Learning DNF in Time $2^{\tilde{O}(n^{1/3})}$

Adam R. Klivans[*]
Laboratory for Computer Science
MIT
Cambridge, MA 02139
klivans@math.mit.edu

Rocco A. Servedio[†]
Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
rocco@deas.harvard.edu

Abstract

Using techniques from learning theory, we show that any s-term DNF over n variables can be computed by a polynomial threshold function of degree $O(n^{1/3} \log s)$. This upper bound matches, up to a logarithmic factor, the longstanding lower bound given by Minsky and Papert in their 1968 book *Perceptrons*. As a consequence of this upper bound we obtain the fastest known algorithm for learning polynomial size DNF, one of the central problems in computational learning theory.

[*]Supported in part by NSF grant CCR-97-01304.
[†]Supported in part by NSF grant CCR-95-04436 and by NSF grant CCR-98-77049.

1 Introduction

1.1 Polynomial Threshold Functions

Let f be a Boolean function $f : \{0,1\}^n \rightarrow \{-1,1\}$ and let p be a degree d polynomial in n variables with rational coefficients. If the sign of $p(x)$ equals $f(x)$ for every $x \in \{0,1\}^n$, then we say that f is computed by a *polynomial threshold function* of degree d. In their well known 1968 book *Perceptrons*, Minsky and Papert studied some computational aspects of polynomial threshold functions from an Artificial Intelligence perspective [32]. They proved, among other things, that no polynomial threshold function of degree less than n can compute the parity function on n variables, and that there is a read-once DNF formula which cannot be computed by any polynomial threshold function of degree less than $\Omega(n^{1/3})$. Since then, complexity theorists have used these and related properties of polynomial threshold functions to prove several important results in both circuit and structural complexity [2, 3, 19].

In the computational learning theory community, learning a polynomial threshold function from labeled examples has long been a central problem and continues to be an active area of research. A special focus of attention has been directed toward learning polynomial threshold functions of degree 1, which are known as *linear threshold functions*. The problem of learning a linear threshold function over $\{0,1\}^n$ can be formulated as a linear programming problem and thus can be solved in poly(n) time in both the PAC model of learning from random examples and in the model of exact learning from equivalence queries [10, 31]. Refinements of the basic linear programming approach have led to polynomial-time algorithms for PAC learning linear threshold functions in the presence of classification noise [7, 14]. Much attention has also been given to fast, simple heuristics, most notably the Winnow and Perceptron algorithms, for learning linear threshold functions [12, 18, 25, 29, 34, 35].

1.2 Learning DNF

Another intensively studied problem in computational learning theory, which has met with less success, is the problem of learning DNF formulae. DNF are attractive from a learning theory perspective because of their high expressive power (any Boolean function can be represented as a DNF) and because they seem to be a natural form of knowledge representation for humans. Valiant first posed the question of whether DNF are efficiently learnable in his seminal 1984 paper introducing the PAC learning model [37]; more than fifteen years later this question is widely regarded as one of the most important open problems in learning theory. While many partial results have been given for restricted versions of the DNF learning problem (see e.g. [8, 9, 21, 23, 24, 26, 27, 33, 38, 39]), the difficulty of the unrestricted DNF learning problem is evidenced by the fact that, prior to the current work, only two algorithms were known which improve on the naive 2^n time bound [11, 36].

The first subexponential time algorithm for learning DNF was due to Bshouty [11], who gave an algorithm which learns any s-term DNF over n variables in time $2^{O(\sqrt{n \log s} \log^{3/2} n)}$. At the heart of Bshouty's algorithm is a structural result which shows that that any s-term DNF can be expressed as an $O(\sqrt{n \log n \log s})$-decision list; armed with this result, Bshouty uses a standard algorithm [22] for learning decision lists to obtain his DNF learning result.

Tarui and Tsukiji [36] gave a completely different proof of a similar time bound for learning DNF. They adapted the machinery of "approximate inclusion/exclusion" developed by Linial and Nisan [28] to show that for any s-term DNF f and any distribution $\mathcal{D}$ over $\{0,1\}^n$, there is a conjunction C of size $O(\sqrt{n} \log s)$ which has $|\Pr_{x \in \mathcal{D}}[C(x) = f(x)] - \frac{1}{2}| = 2^{-O(\sqrt{n} \log n \log s)}$. Using this result in conjunction with Freund's "boost-by-majority" algorithm [17], Tarui and Tsukiji obtained an algorithm for learning s-term DNF in time $2^{O(\sqrt{n}\log n \log s)}$.

1.3 A New Approach: Learning DNF via Polynomial Threshold Functions

In this paper we approach the DNF learning problem by representing a DNF formula as a low-degree polynomial threshold function. As we observe in Section 2, we can use known polynomial-time algorithms for learning linear threshold functions to learn polynomial threshold functions of degree d in time $n^{O(d)}$. Thus, upper bounds on the degree of polynomial threshold functions which compute DNF translate directly into bounds on the running time of a DNF learning algorithm.

Viewing DNF formulae as polynomial threshold functions immediately yields a new interpretation of the DNF learning algorithms of Bshouty [11] and Tarui and Tsukiji [36]. Since any r-decision list is equivalent to a polynomial threshold function of degree r [16], in the language of polynomial threshold functions Bshouty's structural result implies that any s-term DNF can be expressed as a polynomial threshold function of degree $O(\sqrt{n \log n \log s})$. In the case of Tarui/Tsukiji, it can be shown as a corollary of their results that any s-term DNF can be expressed as a polynomial threshold function of degree $O(\sqrt{n}\log s)$. Thus, each of these earlier learning algorithms implies an $O(\sqrt{n}\log n)$ upper bound on the degree of a polynomial threshold function for any polynomial-size DNF. A substantial gap still remains, though, between these $O(\sqrt{n}\log n)$ upper bounds and the $\Omega(n^{1/3})$ lower bound due to Minsky and Papert.[1]

1.4 Our Results

Our first result is the following theorem:

Theorem 1 *Any s-term DNF over $\{0,1\}^n$ in which each conjunction is of size at most t can be expressed as a polynomial threshold function of degree $O(\sqrt{t}\log s)$.*

A useful feature of Theorem 1 is that the degree bound depends on $\sqrt{t}$ which can be much smaller than $\sqrt{n}$. Close inspection of the results due to Tarui/Tsukiji reveal that a similar theorem can be derived from their analysis. An advantage of our proof (which is self-contained and does not use approximate inclusion-exclusion or boosting) is that it highlights this dependence which plays a crucial role in our later results.

We then use Theorem 1 to give several new results about the degree of polynomial threshold functions which compute various classes of Boolean formulas.

By combining Theorem 1 with a decomposition technique due to Bshouty [11] we obtain our main result:

Theorem 2 *Any s-term DNF over $\{0,1\}^n$ can be expressed as a polynomial threshold function of degree $O(n^{1/3}\log s)$.*

Theorem 2 essentially closes the gap which was left open by the $O(\sqrt{n}\log n)$ upper bounds implicit in [11, 36]; it shows that the Minsky-Papert lower bound is in fact tight, up to a logarithmic factor, for *all* polynomial-size DNF. Theorem 3 also yields a $2^{O(n^{1/3}\log^2 n)}$-time algorithm for learning polynomial-size DNF, which improves on the algorithms of Bshouty and Tarui/Tsukiji and is the fastest known algorithm for the unrestricted DNF learning problem.

We can improve upon the bounds of Theorem 2 for read-once DNF:

[1] Beigel et al. stated in [5] that Minsky and Papert gave an $\Omega(\sqrt{n})$ lower bound for DNF but this was in error [4].

Theorem 3 *Any read-once DNF over* $\{0,1\}^n$ *can be expressed as a polynomial threshold function of degree* $O(n^{1/3}\log^{2/3} n)$.

Finally, we would like (but are currently unable) to prove similar upper bounds on the degree of polynomial threshold functions which compute arbitrary AC^0 functions. As a step in this direction, we prove

Theorem 4 *For* $d \geq 3$, *any read-once Boolean formula of depth* d *over* $\{\wedge, \vee, \neg\}$ *can be computed by a polynomial threshold function of degree* $\tilde{O}(n^{1-\frac{1}{3\cdot 2^{d-3}}})$.

Theorem 4 implies that the class of read-once AC^0 formulas can be learned in subexponential time.

2 Preliminaries

2.1 DNF, Decision Lists, Decision Trees, and Polynomial Threshold Functions

A *disjunctive normal form* formula or DNF is a disjunction $T_1 \vee \cdots \vee T_s$ of conjunctions of Boolean literals. An s-term DNF is one which has at most s conjunctions (also known as terms) and a t-DNF is one in which each term is of size at most t. A DNF (or Boolean formula) is *read-once* if it contains at most one occurrence of each variable.

A *k-decision list* is a list $L = (T_1, f_1), \ldots, (T_m, f_m)$ where each T_i is a term of size at most k and each f_i is a Boolean function on $\{0,1\}^n$. Given an input $x \in \{0,1\}^n$ the value of $L(x)$ is $f_j(x)$ where $j \geq 1$ is such that $T_j(x) = 1$ and $T_i(x) = 0$ for $i < j$. If $T_i(x) = 0$ for all $1 \leq i \leq m$ then $L(x) = 1$.

A *k-decision tree* is a rooted binary tree where each internal node has 2 children and is labeled with a term of size at most k and each leaf is labeled with a Boolean function. A decision tree represents a Boolean function as follows: if the root is labeled with a term T then then to compute the value of the tree on an input $x \in \{0,1\}^n$ we go left from the root if $T(x) = 0$ and go right if $T(x) = 1$. We continue in this fashion until reaching a leaf ℓ labeled with some function f_ℓ and then output $f_\ell(x)$.

The *rank* of a decision tree T is defined inductively as follows:

- If T is a single leaf then $rank(T) = 0$.
- If T has subtrees T_0 and T_1 then

$$rank(T) = \begin{cases} \max\left(rank(T_0), rank(T_1)\right) & \text{if } rank(T_0) \neq rank(T_1) \\ rank(T_0) + 1 & \text{otherwise.} \end{cases}$$

The following lemma will be useful:

Lemma 5 *[6] Let* f *be computed by a 1-decision tree of rank* r *whose leaves are labeled with the functions* $f_1, \ldots, f_m$. *Then there is an* r*-decision list* $(T_1, f_1), \ldots, (T_m, f_m)$ *which is equivalent to* f.

A *polynomial threshold function* is defined by a multivariate polynomial $p(x_1, \ldots, x_n)$. The output of the polynomial threshold function on input $x \in \{0,1\}^n$ is 1 if $p(x_1, \ldots, x_n) \geq 0$ and is -1 otherwise. The *degree* of a polynomial threshold function is simply the degree of the polynomial p. If each coefficient a_α of the polynomial is an integer, then the *weight* of the polynomial threshold function is $\sum |a_\alpha|$.

2.2 Learning theory background

We consider two widely studied learning models: the *Probably Approximately Correct* (PAC) model introduced by Valiant [37] and the model of *exact learning from equivalence queries* introduced by Angluin [1] and Littlestone [29]. In each of these models a *concept class C* is a collection of Boolean functions $c : \{0,1\}^n \rightarrow \{-1,1\}$.

In the PAC model, for Boolean functions c, h on $\{0,1\}^n$ and $\mathcal{D}$ a distribution on $\{0,1\}^n$, we say that h is an *ϵ-approximator for c under $\mathcal{D}$* if $\Pr_{\mathcal{D}}[c(x) = h(x)] \geq 1 - \epsilon$. The learning algorithm has access to an *example oracle $EX(c, \mathcal{D})$* which, when queried, provides a labeled example $\langle x, c(x) \rangle$ where x is drawn from $\{0,1\}^n$ according to the distribution $\mathcal{D}$ and $c \in C$ is the unknown target concept which the algorithm is trying to learn. The goal of the learner is to generate an ϵ-approximator for c under $\mathcal{D}$. An algorithm A is a *PAC learning algorithm for a concept class C* if the following condition holds: for any $c \in C$, any distribution $\mathcal{D}$ on $\{0,1\}^n$, and any $0 < \epsilon, \delta < 1$, if A is given ϵ and δ and has access to $EX(c, \mathcal{D})$, then with probability at least $1 - \delta$ algorithm A outputs an ϵ-approximator for c under $\mathcal{D}$.

In the model of exact learning from equivalence queries, learning proceeds in a sequence of stages. In each stage the learning algorithm submits an *equivalence query* (a Boolean function h) to the teacher. If h is equivalent to the target concept c then the teacher answers "YES" and learning halts; otherwise the teacher sends back a point $x \in \{0,1\}^n$ such that $h(x) \neq c(x)$. A learning algorithm A *learns concept class C in time t* if for all $c \in C$, algorithm A can exactly identify the target c in at most t time steps, using at most t equivalence queries, with hypotheses h which each can be represented with t bits and can be evaluated on any point $x \in \{0,1\}^n$ in time t.

The following fact is well known:

Fact 6 ([10, 31]) *In both the PAC model and the model of exact learning from equivalence queries, there are algorithms which learn the class of linear threshold functions over $\{0,1\}^n$ in time poly(n).*

The algorithms of Fact 6 are based on polynomial time linear programming. We will need the following extension of Fact 6:

Fact 7 *Let C be a class of functions each of which can be expressed as an degree-d polynomial threshold function over $\{0,1\}^n$. Then in both the PAC learning model and the model of exact learning from equivalence queries, there is a learning algorithm for C which runs in time $n^{O(d)}$.*

Proof sketch: The idea is to run a polynomial-time algorithm for learning linear threshold functions over an expanded version of the input space. Since $z^2 = z$ for $z \in \{0,1\}$ we can suppose without loss of generality that the target polynomial threshold function is a multilinear polynomial of degree d. Such a polynomial threshold function can be viewed as a linear threshold function over the space of all multilinear monomials of degree at most d. There are $N = \sum_{i=1}^{d} \binom{n}{i} \leq n^d$ such monomials and hence by Fact 6 we can learn such a polynomial threshold function by running a poly(N)-time algorithm for learning linear threshold functions over the domain $\{0,1\}^N$ where $N \leq n^d$. ■

2.3 The Minsky Papert Lower Bound

It is clear that any depth-1 circuit over $\{\wedge, \vee, \neg\}$ can be expressed as a linear threshold function. In contrast, Minsky and Papert gave a $\Omega(n^{1/3})$ lower bound on the degree of any polynomial threshold function which computes a particular read-once DNF. For completeness we give their simple proof.

Theorem 8 (Minsky & Papert [32]) *Let $f = T_1 \vee \cdots \vee T_m$ be an m-term DNF over $\{0,1\}^n$ where each term T_i is a conjunction over $4m^2$ variables, each variable appears in precisely one term, and $n = 4m^3$. Then any polynomial threshold function which computes f must have degree at least m.*

Proof: Let $p(x_1, \ldots, x_n)$ be a polynomial of degree d such that for all $x \in \{0,1\}^n$ we have $p(x) \geq 0$ iff x satisfies f. For $i = 1, \ldots, m$ let S_i be the set of $4m^2$ variables which appears in term T_i. It is clear that for any permutations $\pi_1, \ldots, \pi_m$ over a set of size $4m^2$, we have $p(S_1, \ldots, S_m) \geq 0$ iff $p(\pi_1(S_1), \ldots, \pi_m(S_m)) \geq 0$. Consequently the polynomial

$$q(x_1, \ldots, x_n) = \sum_{\pi_1, \ldots, \pi_m} p(\pi_1(S_1), \ldots, \pi_m(S_m))$$

is of degree at most d and has $q(x_1, \ldots, x_n) \geq 0$ iff x satisfies f. Since $q(x)$ is symmetric in the elements of each set S_i, one can straightforwardly show that there is a polynomial $r(\sum_{S_1} x_j, \cdots, \sum_{S_m} x_j)$ of degree at most d such that $r(\sum_{S_1} x_j, \ldots, \sum_{S_m} x_j) = q(x_1, \ldots, x_n)$ for all $x \in \{0,1\}^n$. It follows from the definition of f that for all $(a_1, \ldots, a_m) \in \{0, 1, \ldots, 4m^2\}^m$, we have $r(a_1, \ldots, a_m) \geq 0$ iff some $a_i = 4m^2$. Let $s(t)$ be the univariate polynomial $r(a_1, \ldots, a_m)$ where $a_i = 4m^2 - (t - (2i-1))^2$ for $i = 1, \ldots, m$. Then the degree of s is at most $2d$, and moreover we have $s(0), s(2), s(4), \ldots, s(2m) < 0$ and $s(1), s(3), \ldots, s(2m-1) \geq 0$. Consequently s has at least $2m$ real zeros, so $2d \geq \deg(s) \geq 2m$. ∎

3 An Optimal Bound for Representing DNF by Polynomial Threshold Functions

In this section we prove our main result: any s-term DNF over $\{0,1\}^n$ can be computed by a polynomial threshold function of degree $O(n^{1/3} \log s)$.

3.1 Low-Degree Polynomial Threshold Functions for DNF with Small Terms

We start by proving Theorem 1:

Theorem 1 *Any s-term t-DNF can be expressed as a polynomial threshold function of degree $O(\sqrt{t} \log s)$.*

This theorem plays an important role in the proof of the main result. We discuss some other consequences of Theorem 1 in Section 4.

Proof of Theorem 1: Let $f = T_1 \vee T_2 \vee \cdots \vee T_s$ be an s-term t-DNF. The *arithmetization* of a Boolean literal ℓ is x_j if $\ell = x_j$ and is $1 - x_j$ if $\ell = \overline{x}_j$. Let S_i denote the sum of the arithmetizations of the literals appearing in T_i and let t_i denote the number of literals in T_i. We define the polynomial

$$Q_i(x) = p\left(\frac{S_i}{t_i}\right)$$

where

$$p(y) = C_d\left(y\left(1 + \frac{1}{t}\right)\right).$$

Here C_d is the d-th Chebyshev polynomial of the first kind and $d = \lceil \sqrt{t} \rceil$.

Consider the polynomial threshold function "$P(x) \geq s + \frac{1}{2}$" where

$$P(x) = \sum_{i=1}^{s} Q_i(x)^{\log 2s}.$$

Since C_d is a polynomial of degree $d = \sqrt{t}$ and S_i is a polynomial of degree 1, this polynomial threshold function has degree $\sqrt{t}\log 2s$. We will show that this polynomial threshold function computes the DNF f exactly.

The following basic facts about the Chebyshev polynomials C_d are well known [13]:

- $|C_d(x)| \leq 1$ for $|x| \leq 1$ with $C_d(1) = 1$;
- $C'_d(x) \geq d^2$ for $x > 1$ with $C'_d(1) = d^2$.

These facts imply that $p(1) \geq 2$ but $|p(y)| \leq 1$ for $y \in [0, 1 - \frac{1}{t}]$.

Fix any element $x \in \{0,1\}^n$.

- If $f(x) = 0$ then in each term T_i at least one arithmetized literal takes value 0 on x. Thus for each $i = 1, \ldots, s$ we have $S_i/t_i \leq (t_i - 1)/t_i \leq 1 - \frac{1}{t}$ and hence each $|Q_i(x)| \leq 1$. Consequently $P(x) \leq s$.
- If $f(x) = 1$ then some term T_i must be satisfied by x so $S_i/t_i = 1$. Consequently $Q_i(x) \geq 2$ and hence $Q_i(x)^{\log 2s}$ contributes at least $2s$ to $P(x)$. Since $Q_i(x)^{\log 2s} \geq -1$ for all i, we have $P(x) \geq s + 1$. ■

3.2 From DNF to Decision Trees

Let f be an arbitrary s-term DNF over n variables. As the first step in our construction of a polynomial threshold function for f, we transform f into a 1-decision tree in which each leaf is a DNF with small terms; this is a refinement of a transformation given by Bshouty in [11]. Our original proof gave a bound on the *size* of the resulting decision tree. S. Lokam [30] has observed that a slightly stronger bound can be obtained by considering the *rank* of the decision tree instead. We use Lokam's approach in the following lemma:

Lemma 9 *Let $f : \{0,1\}^n \rightarrow \{-1,1\}$ be an s-term DNF. For any value $1 \leq t \leq n$, f can be expressed as a 1-decision tree T where*

- *each leaf of T contains an s-term t-DNF,*
- *T has rank at most $(2n/t)\log s + 1$.*

Proof of Lemma 9: Let $T_1, \ldots, T_p$ be the terms of f that have size at least t. Since each term T_i contains at least t literals, there must be some variable x_i that occurs (either negated or unnegated) in at least pt/n of these terms. This variable x_i is placed in the root of the decision tree, and the left and right children of x_i will be decision trees for the restrictions $f|_{x_i \leftarrow 0}$ and $f|_{x_i \leftarrow 1}$ respectively. This construction is recursively carried out for each of the functions $f|_{x_i \leftarrow 0}$ and $f|_{x_i \leftarrow 1}$, stopping when a DNF with no terms larger than t is obtained.

It is clear that this recursive procedure generates some 1-decision tree T. Since the function obtained by fixing some subset of variables of an s-term DNF is an s-term DNF, we have that each leaf of T contains an s-term t-DNF.

Let $r(n,p)$ be the maximum (taken over all DNFs f on n variables with p terms having size at least t) rank of the decision tree generated by the above procedure. We bound $r(n,p)$ using the following simple observation: if T_a is a term of f which contains an unnegated (negated) variable x_i ($\overline{x}_i$), then the restriction $f|_{x_i \leftarrow 0}$ ($f|_{x_i \leftarrow 1}$) causes the term T_a to vanish. Since the variable x_i at the root of T occurs in at least pt/n terms of size at least t, for at least one of the bit values $b \in \{0,1\}$ the restriction $f|_{x_i \leftarrow b}$ will be a DNF which has at most $p(1-\frac{t}{2n})$ terms of size at least t. Let T_0 (T_1) denote the subtree of T which corresponds to the restriction $f|_{x_i \leftarrow 0}$ ($f|_{x_i \leftarrow 1}$), and suppose without loss of generality that $f|_{x_i \leftarrow 0}$ is a s-term DNF which has at most $p(1-\frac{t}{2n})$ terms of size at least t. Note that $rank(T_0) \le r(n-1, p(1-\frac{t}{2n}))$ and $rank(T_1) \le r(n-1,p)$. We consider several cases:

- If $rank(T_0) < rank(T_1)$, then $rank(T) = rank(T_1)$ and hence $r(n,p) \le r(n-1,p)$.
- If $rank(T_0) > rank(T_1)$, then $rank(T) = rank(T_0)$ and hence $r(n,p) \le r(n-1, p(1-\frac{t}{2n}))$.
- If $rank(T_0) = rank(T_1)$, then $rank(T) = rank(T_0)+1$ and hence $r(n,p) \le r(n-1, p(1-\frac{t}{2n}))+1$.

To establish initial conditions for the recurrence relation we consider the case $p=1$. In this case there is one term in f which contains more than t variables; without loss of generality we suppose that this term is $v_1 v_2 \dots v_\ell$. Then the 1-decision list

$$(\overline{v}_1, f|_{v_1 \leftarrow 0}), \dots, (\overline{v}_\ell, f|_{v_\ell \leftarrow 0})$$

is equivalent to a rank-1 decision tree in which each leaf contains an s-term t-DNF. Hence for any n we have $r(n,1)=1$.

Solving this easy recurrence relation for $r(n,p)$ shows that $r(n,p) \le (2n/t)\ln p + 1$. Since $p \le s$ the theorem is proved.

3.3 An Optimal Bound for Representing DNF by Polynomial Threshold Functions

Theorem 2 *Let f be an s-term DNF over n variables. Then f can be expressed as a polynomial threshold function of degree $O(n^{1/3}\log s)$.*

Proof: From Lemma 9 and Theorem 1, we know that f can be expressed as a 1-decision tree T of rank $(2n/t)\ln s + 1$ where each leaf contains a polynomial threshold function of degree $O(\sqrt{t}\log s)$ (the value of t will be fixed later). From Lemma 5 we know that this decision tree T can be expressed as an r-decision list where $r = (2n/t)\ln s + 1$ and each output of the decision list is a polynomial threshold function of degree $O(\sqrt{t}\log s)$. Call this decision list L.

Let $C_1, \dots, C_R$ be the conjunctions contained in the successive nodes of L and let $P_1(x), \dots, P_R(x)$ be the corresponding polynomials for the associated polynomial threshold functions at the outputs, i.e. the polynomial threshold function corresponding to the j-th conjunction C_j computes the function "$P_j(x) \ge 0$." If $P_j(x) = 0$ for some $x \in \{0,1\}^n$ then we can replace $P_j(x)$ by $P_j(x) + \delta/2$, where $\delta = \min\{-P_j(x) \ : \ x \in \{0,1\}^n \text{ and } P_j(x) < 0\}$, without changing the function computed by the polynomial threshold function. Now by scaling each P_j by an appropriate multiplicative factor we can suppose without loss of generality that for each $j = 1, \dots, R$ we have $\min_{x \in \{0,1\}^n} |P_j(x)| \ge 1$.

Consider the polynomial

$$Q(x) = A_1 \tilde{C}_1(x) P_1(x) + A_2 \tilde{C}_2(x) P_2(x) + \cdots + A_R \tilde{C}_R(x) P_R(x). \qquad (1)$$

Here $\tilde{C}_j$ is the zero/one valued polynomial which corresponds to the monomial C_j (e.g. if C_j is $x_3\overline{x}_4x_5$ then $\tilde{C}_j(x)$ is $x_3(1-x_4)x_5$). Each value A_j is a positive constant chosen so as to satisfy the following conditions:

$$\begin{aligned}
A_R &= 1, \\
A_{R-1} &> \max_{x\in\{0,1\}^n} |A_R\tilde{C}_R(x)P_R(x)|, \\
&\vdots \\
A_j &> \max_{x\in\{0,1\}^n} |A_{j+1}\tilde{C}_{j+1}(x)P_{j+1}(x) + \cdots + A_R\tilde{C}_R(x)P_R(x)|. \\
&\vdots \\
A_1 &> \max_{x\in\{0,1\}^n} |A_2\tilde{C}_2(x)P_2(x) + \cdots + A_R\tilde{C}_R(x)P_R(x)|.
\end{aligned}$$

Then the polynomial threshold function "$Q(x) \geq 0$" computes exactly the same function as the decision list L. To see this, fix an input $x \in \{0,1\}^n$. If j is the index of the first conjunction C_j which is satisfied by x, then $\tilde{C}_1(x) = \tilde{C}_2(x) = \cdots = \tilde{C}_{j-1}(x) = 0$, so the only terms of (1) which make a nonzero contribution to Q are $A_i\tilde{C}_i(x)P_i(x)$ for $i \geq j$. Since $\tilde{C}_j(x) = 1$ and $|P_j(x)| \geq 1$, the choice of A_j ensures that the sign of $Q(x)$ will be the same as the sign of $P_j(x)$.

The degree of the polynomial $Q(x)$ is at most $(2n/t)\ln s + 1 + O(\sqrt{t}\log s)$. If we take $t = n^{2/3}$ then this value is $O(n^{1/3}\log s)$. ■

Applying Fact 7 gives our main DNF learning result:

Corollary 10 *The class of polynomial-size DNF can be learned (in both the PAC model and the model of exact learning from equivalence queries) in time* $2^{O(n^{1/3}\log^2 n)}$.

Remark: Several algorithms are known [7, 14] for PAC learning linear threshold functions over $\{0,1\}^n$ in the presence of classification noise in time poly(n). It follows that our time bounds for learning DNF continue to hold in the presence of classification noise.

Corollary 11 *The* $\Omega(n^{1/3})$ *lower bound given by Minsky and Papert for the degree of a polynomial threshold function required to compute a polynomial size DNF is tight up to a logarithmic factor.*

4 Discussion

Since $t \leq n$ in Theorem 1, Fact 7 implies that there is a linear-programming based algorithm for PAC learning DNF which takes $2^{O(\sqrt{n}\log n\log s)}$ time steps. Tarui and Tsukiji gave an identical time bound for a different algorithm based on hypothesis boosting using conjunctions. In this section we note that the proof of Theorem 1 gives an upper bound on the weight of the resulting polynomial threshold function. This observation can be used to prove correctness of the Tarui/Tsukiji boosting-based algorithm and to show that simpler algorithms such as Winnow or Perceptron can be used to learn $\tilde{O}(\sqrt{n})$ degree polynomial threshold functions which compute a DNF (instead of boosting algorithms or algorithms for solving linear programs).

The d-th Chebyshev polynomial $C_d(x) = \sum_{i=0}^{d} a_i x^i$ has all integer coefficients with each $|a_i| \leq 2^d$ [13]. By inspection of the proof of Theorem 1 we obtain

Corollary 12 *Any s-term t-DNF can be expressed as a polynomial threshold function of degree $O(\sqrt{t}\log s)$ and weight $t^{O(\sqrt{t}\log s)}$.*

Using this corollary we obtain an easy proof of one of the main theorems from [36], described in Section 1.2, which asserts that for any DNF f and any probability distribution $\mathcal{D}$ there exists some short conjunction which is noticeably correlated with f under $\mathcal{D}$. We use a simple lemma due to Goldmann, Hastad and Razborov ([20] Lemma 4) which states that if a function f over $\{0,1\}^n$ can be expressed as a majority of at most W ± 1-valued functions (possibly with repetitions) drawn from a set H, then for any distribution $\mathcal{D}$ over $\{0,1\}^n$ there is some function $h \in H$ such that $|\Pr_{x\in\mathcal{D}}[h(x) = f(x)] - \frac{1}{2}| \geq \frac{1}{W}$. In our setting we take H to be the set of all conjunctions of length $O(\sqrt{t}\log s)$ and their negations. There is a clear correspondence between polynomial threshold functions with integer coefficients and depth-2 circuits with a MAJORITY gate at the root and (possibly negated) AND gates at depth 1. Corollary 12 gives the required bound on W, and we obtain

Corollary 13 *Given any s-term t-DNF f and any distribution $\mathcal{D}$ over $\{0,1\}^n$, there is a conjunction C of size at most $O(\sqrt{t}\log s)$ such that $|\Pr_{x\in\mathcal{D}}[C(x) = f(x)] - \frac{1}{2}| = 2^{-O(\sqrt{t}\log t \log s)}$.*

Taking $t = n$ gives Tarui and Tsukiji's Theorem 1.1, which immediately implies the existence of a boosting-based algorithm for learning DNF in time $2^{\tilde{O}(n^{1/2})}$.

Finally, we observe that the weight bound given in Corollary 12 implies that we do not need to solve linear programs (or even to use boosting algorithms) in order to learn polynomial-sized DNF in time $2^{\tilde{O}(\sqrt{n})}$. If f is a polynomial threshold function of degree 1 and weight W over the domain $\{0,1\}^N$, then either the Perceptron algorithm or the Winnow algorithm can be used to learn f in $\mathrm{poly}(N, W)$ time steps [25, 29]. As in Fact 7, we can view an degree-d polynomial threshold function over $\{0,1\}^n$ as a degree-1 polynomial threshold function over $\{0,1\}^{n^d}$, and thus we can in fact use either the Perceptron or Winnow algorithm to learn s-term DNF in time $2^{O(\sqrt{n}\log n \log s)}$.

5 Low-Degree Polynomial Threshold Functions for Read-Once DNF

As seen in Section 2.3 the Minsky-Papert $\Omega(n^{1/3})$ lower bound on polynomial threshold function degree for polynomial size DNF is proved using a read-once DNF. Since any read-once DNF can have at most n terms, Theorem 2 implies that any read-once DNF can be expressed as a polynomial threshold function of degree $O(n^{1/3}\log n)$. Here we give a slightly better bound:

Theorem 3 *Any read-once DNF over variables $x_1, \ldots, x_n$ can be expressed as a polynomial threshold function of degree $O(n^{1/3}\log^{2/3} n)$.*

To prove Theorem 3 we use the following sharper version of Lemma 9:

Lemma 14 *Let $f : \{0,1\}^n \to \{-1,1\}$ be a read-once DNF. For any value $1 \leq t \leq n$, f can be expressed as a 1-decision tree T where each leaf of T contains a read-once t-DNF and T has rank at most n/t.*

Proof of Lemma 14: Let $T_1, \ldots, T_p$ be the terms of f that have size at least t. We use the same decomposition procedure as in Lemma 9, and we let $r(n,p)$ be the maximum (taken over all read-once DNFs f on n variables with p terms having size at least t) rank of the decision tree generated by the decomposition procedure. Since each variable occurs in at most one term, the recurrence which we obtain in this setting is $r(n,p) \leq r(n-1, p-1) + 1$. As before the initial condition is

$r(n,1) = 1$ for all n, and thus $r(n,p) \leq p$. Since f is read-once we have that $p \leq n/t$, and the lemma is proved. (Lemma 14) ■

Proof of Theorem 3: Let f be a s-term read-once DNF over $\{0,1\}^n$. Lemma 14, Theorem 1 and Lemma 5 together imply that f is computed by a (n/t)-decision list where each output of the decision list is a polynomial threshold function of degree $O(\sqrt{t}\log s)$. As in the proof of Theorem 2 there is a polynomial threshold function for f which is of degree $n/t + O(\sqrt{t}\log s)$. Since f is read-once s is at most n, and taking $t = n^{2/3}/\log^{2/3} n$ proves the theorem. (Theorem 3) ■

By the arguments given in Section 2, we immediately have

Corollary 15 *The class of read-once DNF can be learned (in both the PAC model and the model of exact learning from equivalence queries) in time* $2^{O(n^{1/3}\log^{5/3} n)}$.

Remark: Standard reductions are known in learning theory which reduce the problem of PAC learning DNF to that of PAC learning read-once DNF. We note that applying these reductions here does *not* yield a $2^{\tilde{O}(n^{1/3})}$-time algorithm for learning arbitrary polynomial-size DNF. The reductions work by converting a DNF with $p(n)$ total occurrences of variables to a read-once DNF over $p(n)$ variables, and thus if used in conjunction with our theorem would yield a $2^{\tilde{O}(p(n)^{1/3})}$-time algorithm for learning such a DNF.

6 Future Work

Many directions remain for further research. From a learning theory perspective, an obvious goal is to construct learning algorithms for DNF which have even lower time complexity than the algorithm of this paper. The Minsky-Papert lower bound implies that our time bounds are essentially optimal for algorithms which work by learning polynomial threshold functions. It would be interesting to close the remaining gap between the Minsky-Papert $\Omega(n^{1/3})$ lower bound and our $O(n^{1/3}\log n)$ upper bound on the degree of polynomial threshold functions for polynomial-size DNF.

Another goal is to establish a bound on polynomial threshold function *weight* to go along with our degree bound from Theorem 3. Is every polynomial-size DNF computed by a polynomial threshold function of degree $\tilde{O}(n^{1/3})$ and weight $2^{\tilde{O}(n^{1/3})}$? As in Section 4, an affirmative answer to this question would mean that the Perceptron or Winnow algorithm could be used instead of a linear programming based algorithm.

From a circuit complexity perspective, an interesting goal is to obtain results analogous to our upper bound (and to the Minsky-Papert lower bound) for polynomial-size circuits of depth greater than 2. What upper and lower bounds can be established for the degree of polynomial threshold functions which compute arbitrary AC^0 functions? As a step towards answering this question, we show how the techniques of this paper can be used to obtain a nontrivial upper bound on the degree of polynomial threshold functions for read-once AC^0 functions:

Theorem 4 *For* $d \geq 2$, *any read-once Boolean formula of depth* d *over* $\{\wedge, \vee, \neg\}$ *can be computed by a polynomial threshold function of degree* $O(n^{1-\frac{1}{3\cdot 2^{d-3}}} \log^{\frac{1}{3\cdot 2^{d-3}}} n)$.

Proof: The proof is by induction on d. The base case $d = 2$ is supplied by Theorem 3. We suppose that the theorem holds for $d = 2, \ldots, k-1$ and prove it for $d = k$.

Let f be a depth-k read-once formula. We say that a *term* is a gate at the bottom level of f together with the literals that feed into it. Since f is read-once there can be at most n/t terms of size greater than t. We apply the decomposition procedure described in the proof of Lemma 9 to transform f into a 1-decision tree whose leaves each contain a depth-k read-once formula in which each term is of size at most t. As in Lemma 14 this decision tree is of rank at most n/t.

In each leaf of this tree, we replace each term with a new "dummy" variable that appears only once. We thus obtain a decision tree of rank n/t whose leaves each contain a read-once formula of depth $k-1$ over these dummy variables. By the induction hypothesis, each such formula is equivalent to a polynomial threshold function of degree $O(n^{1-\frac{1}{3\cdot 2^{k-4}}}\log^{\frac{1}{3\cdot 2^{k-4}}} n)$ which is defined over the dummy variables described above.

In each such polynomial threshold function, we now replace each dummy variable with a real-valued polynomial over the original variables which interpolates precisely the Boolean function computed by the original term. Since each term was of size at most t, each such polynomial is of degree at most t. Consequently the function computed at each leaf of the decision tree is a polynomial threshold function of degree $O(tn^{1-\frac{1}{3\cdot 2^{k-4}}}\log^{\frac{1}{3\cdot 2^{k-4}}} n)$.

As in the proof of Theorem 3, our original function f can now be expressed as a polynomial threshold function of degree

$$\frac{n}{t} + O(tn^{1-\frac{1}{3\cdot 2^{k-4}}}\log^{\frac{1}{3\cdot 2^{k-4}}} n).$$

Taking $t = n^{\frac{1}{3\cdot 2^{k-3}}}/\log^{\frac{1}{3\cdot 2^{k-3}}} n$ proves the theorem. ■

Corollary 16 *The class of read-once AC^0 functions can be learned in subexponential time.*

7 Acknowledgements

We thank Richard Beigel for several useful conversations and Les Valiant for his advice. We also thank S. Lokam for allowing us to include his proof of Lemma 9.

References

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

[2] J. Aspnes, R. Beigel, M. Furst and S. Rudich. The expressive power of voting polynomials. *Combinatorica* **14**:2 (1994), 1-14. Earlier version in "Proc. 23rd ACM Symposium on Theory of Computation" (1991), 402-409.

[3] R. Beigel. The polynomial method in circuit complexity, in "Proc. 8th Conf. on Structure in Complexity Theory" (1993), 82-95.

[4] R. Beigel, personal communication, 2000.

[5] R. Beigel, N. Reingold and D. Spielman. The perceptron strikes back, *in* "Proc. 6th Conf. on Structure in Complexity Theory" (1991), 286-291.

[6] A. Blum. Rank-r decision trees are a subclass of r-decision lists. *Information Processing Letters* **42**:4 (1992), 183-185.

[7] A. Blum, A. Frieze, R. Kannan, and S. Vempala. A polynomial time algorithm for learning noisy linear threshold functions, *in* "Proc. 37th Symp. on Found. of Comp. Sci." (1996), 330-338.

[8] 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.

[9] A. Blum and S. Rudich. Fast learning of k-term DNF formulas with queries. *J. Comp. Syst. Sci.* **51**(3) (1995), 367-373.

[10] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM* **36**:4 (1989), 929-965.

[11] N. Bshouty. A subexponential exact learning algorithm for DNF using equivalence queries. *Information Processing Letters* **59** (1996), 37-39.

[12] T. Bylander. Worst-case analysis of the perceptron and exponentiated update algorithms. *Artificial Intelligence* **106** (1998).

[13] E. W. Cheney. *Introduction to approximation theory.* McGraw-Hill, 1966.

[14] E. Cohen. Learning noisy perceptrons by a perceptron in polynomial time, *in* "Proc. 38th Symp. on Found. of Comp. Sci." (1997), 514-523.

[15] A. Ehrenfeucht and D. Haussler. Learning decision trees from random examples. *Information and Computation* **82**:3 (1989), 231-246.

[16] A. Ehrenfeucht, D. Haussler, M. Kearns and L. Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation* **82**:3 (1989), 247-251.

[17] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation* **121**:2 (1995), 256-285.

[18] Y. Freund and R. Schapire. Large margin classification using the perceptron algorithm, *in* "Proc. Eleventh Ann. Conf. on Comp. Learning Theory" (1998), 209-217.

[19] B. Fu. Separating PH from PP by relativization. *Acta Math. Sinica* **8**:3 (1992), 329-336.

[20] M. Goldmann, J. Håstad and A. Razborov. Majority gates vs. general weighted threshold gates. *Computational Complexity* **2** (1992), 277-300.

[21] 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.

[22] D. Helmbold, R. Sloan and M. Warmuth. Learning nested differences of intersection-closed concept classes. *Machine Learning* **5** (1990), 165-196.

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

[24] R. Khardon. On using the Fourier transform to learn disjoint DNF. *Inf. Proc. Lett.* **49** (1994), 219-222.

[25] J. Kivinen, M. Warmuth, and P. Auer. The perceptron algorithm vs. winnow: linear vs. logarithmic mistake bounds when few input variables are relevant, *in* "Proc. 8th Conf. on Computational Learning Theory," (1995), 289-296.

[26] L. Kucera, A. Marchetti-Spaccamela and M. Protassi. On learning monotone DNF formulae under uniform distributions. *Inf. and Comput.* **110** (1994), 84-95.

[27] E. Kushilevitz and D. Roth. On learning visual concepts and DNF formulae, *in* "Proc. Sixth Ann. ACM Conference on Computational Learning Theory" (1993), 317-326.

[28] N. Linial and N. Nisan. Approximate inclusion-exclusion. *Combinatorica* **10**:4 (1990), 349-365.

[29] N. Littlestone. Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning* **2** (1988), 285-318.

[30] S. Lokam, personal communication (2001).

[31] W. Maass and G. Turan. How fast can a threshold gate learn? *in* "Computational Learning Theory and Natural Learning Systems: Volume I: Constraints and Prospects," S. J. Hanson, G. Drastal, & R. Rivest, eds., MIT Press (1994), 381-414.

[32] M. Minsky and S. Papert. *Perceptrons.* MIT Press, 1968 (expanded edition 1988).

[33] Y. Sakai and A. Maruoka. Learning monotone log-term DNF formulas under the uniform distribution. *Theory Comput. Systems* **33** (2000), 17-33.

[34] M. Schmitt. Identification criteria and lower bounds for Perceptron-like learning rules. *Neural Computation* **10** (1998), 235-250.

[35] R. Servedio. On PAC learning using Winnow, Perceptron, and a Perceptron-like algorithm, *in* "Proc. Twelfth Ann. Conf. on Comp. Learning Theory" (1999), 296-307.

[36] J. Tarui and T. Tsukiji. Learning DNF by approximating inclusion-exclusion formulae, *in* "Proc. IEEE Conference on Computational Complexity" (1999), 215-220.

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

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

[39] K. Verbeurgt. Learning sub-classes of monotone DNF on the uniform distribution, *in* "Proc. 9th Conf. on Algorithmic Learning Theory" (1998), 385-399.