On Learning Monotone DNF under Product Distributions

Rocco A. Servedio

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

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, and is the first efficient algorithm for monotone $(\log n)^{\omega(1)}$-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. Verbeurgt [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 $n^{poly(\log n)}$ 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. Verbeurgt [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 ϵ 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 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 ϵ be constant in order to achieve 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 poly($\log n$); this result was later sharpened for DNF formulae by Mansour [27].

Our algorithm extends this approach in the following way: Let $C \subset AC^0$ 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 [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 $\binom{|S_f|}{c}$ 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 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 $f : \{0,1\}^n \to \{-1,1\}$ 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 [12, 16] 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 [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*

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

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 $\mathcal{D}$ 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 $\mathrm{error}(h, f) = \Pr_{\mathcal{D}}[h(\overline{x}) \neq f(\overline{x})]$. 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 $\mathrm{error}(h, f) \leq \epsilon$.

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 $\chi_A : \{0,1\}^n \rightarrow \{-1,1\}$ 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 [12, 26]

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

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

3 Learning under Uniform Distributions

3.1 Identifying Relevant Variables

The following lemma, which is implicit in [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 $f : \{0,1\}^n \to \{-1,1\}$ 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*

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

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

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

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 $|A| \leq c$ 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 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 $f : \{0,1\}^n \to \{-1,1\}$ 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 $\Pr_{\mathcal{U}}[f(\overline{x}) \neq f_1(\overline{x})] < \frac{\epsilon}{32n}$ (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 $I_{\mathcal{U},i}(f_1) = 0$ 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

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

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

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 $i \in A$ for some $i \notin S_f$, inequality (6) implies

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

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 $\epsilon > 0$*

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

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 $d = \log\frac{32tn}{\epsilon}$ 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 $c = 20\log\frac{128t}{\epsilon}\log\frac{8}{\epsilon}$. Mansour's lemma implies that

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

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

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

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

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

where

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

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

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

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

$$Y \leq 2\sum_{|A|>c} (\hat{f}(A) - \hat{f}_2(A))^2 + 2\sum_{|A|>c} \hat{f}_2(A)^2$$

$$\leq 2\sum_{A}(\hat{f}(A) - \hat{f}_2(A))^2 + \epsilon/4$$
$$\leq \epsilon/2$$

by inequalities (8) and (9) respectively.

It remains to bound $Z = \sum_{|A|\leq c, A\subseteq S_f}(\alpha_A - \hat{f}(A))^2$. 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 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 $m = \mathrm{poly}(|S_f|^c, 1/\epsilon, \log(1/\delta))$ 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* `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 1. *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 [12] for learning monotone DNF with $O((\log n)^2/(\log\log n)^3)$ terms also requires that ϵ be constant in order to achieve $\mathrm{poly}(n)$ 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 `LearnMonotone` can be used to learn C. 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.* [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*

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

Taking $m = \mathrm{poly}(r^c, 1/\epsilon, \log(1/\delta))$ and $c = \Theta((\log(M/\epsilon))^d)$ in `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 $\epsilon, \delta > 0$, algorithm* `LearnMonotone` *learns class $C_{d,M,r}$ in time polynomial in n, $r^{(\log(M/\epsilon))^d}$ and $\log(1/\delta)$.*

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 ϵ algorithm* `LearnMonotone` *learns class C_d in 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 Okol'nishnikova [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 $\mathcal{D}$ 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.* [13], 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 1 (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 ([13] Lemma 10)

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

Furst *et al.* [13] 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 [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 ϕ Basis Fourier Lemmas

A *random restriction* $\rho_{p,\mathcal{D}}$ is a mapping from $\{x_1, \ldots, x_n\}$ to $\{0, 1, *\}$ where x_i is mapped to $*$ with probability p, to 1 with probability $(1-p)\mu_i$, and to 0 with probability $(1-p)(1-\mu_i)$. If f is a Boolean function then $f\lceil\rho$ represents 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 $\rho_{p,\mathcal{D}}$.

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

Lemma 4. *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 d literals, and let $\rho_{p,\mathcal{D}}$ be a random restriction. Then with probability at least $1-(4\beta pd)^s$,*

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.*

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

Proof. Linial *et al.* [26] show that if $f\lceil\rho$ satisfies properties (1) and (2) of Lemma 4 then $\widehat{f\lceil\rho}(A)=0$ for all $|A|>s$. Hence such a $f\lceil\rho$ is in the space spanned by $\{\chi_A : |A|\leq s\}$. 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. $\square$

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 $\widetilde{f\lceil\rho}(A)=0$ 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,*

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

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 ϕ 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*

$$\sum_{|A|>c}\tilde{f}(A)^2\leq 2M2^{-c^{1/d}/8\beta}.$$

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 $f : \{0,1\}^n \to \{-1,1\}$ 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*

$$i \in S_f \textit{ implies } \sum_{A:i\in A} \tilde{f}(A)^2 \geq \epsilon/2 \qquad \textit{and} \qquad i \notin S_f \textit{ implies } \sum_{A:i\in A} \tilde{f}(A)^2 \leq \epsilon.$$

The proof uses the fact ([12] Lemma 4.1) that $4\sigma_i^2 I_{\mathcal{D},i}(f) = \sum_{A:i\in A} \tilde{f}(A)^2$ for any Boolean function f and any product distribution $\mathcal{D}$. The algorithm uses sampling to approximate each μ_i (and thus σ_i) and to approximate $I_{\mathcal{D},i}(f)$. We call this algorithm `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 [13] 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 $\langle \overline{x}^1, f(\overline{x}^1)\rangle, \dots, \langle \overline{x}^m, f(\overline{x}^m)\rangle$ from $EX(f,\mathcal{D})$. Compute $\mu_i' = \frac{1}{m}\sum_{j=1}^m x_i^j$ for $1 \leq i \leq n$. 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 $\alpha_A' = \frac{1}{m}\sum_{j=1}^m f(\overline{x}^j)\phi_A'(\overline{x}^j)$. 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 sign$(g(\overline{x}))$, where $g(\overline{x}) = \sum_A \alpha_A' \chi_A(\overline{x})$.

We call this algorithm `LearnMonotone2`. As in [13] 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

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 6 provides the required analogue of Mansour's lemma for product distributions; using the new term size bound on f_2 we obtain $c = \Theta(\beta^2 \log \frac{t}{\epsilon} \log \frac{1}{\epsilon})$.

The one new ingredient in the analysis of `LearnMonotone2` comes in bounding the quantity $Z = \sum_{|A| \le c, A \subseteq S_f} (\alpha'_A - \tilde{f}(A))^2$. 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 [13] shows that taking $m = \mathrm{poly}(c, |S_f|^c, \beta^c, 1/\epsilon, \log(1/\delta))$ suffices. We thus have

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

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

Corollary 4. *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 on Few Variables

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

Theorem 5. *Fix $d \ge 1$ 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 class C in time polynomial in n, $r^{(\beta \log \frac{M}{\epsilon})^d}$ and $\log(1/\delta)$.*

Corollary 5. *Fix $d \ge 1$ 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 ϵ and any constant-bounded product distribution $\mathcal{D}$, algorithm* `LearnMonotone2` *learns class C in 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 enouragement.

References

1. M. Ajtai and Y. Gurevich. Monotone versus positive, *J. ACM* **34**(4) (1987), 1004-1015.
2. P. Beame. A switching lemma primer. Tech. report UW-CSE-95-07-01, University of Washington, November 1994.
3. A. Beimel, F. Bergadano, N. Bshouty, E. Kushilevitz and S. Varricchio. On the applicationf of multiplicity automata in learning, *in* "Proc. 37th Ann. Symp. on Foundations of Computer Science" (1996), 349-358.
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. A. Blum and S. Rudich. Fast learning of k-term DNF formulas with queries, *J. Comp. Syst. Sci.* **51**(3) (1995), 367-373.
8. R. Boppana and M. Sipser. The complexity of finite functions, in *Handbook of Theoretical Computer Science,* vol. A, MIT Press, 1990.
9. N. Bshouty. Exact learning via the monotone theory. *Information and Computation* **123**(1) (1995), 146-153.
10. N. Bshouty. Simple learning algorithms using divide and conquer, *Information Processing Letters*
11. 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.
12. N. Bshouty and C. Tamon. On the Fourier spectrum of monotone functions, *J. ACM* **43**(4) (1996), 747-770.
13. 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.
14. T. Hagerup and C. Rub. A guided tour to Chernoff bounds, *Inf. Proc. Lett.* **33** (1989), 305-308.
15. T. Hancock. The complexity of learning formulas and decision trees that have restricted reads, Ph.D. thesis, Harvard University, TR-15-92 (1992).
16. 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.
17. J. Håstad. *Computational Limitations for Small Depth Circuits.* Ph.D. thesis, MIT Press, 1986.
18. J. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution, *J. Comput. Syst. Sci.* **55** (1997), 414-440.

19. 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.
20. 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.
21. M. Kearns, M. Li, and L. Valiant. Learning boolean formulas, *J. ACM* **41**(6) (1994), 1298-1328.
22. R. Khardon. On using the Fourier transform to learn disjoint DNF, *Information Processing Letters* **49** (1994), 219-222.
23. A. Klivans and R. Servedio. Boosting and hard-core sets, *in* "Proc. 40th Ann. Symp. on Found. of Comp. Sci." (1999), 624-633.
24. L. Kucera, A. Marchetti-Spaccamela and M. Protassi. On learning monotone DNF formulae under uniform distributions, *Inf. and Comput.* **110** (1994), 84-95.
25. E. Kushilevitz. A simple algorithm for learing $O(\log n)$-term DNF. *Information Processing Letters* **61**(6) (1997), 289-292.
26. N. Linial, Y. Mansour and N. Nisan. Constant depth circuits, Fourier transform and learnability, J. ACM **40**(3) (1993), 607-620.
27. Y. Mansour. An $O(n^{\log \log n})$ learning algorithm for DNF under the uniform distribution, *J. Comput. Syst. Sci.* **50** (1995), 543-550.
28. 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).
29. 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.
30. L. G. Valiant. A theory of the learnable, *Comm. ACM* **27**(11) (1984), 1134-1142.
31. K. Verbeurgt. Learning DNF under the uniform distribution in quasi-polynomial time, *in* "Proc. 3rd Ann. Workshop on Comp. Learning Theory" (1990), 314-326.
32. K. Verbeurgt. Learning sub-classes of monotone DNF on the uniform distribution, *in* "Proc. 9th Conf. on Algorithmic Learning Theory" (1998), 385-399.