even if \( \mu'_i \) were exactly \( \mu_i \), we must also deal with error due to the fact that \( \alpha'_{A} \) is an estimate of the \( \phi' \) basis coefficient rather than the \( \phi \) basis coefficient \( f(A) \). An analysis entirely similar to that of Section 5.2 of [13] shows that taking 
\[
m = \text{poly}(c, |S|^{1/\epsilon}, \beta^2, 1/\epsilon, \log(1/\delta))
\]

Theorem 4. Under any product distribution \( \mathcal{D} \), for any \( \epsilon, \delta > 0 \), algorithm LearnMonotone2 can be used to learn \( t \)-term monotone DNF in time polynomial in \( n \), \( (\beta t \log\frac{m}{c})^{\beta^2 \log\frac{1}{\epsilon} \log\frac{1}{\delta}} \), and \( \log(1/\delta) \).

Since a constant-bounded product distribution \( \mathcal{D} \) has \( \beta = \Theta(1) \), we obtain

Corollary 4. For any constant \( \epsilon \) and any constant-bounded product distribution \( \mathcal{D} \), algorithm LearnMonotone2 learns \( 2^{O(\sqrt{\log n})} \)-term monotone DNF in \( \text{poly}(n, \log(1/\delta)) \) time.

5.4 Learning Small Constant-Depth Monotone Circuits on Few Variables

Using Lemma 7 and an analysis similar to the above, we obtain

Theorem 5. Fix \( d \geq 1 \) and let \( C \) be the class of depth \( d \), size \( M \) circuits which compute monotone functions on \( r \) out of \( n \) variables. Under any product distribution \( \mathcal{D} \), for any \( \epsilon, \delta > 0 \), algorithm LearnMonotone2 learns \( C \) in time polynomial in \( n \), \( r^{(\beta t \log\frac{m}{c})^{d}} \), and \( \log(1/\delta) \).

Corollary 5. Fix \( d \geq 1 \) and let \( C \) be the class of depth \( d \), size \( 2^{O((\log n)^{1/(d+1)})} \) circuits which compute monotone functions on \( 2^{O((\log n)^{1/(d+1)})} \) variables. Then for any constant \( \epsilon \) and any constant-bounded product distribution \( \mathcal{D} \), algorithm LearnMonotone2 learns \( C \) in \( \text{poly}(n, \log(1/\delta)) \) time.

6 Open Questions

The positive results reported in this paper for \( 2^{O(\sqrt{\log n})} \)-term DNF provide some hope that it may be possible to obtain a polynomial time algorithm for learning polynomial size monotone DNF under the uniform distribution from random examples only. We note that in the non-monotone case much less is known: in particular, it would be a significant step forward to give a polynomial time algorithm for learning arbitrary \( t(n) \)-term DNF under the uniform distribution, from random examples only, for any \( t(n) = \omega(1) \).
7 Acknowledgements

We thank Les Valiant for his advice and encouragement.

References