On Learning Monotone DNF under Product Distributions

Rocco A. Servedio*
Department of Computer Science
Columbia University
New York, NY 10027
rocco@cs.columbia.edu

December 4, 2003

Abstract

We show that the class of monotone $2^{O(\sqrt{\log n})}$-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. We also show that various classes of small constant-depth circuits which compute monotone functions are PAC learnable in polynomial time under the uniform distribution. All of our results extend to learning under any constant-bounded product distribution.

*This research was performed while the author was at Harvard University supported by NSF Grant CCR-98-77049 and by an NSF Mathematical Sciences Postdoctoral Research Fellowship.

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 [25] 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 twenty 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. The open question which motivates our work is whether polynomial-size *monotone* DNF formulas are learnable in polynomial time under the uniform distribution on $\{0,1\}^n$. This is an intriguing question since, as described below, efficient algorithms are known for several related problems.

It is known that if membership queries are allowed, then Angluin's exact learning algorithm [2] for monotone DNF yields an efficient algorithm for PAC learning polynomial size monotone DNF under any probability distribution. On the other hand, if membership queries are not allowed then a simple reduction shows that PAC learning monotone DNF under any distribution is as hard as PAC learning arbitrary DNF [17]. This equivalence is not preserved for distribution-specific learning, though, and thus it is possible that monotone DNF are efficiently learnable under the uniform distribution while general DNF are not.

Verbeurgt [26] gave an algorithm which can learn polynomial-size DNF (including monotone DNF) under the uniform distribution in time $n^{\log n}$. In the model of weak learning, Kearns *et al.* [18] showed that the class of all monotone Boolean functions (including monotone polynomial-size DNF) can be weakly learned under the uniform distribution in polynomial time. However, since weak and strong learnability are not necessarily equivalent under specific distributions, this latter result does not imply that monotone DNF are efficiently learnable under the uniform distribution.

A natural approach which several researchers have pursued is to try to learn monotone DNF with a limited number of terms under the uniform distribution. It has long been known [25] that DNF formulas with a constant number of terms can be PAC learned in polynomial time under arbitrary distributions. More recently Sakai and Maruoka [24] gave a polynomial-time algorithm for learning monotone $O(\log n)$-term DNF under the uniform distribution. In [8] Bshouty gave a polynomial-time uniform-distribution algorithm for learning a class which includes monotone $O(\log n)$-term DNF. Later Bshouty and Tamon [10] gave a polynomial-time algorithm for learning (under any constant-bounded product distribution) a class which includes monotone $O(\log^2 n/(\log\log n)^3)$-

term DNF.

1.2 Our Results

We give an algorithm for learning monotone DNF under the uniform distribution. If the desired accuracy level ϵ is any constant independent of n (the number of variables), then the algorithm learns $2^{O(\sqrt{\log n})}$-term monotone DNF over n variables in poly(n) time. The algorithm thus does not quite meet the usual definition of strong learning (which requires that any $\epsilon = 1/\text{poly}(n)$ be achievable in poly(n) time), but meets a much stronger condition than that of weak learning (which only requires accuracy $1/2 - 1/\text{poly}(n)$). We note that the algorithm of [10] for learning monotone DNF with $O((\log n)^2/(\log\log n)^3)$ terms also requires that ϵ be constant in order to achieve poly(n) runtime. Ours is the first polynomial time algorithm which uses only random examples and successfully learns monotone DNF with more than a polylogarithmic number of terms to high accuracy. We also show that essentially the same algorithm learns various classes of small constant-depth circuits which compute monotone functions. All of our results extend to learning under any constant-bounded product distribution.

Our algorithm combines ideas from Linial *et al.*'s influential paper [21] on learning AC^0 functions using the Fourier transform and Bshouty and Tamon's paper [10] 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 poly($\log n$); this result was later sharpened for DNF formulae by Mansour [22].

Our algorithm extends this approach in the following way: Let $C \subset AC^0$ 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 [10] 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 $\binom{|S_f|}{c}$ 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 $\binom{n}{c}$ 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 bit-strings, 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 unless otherwise indicated. 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 $\mathcal{D}$ is a distribution and f is a Boolean function on $\{0,1\}^n$, then as in [10, 13] we say that the *influence of x_i on f with respect to $\mathcal{D}$* 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 $\mathcal{D}$. 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

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

For monotone f this can be further simplified to

$$I_{\mathcal{D},i}(f) = \frac{1}{2} E_{\mathcal{D}}[f_{i,1} - f_{i,0}] = \frac{1}{2}\left(E_{\mathcal{D}}[f_{i,1}] - E_{\mathcal{D}}[f_{i,0}]\right). \tag{1}$$

We frequently use Chernoff bounds on sums of independent random variables [12]:

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

$$m \geqslant \frac{2B^2}{\epsilon^2} \ln \frac{2}{\delta} \qquad \textit{implies that} \qquad \Pr\left[\left|\frac{s_m}{m} - p\right| > \epsilon\right] \leqslant \delta.$$

2.1 The Learning Model

Our learning model is a distribution-specific version of Valiant's Probably Approximately Correct (PAC) model [25] which has been studied by many researchers, e.g. [4, 6, 9, 10, 11, 13, 15, 19, 20, 21, 22, 26]. Let C be a class of Boolean functions over $\{0,1\}^n$, let $\mathcal{D}$ 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* $EX(f, \mathcal{D})$ which, when queried, generates a random labeled example $\langle \overline{x}, f(\overline{x}) \rangle$ where $\overline{x}$ is drawn according to $\mathcal{D}$. 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 $\text{error}(h, f) = \Pr_{\mathcal{D}}[h(\overline{x}) \neq f(\overline{x})]$. We say that A *learns* C *under* $\mathcal{D}$ 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 $\text{error}(h, f) \leqslant \epsilon$.

2.2 The Discrete Fourier Transform

Let $\mathcal{U}$ 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

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

and norm defined as $\|f\| = \sqrt{\langle f, f \rangle}$. Given any subset $A \subseteq [n]$, the Fourier basis function $\chi_A : \{0,1\}^n \to \{-1, 1\}$ is defined by $\chi_A(\overline{x}) = (-1)^{|A \cap X|}$, 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 χ_A form an orthonormal basis for the vector space of real valued functions on $\{0,1\}^n$; we refer to this basis as *the* χ *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 χ basis. Since the functions χ_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

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

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 [10, 21]

$$\Pr_{\mathcal{U}}[f \neq \text{sign}(g)] \leqslant E_{\mathcal{U}}[(f - g)^2] \tag{2}$$

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

3 Learning under Uniform Distributions

3.1 Identifying Relevant Variables

The following lemma, which is implicit in [10], gives an efficient algorithm for identifying the important variables of a monotone Boolean function. We refer to this algorithm as `FindVariables`.

Lemma 2 *Let $f : \{0,1\}^n \rightarrow \{-1,1\}$ be a monotone Boolean function. There is an algorithm which has access to $EX(f,\mathcal{U})$, runs in 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*

$$i \in S_f \textit{ implies } \sum_{A:i\in A} \hat{f}(A)^2 \geqslant \epsilon/2 \qquad \textit{and} \qquad i \notin S_f \textit{ implies } \sum_{A:i\in A} \hat{f}(A)^2 \leqslant \epsilon.$$

Proof: Kahn *et al.* ([16] Section 3) have shown that

$$I_{\mathcal{U},i}(f) = \sum_{A:i\in A} \hat{f}(A)^2. \tag{3}$$

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 $E_{\mathcal{U}}[f_{i,b}]$. ■

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 $\langle \overline{x}^1, f(\overline{x}^1)\rangle, \ldots, \langle \overline{x}^m, f(\overline{x}^m)\rangle$ from $EX(f,\mathcal{U})$. For every $A \subseteq S_f$ with $|A| \leqslant c$ 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 sign$(g(\overline{x}))$, where $g(\overline{x}) = \sum_A \alpha_A \chi_A(\overline{x})$.

The algorithm thus estimates $\hat{f}(A)$ for $A \subseteq S_f, |A| \leqslant c$ by sampling and constructs a hypothesis using these approximate Fourier coefficients. The values of m and c and the parameter settings for ϵ and δ in `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. (This term size bound is chosen so that we will ultimately end up with an $\epsilon/4$ on the right side of inequality (7) below.) 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 $\Pr_{\mathcal{U}}[f(\overline{x}) \neq f_1(\overline{x})] < \frac{\epsilon}{32n}$ (this type of argument was first used by Verbeurgt [26]). Let $R \subseteq [n]$ be the set of variables which f_1 depends on; it is clear that $|R| \leqslant t \log \frac{32tn}{\epsilon}$. Moreover, since $I_{\mathcal{U},i}(f_1) = 0$ for $i \notin R$, 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

$$\begin{aligned} E_{\mathcal{U}}[(f-f_1)^2] &= \sum_A (\hat{f}(A) - \hat{f}_1(A))^2 \\ &= \sum_{A \subseteq R} (\hat{f}(A) - \hat{f}_1(A))^2 + \sum_{A \not\subseteq R} (\hat{f}(A) - \hat{f}_1(A))^2 \\ &= \sum_{A \subseteq R} (\hat{f}(A) - \hat{f}_1(A))^2 + \sum_{A \not\subseteq R} (\hat{f}(A))^2 \\ &< \epsilon/8n. \end{aligned}$$

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

$$i \notin R \text{ implies } \sum_{A: i \in A} \hat{f}(A)^2 < \frac{\epsilon}{8n}. \tag{4}$$

We set the parameters of `FindVariables` so that with high probability

$$i \in S_f \quad \text{implies} \quad \sum_{A: i \in A} \hat{f}(A)^2 \geqslant \epsilon/8n \tag{5}$$

$$i \notin S_f \quad \text{implies} \quad \sum_{A: i \in A} \hat{f}(A)^2 \leqslant \epsilon/4n. \tag{6}$$

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

$$\sum_{A \not\subseteq S_f} \hat{f}(A)^2 \leqslant \epsilon/4. \tag{7}$$

The following lemma is due to Mansour ([22] Lemma 3.2):

Lemma 3 (Mansour) *Let f be a DNF with terms of size at most d. Then for all $\epsilon > 0$*

$$\sum_{|A| > 20d \log(2/\epsilon)} \hat{f}(A)^2 \leqslant \epsilon/2.$$

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 $d = \log \frac{32tn}{\epsilon}$ 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 $\log \frac{32t}{\epsilon}$ variables. Let $c = 20 \log \frac{32t}{\epsilon} \log \frac{8}{\epsilon}$. Mansour's lemma implies that

$$\sum_{|A|>c} \hat{f}_2(A)^2 \leqslant \epsilon/8. \tag{8}$$

Moreover, we have $\Pr_{\mathcal{U}}[f \neq f_2] \leqslant \epsilon/32$ and hence

$$4 \Pr_{\mathcal{U}}[f \neq f_2] = E_{\mathcal{U}}[(f - f_2)^2] = \sum_A (\hat{f}(A) - \hat{f}_2(A))^2 \leqslant \epsilon/8. \tag{9}$$

Let α_A and $g(\overline{x})$ be as defined in `LearnMonotone`. Using inequality (2) from Section 2.2, we have

$$\Pr[\text{sign}(g) \neq f] \leqslant E_{\mathcal{U}}[(g - f)^2] = \sum_A (\alpha_A - \hat{f}(A))^2 = X + Y + Z,$$

where

$$\begin{aligned} X &= \sum_{|A|\leqslant c, A \not\subseteq S_f} (\alpha_A - \hat{f}(A))^2, \\ Y &= \sum_{|A|>c} (\alpha_A - \hat{f}(A))^2, \\ Z &= \sum_{|A|\leqslant c, A \subseteq S_f} (\alpha_A - \hat{f}(A))^2. \end{aligned}$$

To bound X, we observe that $\alpha_A = 0$ for $A \not\subseteq S_f$, so by (7) we have

$$X = \sum_{|A|\leqslant c, A \not\subseteq S_f} \hat{f}(A)^2 \leqslant \sum_{A \not\subseteq S_f} \hat{f}(A)^2 \leqslant \epsilon/4.$$

To bound Y, we note that $\alpha_A = 0$ for $|A| > c$ and hence $Y = \sum_{|A|>c} \hat{f}(A)^2$. Since $\hat{f}(A)^2 \leqslant 2(\hat{f}(A) - \hat{f}_2(A))^2 + 2\hat{f}_2(A)^2$, we have

$$\begin{aligned} Y &\leqslant 2 \sum_{|A|>c} (\hat{f}(A) - \hat{f}_2(A))^2 + 2 \sum_{|A|>c} \hat{f}_2(A)^2 \\ &\leqslant 2 \sum_A (\hat{f}(A) - \hat{f}_2(A))^2 + \epsilon/4 \\ &\leqslant \epsilon/2 \end{aligned}$$

by inequalities (8) and (9) respectively.

It remains to bound $Z = \sum_{|A|\leqslant c, A \subseteq S_f} (\alpha_A - \hat{f}(A))^2$. As in Linial *et al.* [21] this sum can be made less than $\epsilon/4$ by taking m sufficiently large so that with high probability each estimate α_A differs from the true value $\hat{f}(A)$ by at most $\sqrt{\epsilon/(4|S_f|^c)}$. A straightforward Chernoff bound argument shows that taking $m = \text{poly}(|S_f|^c, 1/\epsilon, \log(1/\delta))$ suffices.

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

Theorem 4 *Under the uniform distribution, for any $\epsilon, \delta > 0$, the algorithm* `LearnMonotone` *can be used to learn t-term monotone DNF in time polynomial in n, $(t \log \frac{tn}{\epsilon})^{\log \frac{t}{\epsilon} \log \frac{1}{\epsilon}}$ and* $\log(1/\delta)$.

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

Corollary 5 *For any constant ϵ algorithm* `LearnMonotone` *learns $2^{O(\sqrt{\log n})}$-term monotone DNF in poly$(n, \log(1/\delta))$ time under the uniform distribution.*

As noted earlier, Bshouty and Tamon's algorithm [10] for learning monotone DNF with $O((\log n)^2/(\log\log n)^3)$ terms also requires that ϵ be constant in order to achieve poly(n) runtime.

3.4 Learning Small Constant-Depth Monotone Circuits

3.4.1 Circuits with Few Relevant 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 `LearnMonotone` can be used to learn C in time polynomial in n, $r^{(\log(M/\epsilon))^d}$ and $\log(1/\delta)$. As in the last section the `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.* [21] to bound the total weight of high-order Fourier coefficients for constant-depth circuits:

Lemma 6 (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*

$$\sum_{|A|>c} \hat{f}(A)^2 \leqslant 2M2^{-c^{1/d}/20}.$$

More precisely, fix $f \in C$ and let $R \subseteq [n], |R| \leqslant r$ be the variables which f depends on. Clearly $i \notin R$ implies that $I_{\mathcal{U},i}(f) = 0$. We may run `FindVariables` with parameter settings such that

$$i \in S_f \text{ implies } \sum_{A:i\in A} \hat{f}(A)^2 \geqslant \epsilon/8n$$
$$i \notin S_f \text{ implies } \sum_{A:i\in A} \hat{f}(A)^2 \leqslant \epsilon/4n.$$

Consequently we have that $S_f \subseteq R$ so $|S_f| \leqslant r$.

The proof that `LearnMonotone` learns C is similar to the proof of Section 3.3 but simpler. From the second equation above we get that

$$\sum_{A\not\subseteq S_f} \hat{f}(A)^2 \leqslant \epsilon/4$$

which is an analogue to equation (7). As before we get that

$$\Pr[\mathrm{sign}(g) \neq f] \leqslant X + Y + Z$$

with

$$X = \sum_{|A| \leqslant c, A \not\subseteq S_f} (\alpha_A - \hat{f}(A))^2, \quad Y = \sum_{|A|>c} (\alpha_A - \hat{f}(A))^2, \quad Z = \sum_{|A| \leqslant c, A \subseteq S_f} (\alpha_A - \hat{f}(A))^2.$$

The bound $X \leqslant \epsilon/4$ follows from the analogue to (7). The bound $Y = \sum_{|A|>c} \hat{f}(A)^2 \leqslant \epsilon/2$ follows from the lemma of Linial *et al.* with a suitable choice of c as described below. The bound $Z \leqslant \epsilon/4$ follows by sampling to estimate each Fourier coefficient to high accuracy with high confidence. More precisely, there are at most r^c coefficients so we estimate each to accuracy $\sqrt{\epsilon/4r^c}$ to obtain $Z \leqslant \epsilon/4$. Thus taking m to be $\mathrm{poly}(r^c/\epsilon, \log(1/\delta))$ is sufficient.

So all in all, the running time of the algorithm is $\mathrm{poly}(n, 1/\epsilon, r^c, \log(1/\delta))$ and the algorithm succeeds in learning provided that $2M2^{-c^{1/d}/20} \leqslant \epsilon/2$ i.e. provided that $c \geqslant (20\log(4M/\epsilon))^d$. Choosing c appropriately, we have the following theorem:

Theorem 7 *Fix $d \geqslant 1$ 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 $\epsilon, \delta > 0$, algorithm* `LearnMonotone` *learns class $C_{d,M,r}$ in time polynomial in n, $r^{(\log(M/\epsilon))^d}$ and $\log(1/\delta)$.*

3.4.2 Learning Small Constant-Depth Monotone Circuits

In this section we strengthen Theorem 7 by removing the restriction that the circuits being learned have only r relevant variables. The key idea which enables this is that any AND (respectively OR) gate with many literals as inputs will almost always take value 0 (respectively 1), so ignoring all such gates will not incur much error.

Specifically, let F be a Boolean circuit of depth d and size M which computes a monotone function. By De Morgan's laws, without loss of generality we may suppose that F contains only AND and NOT gates. (Recall that monotone functions can be computed by circuits which contain NOT gates; here we are allowing NOT gates to be located anywhere in the circuit, not just at the inputs.) Let F' be the circuit obtained from F by replacing each AND gate which has more than ℓ distinct literals among its inputs with the constant 0 (the value of ℓ will be specified later). For any such AND gate, this changes its output (for a uniformly random setting of the input variables) with probability at most $2^{-\ell}$, so we have that $\Pr[F(x) \neq F'(x)] \leqslant M2^{-\ell}$ for uniform random x. Consequently, given a sample of t labeled examples of F, we have that F' is consistent with the sample with probability at least $1 - tM2^{-\ell}$. Since F' has at most $r \equiv M\ell$ relevant variables, by Theorem 7 we have that if `LearnMonotone` is run on a uniform sample of t =poly$(n, (M\ell)^{(\log(M/\epsilon))^d}, \log 1/\delta)$ examples labeled according to F', then with probability at least $1 - \delta/2$ the hypothesis h which it outputs will satisfy $\Pr[h(x) \neq F'(x)] \leqslant \epsilon/2$.

Thus, the necessary conditions on ℓ are as follows:

- $\ell \geqslant \log(2M/\epsilon)$: this ensures that $\Pr[F(x) \neq F'(x)] \leqslant \epsilon/2$, so an $\epsilon/2$-approximator h for F' will be an ϵ-approximator for F as desired.
- $\ell \geqslant 1 + \log(2Mt/\delta)$: this ensures that the sample used for learning is consistent with F' with probability at least $1 - \delta/2$.
- t =poly$(n, (M\ell)^{(\log(M/\epsilon))^d}, \log 1/\delta)$: this ensures that `LearnMonotone` has enough examples to learn successfully.

Taking $\ell = O((\log(Mn/\delta\epsilon))^{d+1})$ satisfies all of these conditions. We thus have the following theorem:

Theorem 8 *Fix $d \geqslant 1$ and let $C_{d,M}$ be the class of depth d, size M circuits which compute monotone functions on n variables. Under the uniform distribution, for any $\epsilon, \delta > 0$, algorithm* `LearnMonotone` *learns class $C_{d,M}$ in time polynomial in $(M(\log(Mn/\delta\epsilon))^{d+1})^{(\log(M/\epsilon))^d}$ and n.*

One interesting corollary is the following:

Corollary 9 *Fix $d \geqslant 1$ and let C_d be the class of depth d, size $2^{O((\log n)^{1/(d+1)})}$ circuits which compute monotone functions. Then for any constant ϵ, δ algorithm* `LearnMonotone` *learns class C_d in poly(n) 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 $2^{O((\log n)^{1/(d+1)})}$ monotone circuits (i.e. circuits of the stated size and depth which contain only AND and OR gates). This follows from results of Okol'nishnikova [23] and Ajtai and Gurevich [1] (see also [7] 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 $\mathcal{D}$ assigns values independently to each variable, so for $\overline{a} \in \{0,1\}^n$ we have $\mathcal{D}(\overline{a}) = \left(\prod_{a_i=1} \mu_i\right)\left(\prod_{a_i=0}(1-\mu_i)\right)$. 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 $\mathcal{D}$ 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 β denote $\max_{i=1,\ldots,n}(1/\mu_i, 1/(1-\mu_i))$. Throughout the rest of this paper $\mathcal{D}$ denotes a product distribution.

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

$$\langle f, g\rangle_{\mathcal{D}} = \sum_{\overline{x}\in\{0,1\}^n} \mathcal{D}(\overline{x}) f(\overline{x}) g(\overline{x}) = E_{\mathcal{D}}[fg]$$

and a corresponding norm $\|f\|_{\mathcal{D}} = \sqrt{\langle f, f\rangle_{\mathcal{D}}}$. We refer to this norm as *the $\mathcal{D}$-norm.* For $i = 1, \ldots, n$ let $z_i = (x_i - \mu_i)/\sigma_i$. Given $A \subseteq [n]$, let ϕ_A be defined as $\phi_A(\overline{x}) = \prod_{i \in A} z_i$. As noted by Bahadur [5] and Furst *et al.* [11], the 2^n functions ϕ_A form an orthonormal basis for the vector space of real valued functions on $\{0,1\}^n$ with respect to the $\mathcal{D}$-norm, i.e. $\langle \phi_A, \phi_B\rangle_{\mathcal{D}}$ is 1 if $A = B$ and is 0 otherwise. We refer to this basis as *the ϕ basis.* The following fact is useful:

Fact 10 (Bahadur; Furst *et. al*) *The ϕ basis is the basis which would be obtained by Gram-Schmidt orthonormalization (with respect to the $\mathcal{D}$-norm) of the χ basis performed in order of increasing $|A|$.*

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

$$E_{\mathcal{D}}[f^2] = \|f\|_{\mathcal{D}}^2 = \sum_{A \subseteq [n]} \tilde{f}(A)^2$$

which is 1 for Boolean f. Finally, for Boolean f and real-valued g we have ([11] Lemma 10)

$$\Pr_{\mathcal{D}}[f \neq \text{sign}(g)] \leqslant E_{\mathcal{D}}[(f-g)^2]. \tag{10}$$

Furst *et al.* [11] analyzed the ϕ 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 [11], and in Section 5 we use these sharpened results together with techniques from [11] to obtain product distribution analogues of our algorithms from Section 3.

4.1 Some ϕ Basis Fourier Lemmas

A *random restriction* $\rho_{p,\mathcal{D}}$ is a mapping from $\{x_1, \ldots, x_n\}$ to $\{0, 1, *\}$ which is chosen randomly in the following way: each x_i is mapped to $*$ with probability p, to 1 with probability $(1-p)\mu_i$, and to 0 with probability $(1-p)(1-\mu_i)$. Given a restriction $\rho_{p,\mathcal{D}}$ and a Boolean function f, we write $f\lceil\rho$ to represent the function $f(\rho_{p,\mathcal{D}}(\overline{x}))$ whose variables are those x_i which are mapped to $*$ and whose other x_i are instantiated as 0 or 1 according to $\rho_{p,\mathcal{D}}$. Note that once $\rho_{p,\mathcal{D}}$ has been chosen, $f\lceil\rho$ is a specific deterministic function; the randomness stems entirely from the choice of $\rho_{p,\mathcal{D}}$ as described above.

The following variant of Håstad's well known switching lemma [14] follows directly from the argument in Section 4 of [3]:

Lemma 11 *Let $\mathcal{D}$ be a product distribution with parameters μ_i and β as defined above, let f be a CNF formula where each clause has at most t literals, and let $\rho_{p,\mathcal{D}}$ be a random restriction. Then with probability at least $1-(4\beta pt)^s$ over the choice of $\rho_{p,\mathcal{D}}$ we have that:*

1. *the function $f\lceil\rho$ 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.*

The following corollary is a product distribution analogue of ([21] Corollary 1):

Corollary 12 *Let $\mathcal{D}$ be a product distribution with parameters μ_i and β, let f be a CNF formula where each clause has at most t literals, and let $\rho_{p,\mathcal{D}}$ be a random restriction. Then with probability at least $1-(4\beta pt)^s$ we have that $\widetilde{f\lceil\rho}(A)=0$ for all $|A|>s$.*

Proof: Linial *et al.* [21] show (in the proof of Corollary 1 in their paper) that if $f\lceil\rho$ satisfies properties (1) and (2) of Lemma 11 then $\widehat{f\lceil\rho}(A)=0$ for all $|A|>s$. Hence such a $f\lceil\rho$ is in the space spanned by $\{\chi_A : |A| \leqslant s\}$. By Fact 10 and the nature of Gram-Schmidt orthonormalization, this is the same space which is spanned by $\{\phi_A : |A| \leqslant s\}$, and the corollary follows. ■

Corollary 12 is a sharpened version of a similar lemma, implicit in [11], which states that under the same conditions with probability at least $1-(5\beta pt/2)^s$ we have $\widetilde{f\lceil\rho}(A)=0$ for all $|A|>s^2$. Armed with the sharper Corollary 12, the proofs of Lemmas 7, 8 and 9 from [11] now directly yield

Lemma 13 *For any Boolean function f, for any integer c,*

$$\sum_{|A|>c} \tilde{f}(A)^2 \leqslant 2 \Pr_{\rho_{p,D}}[\widetilde{f\lceil\rho}(A) \neq 0 \textit{ for some } |A| > cp/2].$$

Boolean duality implies that the conclusion of Corollary 12 also holds if f is a DNF with each term of length at most t. Taking $p=1/8\beta t$ and $s=\log\frac{4}{\epsilon}$ in this DNF version of Corollary 12 and $c=16\beta t\log\frac{4}{\epsilon}$ in Lemma 13, we obtain the following analogue of Mansour's lemma (Lemma 3) for the ϕ basis:

Lemma 14 *Let f be a DNF with terms of size at most t. Then for all $\epsilon>0$*

$$\sum_{|A|>16\beta t\log(4/\epsilon)} \tilde{f}(A)^2 \leqslant \epsilon/2.$$

We will also need an analogue of the Linial *et al.* lemma (Lemma 6) for the ϕ basis. As in Lemma 2 of [21], by successively applying Lemma 11 and the DNF version of the lemma to the lowest levels of a circuit and then applying Corollary 12 we obtain the following:

Lemma 15 *Let $\mathcal{D}$ be a product distribution with parameters μ_i and β, let f be a Boolean function computed by a circuit of size M and depth d, and let $\rho_{p,\mathcal{D}}$ be a random restriction. If*

$$p=\frac{1}{2^{2\beta}(8\beta s)^{d-1}}$$

then we have that

$$\Pr[\widetilde{f\lceil\rho}=0 \textit{ for some } |A|>s] \leqslant M2^{-s}.$$

Proof sketch: Our proof closely follows the proof of Lemma 2 in [21]. We view the restriction ρ as being obtained by first performing a random restriction in which $\Pr[*] = 1/2^{2\beta}$, and then $d-1$ consecutive restrictions each with $\Pr[*] = 1/(8\beta s)$.

After the first restriction, each original bottom-level gate has fanin greater than s with probability at most 2^{-s}. To see this, observe that under the first restriction each literal is set to $*$ with probability $p' = 1/2^{2\beta}$ and is set to 0 (1) with probability at least $(1-p')/\beta$. Now set $r = s\beta/(1-p')$ and consider separately each bottom-level gate depending on how its fanin compares to r:

1. For any gate with fanin at least r, the probability that the gate is not eliminated (that no literal is set to 0 for an AND, set to 1 for an OR) is at most $(1-\frac{1-p'}{\beta})^r \leqslant e^{-s} < 2^{-s}$.

2. For any gate with fanin at most r, the probability that at least s input literals are assigned a $*$ is at most $\binom{r}{s}p'^s < 2^r p'^s = 2^{\beta s/(1-p')}p'^s$. This is at most 2^{-s} provided that $2^{\beta/(1-p')}p' \leqslant 1/2$ which is easily verified from the definition of p'.

Now we apply $d-2$ more restrictions, each with $\Pr[*] = 1/(8\beta s)$. As in [21] we use Lemma 11 after each restriction to convert the lower two levels of the circuit from CNF to DNF (or vice versa), preserving by our choice of p the property that each clause (term) has size at most s, and incurring a failure probability of at most 2^{-s} for each gate.

After these $d-2$ stages, what remains is a CNF (or DNF) with clauses (terms) of size at most s. We apply the last restriction with $p = 1/(8\beta s)$, and Corollary 12 implies that the failure probability at this stage is also at most 2^{-s}. Thus, as in [21], with overall probability at least $1 - M2^{-s}$ we have that $\widetilde{f\lceil\rho}(A) = 0$ for all $|A| > s$, and the lemma is proved. ∎

With Lemma 15 in hand we can prove the following sharper version of the main lemma from [11]:

Lemma 16 *Let f be a Boolean function computed by a circuit of depth d and size M and let c be any integer. Then*

$$\sum_{|A|>c} \tilde{f}(A)^2 \leqslant 2M2^{-\left(c/2^{2\beta+1}(8\beta)^{d-1}\right)^{1/d}} < 2M2^{-c^{1/d}/2^{(2\beta+1)/d}(8\beta)}.$$

Proof: The proof closely follows the proof of Lemma 9 in [11]. From Lemma 13 we have that

$$\sum_{|A|>c} \tilde{f}(A)^2 \leqslant 2 \Pr_{\rho_{p,D}}[\widetilde{f\lceil\rho}(A) \neq 0 \text{ for some } |A| > cp/2]. \qquad (11)$$

Let p and s satisfy $p = 1/2^{2\beta}(8\beta s)^{d-1}$ and $s = cp/2$. Lemma 15 now implies that Equation (11) is at most $2M2^{-s}$. Straightforward algebraic manipulations show that $s = \left(c/2^{2\beta+1}(8\beta)^{d-1}\right)^{1/d}$ and the lemma is proved. ∎

The version of Lemma 16 given in [11] has $1/(d+2)$ instead of $1/d$ in the exponent of c. This new tighter bound enables 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 [11].

5 Learning under Product Distributions

5.1 Identifying Relevant Variables

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

Lemma 17 *Let $f : \{0,1\}^n \to \{-1,1\}$ be a monotone Boolean function. There is an algorithm which has access to $EX(f,\mathcal{D})$, runs in 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*

$$i \in S_f \text{ implies } \sum_{A:i\in A} \tilde{f}(A)^2 \geqslant \epsilon/2 \qquad \text{and} \qquad i \notin S_f \text{ implies } \sum_{A:i\in A} \tilde{f}(A)^2 \leqslant \epsilon.$$

Proof: We show that for each i, with probability $1-\delta/n$ the value $\sum_{A:i\in A} \tilde{f}(A)^2$ can be estimated to within an additive $\epsilon/4$ in poly$(n, \beta, 1/\epsilon, \log 1/\delta)$ time steps. By Lemma 4.1 of [10] we have that $\sum_{A:i\in A} \tilde{f}(A)^2 = 4\sigma_i^2 I_{\mathcal{D},i}(f)$ for any Boolean function f and any product distribution $\mathcal{D}$. Now observe that if $a = bc$ and $0 \leqslant b, c \leqslant 1$ and $-\tau \leqslant \tau_1, \tau_2 \leqslant \tau \leqslant 1$, then we have

$$|(b+\tau_1)(c+\tau_2) - a| = |bc + b\tau_2 + c\tau_1 + \tau_1\tau_2 - a| \leqslant (b+c)\tau + \tau^2 \leqslant 3\tau.$$

Consequently, since $0 \leqslant I_{\mathcal{D},i}(f) \leqslant 1$ and $0 \leqslant 4\sigma_i^2 \leqslant 1$, in order to estimate $\sum_{A:i\in A} \tilde{f}(A)^2 = 4\sigma_i^2 I_{\mathcal{D},i}(f)$ to within an additive $\epsilon/4$ it is sufficient to estimate each of $I_{\mathcal{D},i}(f)$ and $4\sigma_i^2$ to within an additive $\epsilon/12$.

First we consider $I_{\mathcal{D},i}(f)$. Recalling that $I_{\mathcal{D},i}(f) = \frac{1}{2}(E_{\mathcal{D}}[f_{i,1}] - E_{\mathcal{D}}[f_{i,0}])$, it is sufficient to estimate each of these expectations to within an additive $\epsilon/12$. A standard application of Chernoff bounds shows that poly$(1/\epsilon, \log \frac{1}{\delta'})$ random examples with $x_i = 1$ are required to estimate $E_{\mathcal{D}}[f_{i,1}]$ to within $\epsilon/12$ with confidence $1-\delta'$. Since a random example drawn from $EX(f,\mathcal{D})$ has $x_i = 1$ with probability at least $1/\beta$, another application of Chernoff bounds shows that with probability $1-\delta'$, at most poly$(1/\beta, \log \frac{1}{\delta'})$ draws from $EX(f,\mathcal{D})$ are required to obtain a random example with $x_i = 1$. Combining these bounds we can estimate $E_{\mathcal{D}}[f_{i,1}]$ to within an additive $\epsilon/12$ with confidence $1-\frac{\delta}{4n}$ in time poly$(n, \beta, 1/\epsilon, \log \frac{1}{\delta})$. The same is easily seen to hold for estimating $E_{\mathcal{D}}[f_{i,1}]$. Thus we can estimate $I_{\mathcal{D},i}(f)$ to within an additive $\epsilon/12$ in time poly$(n, \beta, 1/\epsilon, \log \frac{1}{\delta})$ with confidence $1-\frac{\delta}{2n}$.

Now we show that σ_i^2 can be efficiently estimated to within an additive $\epsilon/48$. Chernoff bounds imply that by sampling we can obtain an estimate $\tilde{\mu}_i$ of μ_i which is accurate to within an additive error of $\pm\tau$ with probability $1-\frac{\delta}{2n}$ in poly$(1/\tau, \log 1/\delta)$ time. We use each estimated value $\tilde{\mu}_i$ to compute an estimate $\tilde{\sigma}_i = \sqrt{\tilde{\mu}_i(1-\tilde{\mu}_i)}$ of σ_i. One can straightforwardly verify that $\tilde{\sigma}_i$ differs from the true value σ_i by at most $\sqrt{\tau}$, and thus $\tilde{\sigma}_i^2$ differs from the true value σ_i^2 by at most τ. Thus it suffices to take $\tau = \epsilon/48$, and the required estimate of σ_i^2 can be obtained in poly$(n, 1/\epsilon, \log \frac{1}{\delta})$ time. ■

We refer to the algorithm of Lemma 17 as `FindVariables2`.

5.2 The Learning Algorithm

We would like to modify LearnMonotone so that it uses the ϕ basis rather than the χ basis. However, as in [11] the algorithm does not know the exact values of μ_i so it cannot use exactly the ϕ basis; instead it approximates each μ_i by a sample value μ_i' and uses the resulting basis, which we call the ϕ' basis. In more detail, the algorithm is as follows:

- Use FindVariables2 to identify a set S_f of important variables.
- Draw m labeled examples $\langle \overline{x}^1, f(\overline{x}^1)\rangle, \ldots, \langle \overline{x}^m, f(\overline{x}^m)\rangle$ from $EX(f, \mathcal{D})$. Compute $\mu_i' = \frac{1}{m}\sum_{j=1}^m x_i^j$ for $1 \leqslant i \leqslant n$. 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 $|A| \leqslant c$ set $\alpha_A' = \frac{1}{m}\sum_{j=1}^m f(\overline{x}^j)\phi_A'(\overline{x}^j)$. If $|\alpha_A'| > 1$ set $\alpha_A' = \mathrm{sign}(\alpha_A')$. For every A such that $|A| > c$ or $A \not\subseteq S_f$ set $\alpha_A' = 0$.
- Output the hypothesis $\mathrm{sign}(g(\overline{x}))$, where $g(\overline{x}) = \sum_A \alpha_A' \chi_A(\overline{x})$.

We call this algorithm LearnMonotone2. As in [11] we note that setting α_A' to ± 1 if $|\alpha_A'| > 1$ can only bring the estimated value closer to the true value of $\tilde{f}(A)$.

5.3 Learning Monotone $2^{O(\sqrt{\log n})}$-term DNF under Product Distributions

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 $(\frac{\beta-1}{\beta})^d$, we now take $\log_{\frac{\beta}{\beta-1}} \frac{32tn}{\epsilon} = \Theta(\beta \log \frac{tn}{\epsilon})$ as the term size bound for f_1. Proceeding as in Section 3.3 we obtain $|S_f| = O(\beta t \log \frac{tn}{\epsilon})$. We similarly set a term size bound of $\Theta(\beta \log \frac{t}{\epsilon})$ for f_2. We use the ϕ basis Parseval identity and inequality (10) in place of the χ basis identity and inequality (2) respectively. Lemma 14 provides the required analogue of Mansour's lemma for product distributions; using the new term size bound on f_2 we obtain $c = \Theta(\beta^2 \log \frac{t}{\epsilon} \log \frac{1}{\epsilon})$.

The one new ingredient in the analysis of LearnMonotone2 comes in bounding the quantity $Z = \sum_{|A| \leqslant c, A \subseteq S_f} (\alpha_A' - \tilde{f}(A))^2$. In addition to the sampling error which would be present even if μ_i' were exactly μ_i, we must also deal with error due to the fact that α_A' is an estimate of the ϕ' basis coefficient rather than the ϕ basis coefficient $\tilde{f}(A)$. An analysis entirely similar to that of Section 5.2 of [11] shows that taking $m = \mathrm{poly}(c, |S_f|^c, \beta^c, 1/\epsilon, \log(1/\delta))$ suffices. We thus have

Theorem 18 *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, $(\beta t \log \frac{tn}{\epsilon})^{\beta^2 \log \frac{t}{\epsilon} \log \frac{1}{\epsilon}}$, and $\log(1/\delta)$.*

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

Corollary 19 *For any constant ϵ and any constant-bounded product distribution $\mathcal{D}$, algorithm* LearnMonotone2 *learns $2^{O(\sqrt{\log n})}$-term monotone DNF in poly$(n, \log(1/\delta))$ time.*

5.4 Learning Small Constant-Depth Monotone Circuits under Product Distributions

Let f be a monotone function computed by a size M, depth d Boolean circuit with r relevant variables An analysis similar to that of Section 3.4.1 but using Lemma 16 in place of Lemma 6 shows that it is sufficient for us to take $c \geqslant 2^{2\beta+1}(8\beta \log(4M/\epsilon))^d$. We obtain

Theorem 20 *Fix $d \geqslant 1$ 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 any constant-bounded product distribution $\mathcal{D}$, for any $\epsilon, \delta > 0$, algorithm* `LearnMonotone2` *learns class $C_{d,M,r}$ in time polynomial in n, $(M(\log(Mn/\delta\epsilon))^{d+1})^{(\log(M/\epsilon))^d}$ and $\log 1/\delta$.*

The argument from Section 3.4.2 can be used here as well to show that we do not need to put an *a priori* upper bound on the number of relevant variables. We obtain:

Theorem 21 *Fix $d \geqslant 1$ and let $C_{d,M}$ be the class of depth d, size M circuits which compute monotone functions on n variables. Under any constant-bounded product distribution $\mathcal{D}$, for any $\epsilon, \delta > 0$, algorithm* `LearnMonotone2` *learns class $C_{d,M}$ in time polynomial in n and $(M(\log(Mn/\delta\epsilon))^{d+1})^{(\log(M/\epsilon))^d}$.*

Corollary 22 *Fix $d \geqslant 1$ and let C_d be the class of depth d, size $2^{O((\log n)^{1/(d+1)})}$ circuits which compute monotone functions. Then for any constant ϵ, δ and any constant-bounded product distribution $\mathcal{D}$, algorithm* `LearnMonotone2` *learns class C_d in poly(n) time.*

Thus all of our uniform distribution learning results generalize to learning under any constant-bounded product distribution.

6 Open Questions

There are several natural questions for further work. Can the $2^{\sqrt{\log n}}$ term bound of our algorithm be improved to $2^{(\log n)^{1-\alpha}}$ for any $\alpha > 0$? Can an algorithm be obtained which runs in polynomial time for $\epsilon = o(1)$ or even for $\epsilon = 1/\text{poly}(n)$? These would be interesting steps toward the more ambitious goal of developing a polynomial time algorithm for learning poly(n)-term monotone DNF under the uniform distribution.

We close by noting that in the non-monotone case much less is known; in particular, it would be a breakthrough result 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 enouragement. We thank the anonymous referees for helpful suggestions which improved the exposition of the paper.

References

[1] M. Ajtai and Y. Gurevich. Monotone versus positive, *J. ACM* **34**(4) (1987), 1004-1015.

[2] D. Angluin. Queries and concept learning. *Machine Learning* **2**(4) (1988), 319-342.

[3] P. Beame. A switching lemma primer. Tech. report UW-CSE-95-07-01, University of Washington, November 1994.

[4] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. Rudich. Weakly learning DNF and characterizing statistical query learning using Fourier analysis, *in* "Proc. 26th Ann. Symp. on Theory of Computing" (1994), 253-262.

[5] R. Bahadur. A representation of the joint distribution of responses to n dichotomous items, *in* Herbert Solomon, ed., *Studies in Item Analysis and Prediction,* pp. 158-168, Stanford University Press, 1961.

[6] M. Bellare. A technique for upper bounding the spectral norm with applications to learning, *in* "Proc. Fifth Ann. Workshop on Comp. Learning Theory" (1992), 62-70.

[7] R. Boppana and M. Sipser. The complexity of finite functions, in *Handbook of Theoretical Computer Science,* vol. A, MIT Press, 1990.

[8] N. Bshouty. Exact learning Boolean functions via the monotone theory. *Information and Computation* **123**(1) (1995), 146-153.

[9] N. Bshouty, J. Jackson, and C. Tamon. More efficient PAC-learning of DNF with membership queries under the uniform distribution, *in* "Proc. 12th Ann. Conf. on Comp. Learning Theory" (1999), 286-295.

[10] N. Bshouty and C. Tamon. On the Fourier spectrum of monotone functions, *J. ACM* **43**(4) (1996), 747-770.

[11] M. Furst, J. Jackson, and S. Smith. Improved learning of AC^0 functions, *in* "Proc. Fourth Ann. Workshop on Comp. Learning Theory" (1991), 317-325.

[12] T. Hagerup and C. Rub. A guided tour to Chernoff bounds, *Inf. Proc. Lett.* **33** (1989), 305-308.

[13] T. Hancock and Y. Mansour. Learning monotone k-μ DNF formulas on product distributions, *in* "Proc. 4th Ann. Workshop on Comp. Learning Theory" (1991), 179-183.

[14] J. Håstad. *Computational Limitations for Small Depth Circuits.* Ph.D. thesis, MIT Press, 1986.

[15] J. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution, *J. Comput. Syst. Sci.* **55** (1997), 414-440.

[16] J. Kahn, G. Kalai, and N. Linial. The influence of variables on Boolean functions, *in* "Proc. 29th Ann. Symp. on Found. of Comp. Sci." (1988), 68-80.

[17] M. Kearns, M. Li, L. Pitt, and L. Valiant. On the learnability of Boolean formulae, *in* "Proc. 19th Ann. ACM Symp. on Theory of Computing" (1987), 285-295.

[18] M. Kearns, M. Li, and L. Valiant. Learning boolean formulas, *J. ACM* **41**(6) (1994), 1298-1328.

[19] R. Khardon. On using the Fourier transform to learn disjoint DNF, *Information Processing Letters* **49** (1994), 219-222.

[20] A. Klivans and R. Servedio. Boosting and hard-core sets, *in* "Proc. 40th Ann. Symp. on Found. of Comp. Sci." (1999), 624-633.

[21] N. Linial, Y. Mansour and N. Nisan. Constant depth circuits, Fourier transform and learnability, J. ACM **40**(3) (1993), 607-620.

[22] Y. Mansour. An $O(n^{\log\log n})$ learning algorithm for DNF under the uniform distribution, *J. Comput. Syst. Sci.* **50** (1995), 543-550.

[23] E. Okol'nishnikova. On the influence of negations on the complexity of a realization of monotone Boolean functions by formulas of bounded depth, *Metody Diskret. Analiz.* **38** (1982), 74-80 (in Russian).

[24] Y. Sakai and A. Maruoka. Learning monotone log-term DNF formulas under the uniform distribution, *Theory Comput. Systems* **33** (2000), 17-33. A preliminary version appeared in "Proc. Seventh Conf. on Comp. Learning Theory" (1994), 165-172.

[25] L. G. Valiant. A theory of the learnable, *Comm. ACM* **27**(11) (1984), 1134-1142.

[26] K. Verbeurgt. Learning DNF under the uniform distribution in quasi-polynomial time, *in* "Proc. 3rd Ann. Workshop on Comp. Learning Theory" (1990), 314-326.