Exact Learning Boolean Functions via the Monotone Theory

Nader H. Bshouty[*]

Department of Computer Science

The University of Calgary

Calgary, Alberta, Canada T2N 1N4

e-mail: bshouty@cpsc.ucalgary.ca

Abstract

We study the learnability of boolean functions from membership and equivalence queries. We develop the Monotone Theory that proves

1) Any boolean function is learnable in time polynomial in its minimal DNF size, its minimal CNF size and the number of variables n.

In particular,

2) Decision trees are learnable.

Our algorithms are in the model of exact learning with membership queries and unrestricted equivalence queires. The hypotheses to the equivalence queries and the output hypotheses are depth 3 formulas.

1 Introduction

One of the main open problems in machine learning is whether boolean functions are learnable from membership and equivalence queries in time polynomial in the number of variables and their (disjunctive normal form) DNF sizes (the minimum number of terms in an equivalent formula in Disjunctive Normal Form). A more general line of research is whether all boolean functions are learnable in time polynomial using other representations, such as (conjunctive normal form) CNF, decision trees and multivariate polynomial. The latter problem for multivariate polynomials was recently solved by Schapire and Sellie [SS]. They show that any boolean function is learnable in time polynomial in n and its multivariate polynomial size.

Many results in the literature give evidence that learning boolean functions in different models in time polynomial in the DNF size and the number of variables is hard [AK91, AHP92, PR94]. On the other hand, many other results gave subclasses of boolean functions that are learnable in their DNF representations. Such subclasses include monotone DNF [A88] (DNF which contains

[*]This research was supported in part by the NSERC of Canada.

no negated variables), read-twice DNF [AP192,PR294] (DNF where each variable occurs at most twice), Horn sentences [AFP92] (DNF where each term contains at most one negated variable), k-DNF [V84] (DNF where each term contains at most k literals, for a constant k), $O(\log n)$-term DNF [BR92], where n is the number of variables (DNF with $O(\log n)$ terms) and read-k sat-j DNF [AP292] (DNF where each variable appears at most k times and every assignment to the variables satisfies at most j terms, j and k are constants).

In this paper we develop new techniques for exact learning of boolean functions. We show

1. Any boolean function is learnable in time polynomial in its minimal DNF size, its minimal CNF size and the number of variables n.
2. Decision trees are learnable.

This solves the open problem of learnability of decision trees.

Learning in this model implies learning in Valiant's PAC model if membership queries are available [A88], [V84]. It also implies that these classes are polynomial time predictable with a polynomial mistake bound if membership queries are available [L88].

Our algorithms are in the model of exact learning with membership queries and unrestricted equivalence queires. The hypotheses to the equivalence queries and the output hypotheses are depth 3 formulas.

2 The Learning Model

The learning criterion we consider is *exact identification.* There is a function f, called the *target function*, which is a member of a class of functions C defined over the variable set $\{x_1, \ldots, x_n\}$. The goal of the learning algorithm is to halt and output a formula h that is logically equivalent to f.

In a *membership query* the learning algorithm supplies an assignment a to the variables in $\{x_1, \cdots, x_n\}$ as input to a *membership oracle* and receives in return the value of $f(a)$.

In an (unrestricted) *equivalence query* the learning algorithm supplies any function h from a class of functions $H \supseteq C$ as input to an *equivalence oracle* and the reply of the oracle is either "yes", signifying that h is equivalent to f, or a *counterexample*, which is an assignment b such that $h(b) \neq f(b)$.

For more about this model see [A88].

3 Results

We prove the following results.

Theorem 1. *For assignments $A = \{a_1, \ldots, a_t\}$ let $\Lambda(A)$ be the set of all boolean functions that can be represented as CNF with clauses in which each clause is falsified by some assignment in A. Then any $f \in \Lambda(A)$ is learnable in time polynomial in the number of variables, the DNF size of f, and t.*

Theorem 2. *Any boolean function is learnable in time polynomial in the number of variables, its DNF size, and its CNF size.*

Theorem 3. *The conjunction of any boolean function f with a boolean function $g \in \Lambda(A)$ is learnable in time polynomial in the number of variables, the DNF size of $f \wedge g$, the size of A, and the CNF size of f.*

The above theorems imply

Result 1. *The class of k-almost monotone DNF is the class monotone DNF formulas that allow the existence of at most (constant) k non-monotone terms. This class includes the monotone DNF and the k-term DNF classes. Any k-almost monotone DNF is learnable in time polynomial in its DNF size and n. Also any conjunction of functions from this class is learnable in time polynomial in its DNF size and n.*

Proof. If we change k-almost monotone DNF to CNF by taking the conjunction of all possible disjunction of one literal from each term, we notice that every clause contains at most k negated variables.

Let $A = \{$ all vectors of Hamming weight $\leq k\}$. It is clear that any clause with at most k negated variables is falsified by some assignment in A. Now using Theorem 1 the result follows. A conjunction of any number of k-almost monotone DNF can also represented as CNF where each clause has at most k negated variables.$\Box$

This result is a generalization of the learnability of monotone DNF and k-term DNF [A88].

Result 2. *Any k-CNF, for $k = O(\log n)$, is learnable in time polynomial in its DNF size and n. In particular, a conjunction of any number of $O(\log n)$-term DNF is learnable in time polynomial in its DNF size and n. This implies that a conjunction of $O(\log n/\log\log n)$ number of $O(\log n)$-term DNF is learnable in time polynomial in n.*

Proof. An $(n,k)-$*universal set* is a set $\{b_1,\ldots,b_t\} \subseteq \{0,1\}^n$ such that every subset of k variables assumes all of its 2^k possible assignments in the b_i's. In [ABNR92,CZ93] an $(n,k)-$universal set is constructed of size $t = O(k2^{3k}\log n)$. This size is polynomial when $k = O(\log n)$.

We show that the above classes are subset of $\Lambda(A)$ where A is any (n,k)-universal set. Let f be any k-CNF. Let $C = x_{i_1}^{c_1} \vee \cdots \vee x_{i_l}^{c_l}$, $l \leq k$ be a clause in f where $x^c = x$ when $c = 0$ and $\bar{x}$ when $c = 1$. Since A is an (n,k)-universal set there is $a \in A$ such that $a_{i_j} = c_j$ for $j = 1,\ldots,l$ and this assignment falsifies C.

A conjunction of $O(\log n/\log\log n)$ number of $O(\log n)$-term DNF has DNF size at most $O(\log n)^{O(\log n/\log\log n)} = poly(n)$.$\Box$

This is a generalization of Blum and Rudich's result for learning $O(\log n)$-term DNF [BR92].

Result 3. *A (constant) depth $2t+1$ $(k_1,\ldots,k_{2t})$-formula is a formula that is a conjunction of at most k_{2t} formulas, each of which is a disjunction of at most k_{2t-1} formulas, each of which is $(k_1,\ldots,k_{2t-2})$-formula. A $()$-formula is a term. The class of $(k_1,\ldots,k_{2t})$-formulas, where $k_1 = k_3 = \cdots = k_{2t-1} = O((\log n)^{1/t})$ and $k_2 = k_4 = \cdots = k_{2t} = O((\log n/\log\log n)^{1/t})$ is* *learnable in time polynomial in n. Also any circuit of depth t and fanin $O((\log n)^{1/t})$ for the OR gates (no restriction on the AND-fanin) is learnable in time polynomial in its DNF size and n.*

Proof. It is not hard to demonstrate that the above classes are subclasses of $O(\log(n))$-CNF.□

The only nontrivial large depth formulas (depth more than 2) that are proved to be learnable in the literature are read-once formulas over different basis [BHH92a, BHH92b, BHHK91]. Linial, Mansour and Nisan [LMN93] showed that constant depth circuits are PAC learnable with membership queries in time $n^{poly(\log n)}$ under the uniform distribution. The above formula is the first nontrivial class of constant depth formulas that are learnable and have no read restrictions.

Result 4. *Any boolean function is learnable in time polynomial in its decision tree size and n. This implies that any decision tree of polynomial size is learnable in time polynomial in n.*

Proof. The size of any decision tree of a boolean function f is polynomial in its DNF and CNF size. By Theorem 2 the result follows.□

Several polynomial time learning algorithms have been given for learning certain restricted subclasses of decision trees. Ehrenfeucht and Haussler [EH89] gave an algorithm for PAC- learning (without membership queries) decision trees of constant rank in time polynomial and gave a PAC-learning algorithm for general polynomial size decision trees in time $n^{O(\log n)}$, (see also [Bl92,R87]). Kushilevitz and Mansour [KM91] gave a polynomial time PAC learning algorithm with membership queries for decision trees under the uniform distribution. Hancock [H93] gave a polynomial time algorithm for PAC learning read-k decision trees. Our result implies PAC learning of decision trees with membership queries under any distribution.

Result 5. *A CDNF boolean function is a boolean function whose CNF size is polynomial in its DNF size. Any CDNF boolean formula is learnable in time polynomial in its DNF size and n. In particular, a polynomial size DNF that has a polynomial size CNF is learnable in time polynomial in n. In the former case the learner need not know any information about the DNF and the CNF size of the target formula.*

Proof. Follows immediately from Theorem 2.□

Ehrenfeucht and Haussler [EH89] showed that polynomial size DNF that have polynomial size CNF are PAC learnable under any distribution in time $n^{O(\log^2 n)}$.

Result 6. *Decision trees over terms are decision trees where each node contains a term. Decision trees of constant depth over terms are learnable in time polynomial in n.*

Proof. We show that depth k decision trees over terms have DNF and CNF size at most $(2n)^k$. Then using theorem 2 the result follows. Any depth k decision tree (over variables) is a k-DNF and k-CNF. Therefore if f is a decision tree over terms of depth k then it can written as $f=\bigvee_{i=1}^{r} D_i$ where

$$D_i = T_{i,1}\wedge\cdots\wedge T_{i,j_i}\wedge\overline{T_{i,j_i+1}}\wedge\cdots\overline{T_{i,m_i}}$$

$m_i \le k$ and $T_{i,l}$ are terms. The DNF size of D_i is at most $n^{m_i}\le n^k$. Since the depth of the tree is k we have that the number of leaves in the tree is at most 2^k and therefore the DNF size of f is at most $(2n)^k$. Duality implies the CNF size of f is at most $(2n)^k$.□

This is a generalization of the learnability of k-term DNF, k-clause CNF and k-term decision lists, [A88,R87]. It also implies that boolean multivariate polynomials with (not necessarily monotone) k terms is learnable.

Result 7 . *Any conjunction of the formulas in results 1-6 is learnable in time polynomial in its DNF size and n. For example, a conjunction of CDNF boolean functions with $O(\log n)$-CNF is learnable in time polynomial in its DNF size and n.*

Proof. Follows from theorem 3 and the fact that the conjunction of a boolean function in $\Lambda(A)$ with a boolean function in $\Lambda(B)$ is in $\Lambda(A \cup B)$.$\square$

All of the above results are for boolean functions. In a companion paper [B93] we study the learnability of concept classes over nonboolean domains. We define DNFs, CNFs and decision trees over any alphabet and show conditions for which the above results are true for any concept. In particular, we show that

Result 8. *Any polynomial size decision tree over any alphabet is learnable and any function $f : \Sigma^n \rightarrow \Sigma$ over any alphabet Σ is learnable in time polynomial in its minimal decision tree size and the number of variables.*

This paper is organized as follows. In section 4 we give the monotone theory of boolean functions and prove some preliminary results. In section 5 we present the first algorithm and prove Theorem 1. In section 6 we present the second algorithm and prove Theorem 2. In section 7 we show how to combine the two algorithms into one general algorithm and prove Theorem 3.

4 The Monotone Theory

In this section we give the notation, definitions, and preliminary results, that we need to write the algorithms and prove their correctness.

For a vector x, x_i or $x[i]$ is the i-th entry of x. For two vectors we write $y \le x$ if '$y_i = 1$ implies $x_i = 1$' for all i. For an assignment $a \in \{0,1\}^n$ we define $x \le_a y$ if and only if $x + a \le y + a$. Here $x + a$ is the group addition in $GF(2)^n$ (bitwise XOR).

A boolean function f is called a-monotone if $f(x+a)$ is monotone. The *(minimal) a-monotone boolean function* $\mathcal{M}_a(f)$ of f is defined as follows:

$$\mathcal{M}_a(f)(x) = \begin{cases} 1 & (\exists y \le_a x)\ f(y) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

We write $\mathcal{M}$ for $\mathcal{M}_0$. The following are properties of $\mathcal{M}_a$.

Lemma 1 . We have

1. $\mathcal{M}_a(f) = \mathcal{M}(f(x+a))(x+a)$.
2. $\mathcal{M}_a(f \vee g) = \mathcal{M}_a(f) \vee \mathcal{M}_a(g)$.
3. $\mathcal{M}_a(f \wedge g) \Rightarrow \mathcal{M}_a(f) \wedge \mathcal{M}_a(g)$.
4. f is a-monotone iff $\mathcal{M}_a(f) = f$.

5. $f \Rightarrow \mathcal{M}_a(f)$.

6. If $f(a)=1$ then $\mathcal{M}_a(f)\equiv 1$.

7. For a DNF $f=\bigvee_{i=1}^{k}(x_{e_{i1}}^{c_{i1}}\wedge\cdots\wedge x_{e_{i\delta_i}}^{c_{i\delta_i}})$ (here $x^0=x$ and $x^1=\bar{x}$) we have
$$\mathcal{M}_a(f)=\bigvee_{i=1}^{k}\ \bigwedge_{j:c_{i,j}=a_{e_{i,j}}} x_{e_{ij}}^{c_{i,j}}.$$
If all $c_{i,j}\neq a_{e_{i,j}}$ for some term, then $\mathcal{M}_a(f)=1$.

Proof. The proof is given in the Appendix.$\Box$

Another interesting property of the monotone boolean function of f is

Lemma 2 . We have
$$f\equiv\bigwedge_{a\in\{0,1\}^n}\mathcal{M}_a(f).$$

Proof. If $\bigwedge_{a\in\{0,1\}^n}\mathcal{M}_a(f)(x_0)=1$ then for every a there exists $y\le_a x_0$ such that $f(y)=1$. If we choose $a=x_0$ then since $y+a\le 0$ we must have $y=a=x_0$ and then $f(x_0)=1$. This implies that $\bigwedge_{a\in\{0,1\}^n}\mathcal{M}_a(f)\Rightarrow f$. Since by (5) in lemma 1, $f\Rightarrow\mathcal{M}_a(f)$ we have $f\Rightarrow\bigwedge_{a\in\{0,1\}^n}\mathcal{M}_a(f)$.$\Box$

Using lemma 2 we define the *Monotone dimension of a class* of boolean functions C.

Definition 1 . Let C be a class of boolean functions. We define the monotone dimension $d=Mdim(C)$ of C to be the minimal number of assignments $a_1,\ldots,a_d$ such that for any $f\in C$ we have
$$f\equiv\bigwedge_{i=1}^{d}\mathcal{M}_{a_i}(f).$$

A set of assignments $\{a_1,\ldots,a_d\}$ that satisfies the above equivalence for all $f\in C$ is called an *M-basis* of C. (d need not be monimal).

The following lemma shows a simple way to find the M-basis and Mdim of a class.

Lemma 3 . Let C be a class of boolean functions. Then $Mdim(C)$ is the minimal number of assignments $a_1,\ldots,a_d$ such that for every $f\in C$, there exist d monotone boolean functions $M_1,\ldots,M_d$ such that
$$f\equiv M_1(x+a_1)\wedge\cdots\wedge M_d(x+a_d).$$
A set $\{a_1,\ldots,a_d\}$ that satisfies the above equivalence for any $f\in C$ (with possibly different M_i) is an M-basis for C.

Proof. Let $a_1,\ldots,a_d$ be assignments that satisfy the conditions in the lemma. Let
$$g=\bigwedge_{i=1}^{d}\mathcal{M}_{a_i}(f).$$
If we show that for every $f\in C$, $g\equiv f$, then $Mdim(C)=d$ and $\{a_1,\ldots,a_d\}$ is an M-basis for C.

Since by (5) in lemma 1 $f(x)\Rightarrow\mathcal{M}_a(f)$ for any a we have $f(x)\Rightarrow g(x)$. Now by (3) and (4) in lemma 1,
$$\mathcal{M}_{a_i}(f)\Rightarrow\mathcal{M}_{a_i}(M_i(x+a_i))\Rightarrow M_i(x+a_i),$$

and therefore $g(x) \Rightarrow f(x)$.$\square$

Lemma 4. A set of assignments $A = \{a_1, \ldots, a_d\}$ is an M-basis for C if and only if every boolean function in C can be represented as CNF such that every clause in this CNF is falsified by some assignment in A.

In particular, if $f = c_1 \wedge \cdots \wedge c_s$ is any CNF of f and each clause c_i is falsified by some $a \in A$ then A is an M-basis for $\{f\}$ and

$$f = \bigwedge_{a \in A} \mathcal{M}_a(f).$$

Proof. If $A = \{a_1, \ldots, a_d\}$ is an M-basis for C then by lemma 3 every function $f \in C$ can be written as $f = M_1(x + a_1) \wedge \cdots \wedge M_d(x + a_d)$ for some monotone M_i. For $1 \le i \le d$ let c be a monotone clause in the monotone CNF representation of M_i. Since $c(x + a_i)$ is a clause in the CNF representation of $M_i(x + a_i)$ and $c(a_i + a_i) = c(0) = 0$, this clause is falsified by $a_i \in A$.

Now suppose every boolean function in $f \in C$ can be represented as CNF, $f = c_1 \wedge \cdots \wedge c_s$ such that every clause c_i in this CNF is falsified by some assignment $a_{j(i)}$ in A. Then it is clear that $M'_i = c_i(x + a_{j(i)})$ is monotone and f can be written as

$$f = M_1(x + a_1) \wedge \cdots \wedge M_d(x + a_d)$$

where

$$M_r(x) = \bigwedge_{\{i | j(i) = r\}} M'_i(x)$$

is monotone. By lemma 3, A is an M-basis for C.$\square$

For a function f, the DNF size (CNF size) of f, $size_{DNF}(f)$, ($size_{CNF}(f)$), is the minimal number of terms (clauses) over all possible DNF (CNF) formulas of f. For a decision tree T the *decision tree size* of T, $size_{DT}(T)$, is the number of leaves in the tree. The decision tree size of a boolean function f is

$$size_{DT}(f) = \min_{T \equiv f} size_{DT}(T).$$

Lemma 5 . We have

1. $size_{DNF}(\mathcal{M}(f)) \le size_{DNF}(f)$.
2. $size_{DT}(f) \ge size_{DNF}(f) + size_{CNF}(f)$.
3. $Mdim(\{f\}) \le size_{CNF}(f)$.

Proof. Part 1 follows from (7) in lemma 1. Part 2 follows from the fact that in a decision tree each leaf that is labeled with 1 corresponds to one term in the corresponding DNF representation of the tree and each leaf that is labeled with 0 corresponds to one clause in the corresponding CNF representation of the tree. Part 3 follows from lemma 4.$\square$

Let a be an assignment. We define $M_{DNF}(a) = \bigwedge_{a_i = 1} x_i$ for $a \ne 0_n$ and $M_{DNF}(0_n) = 1$ (0_n is the zero assignment). For a set of assignments S we define $M_{DNF}(S) = \vee_{a \in S} M_{DNF}(a)$ if S is not empty and $M_{DNF}(\emptyset) = 0$.

For an assignment a and a variable x_j we define the assignment $b = a_{\neg x_j}$ where $b_i = a_i$ for $i \ne j$ and $b_j = 1 + a_j$.

```
The Λ-Algorithm
{a_1, ..., a_t} is an M-basis for C.

1) $S_i \leftarrow \emptyset, H_i \leftarrow 0$ for $i = 1, \ldots, t$.
2) EQ($\bigwedge_{i=1}^{t} H_i$) $\rightarrow v$; If the answer is "YES" then stop.
3) $I \leftarrow \{i | H_i(v) = 0\}$.
4) For each $i \in I$ do
        $v_i \leftarrow v$;
        Walk from $v_i$ toward $a_i$ while keeping $f(v_i) = 1$
        $S_i \leftarrow S_i \cup \{v_i + a_i\}$.
5) $H_i \leftarrow M_{DNF}(S_i)(x + a_i)$ for $i = 1, \ldots, t$.
6) Goto 2.
```

Figure 1: *An algorithm that learns f from $size_{DNF}(f) Mdim(C) n^2$ membership queries and $size_{DNF}(f) Mdim(C)$ equivalence queries.*

5 Learning A Class With Known M-basis

In this section we prove that any boolean function f in a class C is learnable from $size_{DNF}(f)$ $Mdim(C)$ n^2 membership queries and $size_{DNF}(f) Mdim(C)$ equivalence queries.

5.1 The Λ-Algorithm

The algorithm is given in Figure 1. In the algorithm the command EQ($\wedge_{i=1}^{t} H_i$)$\rightarrow v$ asks equivalence query on the formula $\wedge_{i=1}^{t} H_i$. If the answer is "YES" then the algorithm stops. Otherwise the equivalence oracle returns the counterexample v. The routine *Walk from v toward a while keeping* $f(v) = 1$ takes two assignments v and a. It walks v towards a (flip an i-th bit in v that disagrees with the i-th bit in a) while preserving the property that $f(v) = 1$. The routine stops flipping bits when flipping any bit that disagree with a_j changes the value of the function to 0. The output of the procedure is a vector v' where $f(v') = 1$ and for every $v'_j \neq a_j$ we have $f(v'_{\neg x_j}) = 0$. This procedure runs with at most n^2 membership queries.

5.1.1 Proof of correctness

Before we prove the correctness of the algorithm we prove the following.

Proposition A. Let f be a boolean function. If y is an assignment such that $f(y) = 1$ and for any i where $y_i = 1$ we have $f(y_{\neg x_i}) = 0$, then for any DNF $\vee_{i=1}^{s} T_i$ of f there exists a term T_{i_0} such that

$$\mathcal{M}(T_{i_0}) = M_{DNF}(y).$$

Proof. If $y = 0_n$ then there exists a term $T = \bar{x}_{i_1} \cdots \bar{x}_{i_k}$ in f and $\mathcal{M}(T) = 1 = M_{DNF}(0_n)$. Suppose $y = (y_1, \ldots, y_n) \neq 0_n$. Since $f(y) = 1$ we have $\vee_{i=1}^{s} T_i(y) = 1$ and therefore there exists $T_{i_0}(y) = 1$. Let $T_{i_0} = x_1 \wedge x_2 \cdots \wedge x_{m_1} \wedge \bar{x}_{m_1+1} \wedge \cdots \wedge \bar{x}_{m_2}$ (without loss of generality). Since $T_{i_0}(y) = 1$ we have $y_1 = y_2 = \cdots = y_{m_1} = 1$ and $y_{m_1+1} = \cdots = y_{m_2} = 0$. If $y_i = 1$ for some $i > m_2$ then $T_{i_0}(y_{\neg x_i}) = 1$ and therefore $f(y_{\neg x_i}) = 1$ which is a contradiction. Therefore, $y_i = 0$ for $i > m_2$ and we have

$$\mathcal{M}(T_{i_0}) = x_1 \wedge \cdots \wedge x_{m_1} = M_{DNF}(y). \square$$

Now, let $f \in C$ and $f = \bigvee_{i=1}^{s} (x_{e_{i1}}^{c_{i1}} \wedge \cdots \wedge x_{e_{i\delta_i}}^{c_{i\delta_i}})$ where $s = size_{DNF}(f)$. Recall that $x^0 = x$ and $x^1 = \bar{x}$. Since $\{a_1, \ldots, a_t\}$ is an M-basis for C we have

$$f = \bigwedge_{i=1}^{t} \mathcal{M}_{a_i}(f). \tag{1}$$

Let $a_i = (a_{i,1}, \ldots, a_{i,n})$. By (7) in lemma 1,

$$\mathcal{M}(f(x + a_k)) = \bigvee_{i=1}^{s} \bigwedge_{j: a_{k,e_{ij}} = c_{ij}} x_{e_{ij}} \tag{2}$$

has at most s terms (If all $a_{k,e_{i,j}} \neq c_{i,j}$, then the i-th term is 1). Let

$$\mathcal{T}_k = \{ \bigwedge_{j: a_{k,e_{ij}} = c_{ij}} x_{e_{ij}} | i = 1, \ldots, s \}.$$

The set $\mathcal{T}_k$ contains the terms of $\mathcal{M}(f(x + a_k))$. Notice that

$$\left(\bigwedge_{T \in \mathcal{T}_i} T \right) (x + a_i) = \mathcal{M}_{a_i}(f).$$

Let $H_{i,\delta}, S_{i,\delta}, (i = 1, \ldots, t), v_\delta, I_\delta$ and $v_{i,\delta}$ for $i \in I_\delta$ be the variables $H_i, S_i, (i = 1, \ldots, t), v, I$ and v_i for $i \in I$, respectively, of the algorithm in the δ-th iteration. We prove by induction that

1. v_δ is a positive counterexample.
2. I_δ is not empty.
3. $M_{DNF}(v_{i,\delta} + a_i) \in \mathcal{T}_i \backslash \{M_{DNF}(s) | s \in S_{i,\delta-1}\}$ for $i \in I_\delta$.
4. $\{M_{DNF}(s) | s \in S_{i,\delta}\} \subseteq \mathcal{T}_i$ for $i = 1, \ldots, t$.
5. $H_{i,\delta} \Rightarrow \mathcal{M}_{a_i}(f)$ for $i = 1, \ldots, t$.

We prove 1-5 for the $\delta + 1$ iteration. The same proof also applies for the first iteration. Since by induction hypothesis 5, $H_{i,\delta} \Rightarrow \mathcal{M}_{a_i}(f)$, we have

$$\bigwedge_{i=1}^{t} H_{i,\delta} \Rightarrow \bigwedge_{i=1}^{t} \mathcal{M}_{a_i}(f) = f. \tag{3}$$

(Also true for $\delta = 1$). Therefore $v_{\delta+1}$ is a positive counterexample. This proves 1 for $\delta + 1$.

Since $\bigwedge_{i=1}^{t} H_{i,\delta}(v_{\delta+1}) = 0$ the set $I_{\delta+1}$ is not empty. This proves 2 for $\delta + 1$. Now we have for all $i \in I_{\delta+1}$

$$H_{i,\delta}(v_{\delta+1}) = 0 \tag{4}$$

and since $v_{\delta+1}$ is a positive counterexample,

$$f(v_{\delta+1}) = 1. \tag{5}$$

From (4) and (5) and by the definition of $H_{i,\delta}$ we have for all $i \in I_{\delta+1}$

$$M_{DNF}(S_{i,\delta})(v_{\delta+1} + a_i) = 0,$$

and

$$f((v_{\delta+1} + a_i) + a_i) = 1. \tag{6}$$

By step 4 in the algorithm we have for every $i \in I_{\delta+1}$

$$v_{i,\delta+1} + a_i \le v_{\delta+1} + a_i, \tag{7}$$

$f(v_{i,\delta+1}) = f((v_{i,\delta+1} + a_i) + a_i) = 1$ and for every j where $(v_{i,\delta+1} + a_i)[j] = 1$,

$$f((v_{i,\delta+1})_{\neg x_j}) = f((v_{i,\delta+1} + a_i)_{\neg x_j} + a_i) = 0. \tag{8}$$

Since $M_{DNF}(S_{i,\delta})(x)$ is monotone, (6) and (7) imply that

$$M_{DNF}(S_{i,\delta})(v_{i,\delta+1} + a_i) = 0. \tag{9}$$

By proposition A, (8) implies that there exists a term T in the DNF of $f(x + a_i)$ such that $\mathcal{M}(T) = M_{DNF}(v_{i,\delta+1} + a_i)$. Therefore,

$$M_{DNF}(v_{i,\delta+1} + a_i) \in \mathcal{T}_i. \tag{10}$$

From (9) and (10) and the induction hypotheses 4, we get 3 for $\delta + 1$. Now since $S_{i,\delta+1} = S_{i,\delta} \cup \{v_{i,\delta+1} + a_i\}$ for $i \in I_{\delta+1}$ and $S_{i,\delta+1} = S_{i,\delta}$ for $i \notin I_{\delta+1}$, the by induction hypothesis 4 and (10) we get 4 for $\delta + 1$.

By 4 for $\delta + 1$ we have

$$\begin{aligned} H_{i,\delta+1}(x + a_i) &= \bigvee_{s \in S_{i,\delta+1}} M_{DNF}(s) \\ &\Rightarrow \bigvee_{T \in \mathcal{T}_i} T = \mathcal{M}(f(x + a_i)). \end{aligned}$$

This proves 5 for $\delta + 1$. This completes the proof of 1-5.

Now, 1-5 shows that at each iteration for some $i = 1, \ldots, t$ we add one new term of $\mathcal{T}_i$ to the formula H_i. Since $|\mathcal{T}_i| \le s$ after at most $\lambda \le st \le size_{DNF}(f)Mdim(C)$ iterations of the algorithm we will have

$$H_{i,\lambda} = \left(\bigwedge_{T \in \mathcal{T}_i} T \right) (x + a_i) = \mathcal{M}_{a_i}(f),$$

and $\bigwedge_{i=1}^{t} H_{i,\lambda} = f$. This completes the proof of the correctness of the algorithm.$\Box$

The CDNF-Algorithm

```
1) $t \leftarrow 0$
2) EQ(1) $\rightarrow v$; If the answer is "YES" then stop.
3) $t \leftarrow t+1, H_t \leftarrow 0, S_t \leftarrow \emptyset, a_t \leftarrow v$
4) EQ($\bigwedge_{i=1}^{t} H_i$) $\rightarrow v$; If the answer is "YES" then stop.
5) $I \leftarrow \{i | H_i(v) = 0\}$.
6) If $I$ is empty then Goto 3.
7) For each $i \in I$ do
        $v_i \leftarrow v$;
        Walk from $v_i$ toward $a_i$ while keeping $f(v_i) = 1$
        $S_i \leftarrow S_i \cup \{v_i + a_i\}$.
8) $H_i \leftarrow M_{DNF}(S_i)(x + a_i)$ for $i = 1, \ldots, t$.
9) Goto 4.
```

Figure 2: *An algorithm that learns f from $size_{DNF}(f)size_{CNF}(f)n^2$ membership queries and $size_{DNF}(f)size_{CNF}(f)$ equivalence queries.*

5.1.2 The Complexity of the Λ-Algorithm

It is clear that the algorithm runs with $n^2 size_{DNF}(f) Mdim(C)$ membership queries and $size_{DNF}(f)$ $Mdim(C)$ equivalence queries. The factor n^2 in the number of the membership queries is due to the fact that the procedure *Walk* might ask n^2 membership queries each iteration.

6 Learning Boolean Functions

In this section we give an algorithm that learns any boolean function in time polynomial in the number of variables, the DNF size of f and the CNF size of f with $size_{DNF}(f)size_{CNF}(f)n^2$ membership queries and $size_{DNF}(f)size_{CNF}(f)$ equivalence queries.

6.1 The CDNF-Algorithm

To prove the correctness of the algorithm, we need the following proposition.

Proposition B. Let f be a boolean function and $f = \bigwedge_{i=1}^{s} c_i$ be a minimal size CNF for f. Let $r < s$ and $\{a_1, \cdots, a_t\}$ be a set of assignments such that for every c_i, $i \leq r < s$, there exists $a_{j(i)}$ such that $c_i(a_{j(i)}) = 0$. Let H_i, $i = 1, \ldots, t$ be a boolean function that satisfies $H_i \Rightarrow \mathcal{M}_{a_i}(f)$.

If for an assignment a we have

$$\left(\bigwedge_{i=1}^{t} H_i\right)(a) = 1 \quad \text{and} \quad f(a) = 0,$$

then there exists a clause $c_i, i > r$, that satisfies $c_i(a) = 0$.

Proof. Since $(\bigwedge_{i=1}^{t} H_i)(a) = 1$ we also have $\wedge_{i=1}^{t} \mathcal{M}(f(x + a_i))(a + a_i) = 1$. If $c_{i_0}(a) = 0$ for some $i_0 \le r$, then by (3) and (4) in lemma 1

$$\begin{array}{rcl} 1 & = & \mathcal{M}(f(x + a_{j(i_0)}))(a + a_{j(i_0)}) \\ & \Rightarrow & \left(\bigwedge_{i=1}^{s} \mathcal{M}(c_i(x + a_{j(i_0)}))\right)(a + a_{j(i_0)}) \\ & \Rightarrow & \mathcal{M}(c_{i_0}(x + a_{j(i_0)}))(a + a_{j(i_0)}) \\ & \Rightarrow & c_{i_0}(a) = 0 \end{array}$$

A contradiction. Therefore $c_i(a) = 1$ for all $i \le r$. Since $f(a) = 0$ we must have $c_i(a) = 0$ for some $i > r$.□

Now to prove the correctness of the CDNF-algorithm notice that steps 4-9 are identical, except for step 6, to the steps of the Λ-algorithm. We have added lines 1-3 and 6 to find an M-basis for $\{f\}$. Let $H = \bigwedge c_i$ be a minimal size CNF for f. By proposition B each negative counterexample a will falsify a new clause c_i of H. After at most $size_{CNF}(f)$ negative counterexamples we will have: for every clause c_i in $H \equiv f$ there exists $a_{j(i)}$ that satisfies $c_i(a_{j(i)}) = 0$. Now by lemma 4, $\{a_i\}$ is an M-basis of $\{f\}$ and therefore the Λ-algorithm learns f.□

7 Learning Conjunctions of Classes

In this section we combine the two algorithms into one algorithm that learns a conjunction of any two classes such that the first is learnable using the Λ-algorithm and the second is learnable using the CDNF-algorithm.

The algorithm is given in Figure 3.

Notice that we can build an M-basis for $f_1 \wedge f_2$ from the union of each M-basis for f_1 and f_2. Since A is an M-basis for C_1 and $f_1 \in C_1$, A is also an M-basis for f_1. Therefore we start with the M-basis A and build a basis for f_2 from negative counterexamples.

8 Appendix

In this appendix we prove lemma 1.

1.

$$\begin{array}{rcl} \mathcal{M}_a(f)(x_0) = 1 & \Longleftrightarrow & (\exists y \le_a x_0) f(y) = 1 \\ & \Longleftrightarrow & (\exists y + a \le x_0 + a) f((y + a) + a) = 1 \\ & \Longleftrightarrow & (\exists z \le x_0 + a) f(z + a) = 1 \\ & \Longleftrightarrow & \mathcal{M}(f(x + a))(x_0 + a) = 1 \end{array}$$

```
**Λ+CDNF-Algorithm**
$A = \{a_1, \ldots, a_r\}$ is an M-basis for $C_1$.

1) $t \leftarrow r, H_i \leftarrow 0, S_i \leftarrow \emptyset, i = 1, \ldots, t$
2) EQ($\bigwedge_{i=1}^{t} H_i$) $\rightarrow v$; If the answer is "YES" then stop.
3) $I \leftarrow \{i | H_i(v) = 0\}$.
4) If $I$ is empty then
       $t \leftarrow t + 1, H_t \leftarrow 0, S_t \leftarrow \emptyset$.
       $a_t \leftarrow v$.
       Goto 2.
5) For each $i \in I$ do
       $v_i \leftarrow v$;
       Walk from $v_i$ toward $a_i$ while keeping $f(v_i) = 1$
       $S_i \leftarrow S_i \cup \{v_i + a_i\}$.
6) $H_i \leftarrow M_{DNF}(S_i)(x + a_i)$ for $i = 1, \ldots, t$.
7) Goto 2.
```

Figure 3: *An algorithm that learns* $f = f_1 \wedge f_2 \in C_1 \wedge C_2$ *from* $size_{DNF}(f)(Mdim(C_1) + size_{CNF}(f_2))n^2$ *membership queries and* $size_{DNF}(f)(Mdim(C_1) + size_{CNF}(f_2))$ *equivalence queries.*

2.

$$\begin{aligned}
\mathcal{M}_a(f\vee g)(x_0)=1 &\iff (\exists y \le_a x_0)(f(y)=1 \vee g(y)=1)\\
&\iff (\exists y \le_a x_0)f(y)=1 \vee (\exists y \le_a x_0)g(y)=1\\
&\iff \mathcal{M}_a(f(x))(x_0)=1 \vee \mathcal{M}_a(g(x))(x_0)=1\\
&\iff \mathcal{M}_a(f(x))(x_0) \vee \mathcal{M}_a(g(x))(x_0)=1
\end{aligned}$$

3.

$$\begin{aligned}
\mathcal{M}_a(f\wedge g)(x_0)=1 &\iff (\exists y \le_a x_0)(f(y)=1 \wedge g(y)=1)\\
&\Rightarrow (\exists y \le_a x_0)f(y)=1 \wedge (\exists y \le_a x_0)g(y)=1\\
&\iff \mathcal{M}_a(f(x))(x_0)=1 \wedge \mathcal{M}_a(g(x))(x_0)=1\\
&\iff \mathcal{M}_a(f(x))(x_0) \wedge \mathcal{M}_a(g(x))(x_0)=1
\end{aligned}$$

4. We have f is a-monotone if and only if $f(x+a)$ is monotone. Let $g(x)=f(x+a)$ and $z=y+a$. Since g is monotone

$$\begin{aligned}
\mathcal{M}_a(f)(x_0)=1 &\iff (\exists y \le_a x_0)f(y)=1\\
&\iff (\exists z \le x_0+a)f(z+a)=1\\
&\iff (\exists z \le x_0+a)g(z)=1\\
&\iff g(x_0+a)=1\\
&\iff f(x_0)=1
\end{aligned}$$

5. Since $x_0 \le_a x_0$,

$$f(x_0)=1 \Rightarrow (\exists y \le_a x_0)f(y)=1 \iff \mathcal{M}_a(f)(x_0)=1.$$

6. Since $a \le_a x_0$ for any x_0

$$f(a)=1 \Rightarrow (\forall x_0)(\exists y \le_a x_0)f(y)=1 \Rightarrow (\forall x_0)\mathcal{M}_a(f(x))(x_0)=1 \iff \mathcal{M}_a(f(x)) \equiv 1.$$

7. First notice that the minimal (in the order $<$) assignment that satisfies $t=x_1\wedge\cdots\wedge x_r\wedge\bar{x}_{r+1}\wedge\cdots\wedge\bar{x}_s$ is $(1_r 0_{n-r})$, where $\xi_r \in \{0,1\}$ is the assignment in $\{0,1\}^r$ such that all of its entries are ξ. Also any assignment that satisfies t is greater than $(1_r 0_{n-r})$. Therefore $\mathcal{M}(x_1\wedge\cdots\wedge x_r\wedge\bar{x}_{r+1}\wedge\cdots\wedge\bar{x}_s)=x_1\wedge\cdots\wedge x_r$.
Now 7 follows immediately from 1 and 2.$\square$

Acknowlegement . I would like to thank Dana Angluin, Sally Goldman, Tom Hancock, Lisa Hellerstein, Martin Krikis, Sleiman Matar, Dan Roth and Tino Tamon for many useful comments.

References

[A88] Dana Angluin. Queries and concept learning. *Machine Learning*, 2:319-342, 1988.

[ABNR92] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs, *IEEE Transactions on information theory*, 38, 2:509-516, 1992.

[AFP92] Dana Angluin, Michael Frazier, and Leonard Pitt. Learning conjunctions of horn clauses. *Machine Learning*, 9:147-164, 1992.

[AHP92] Howard Aizenstein, Lisa Hellerstein, and Leonard Pitt. Read thrice DNF is hard to learn with membership and equivalence queries. In *Proceedings of the 33rd Symposium on Foundations of Computer Science*, pages 523–532, 1992.

[AK91] Dana Angluin and Michael Kharitonov. When won't membership queries help? In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 444–454, 1991.

[AP192] Howard Aizenstein and Leonard Pitt. Exact learning of read-twice DNF formulas. In *Proceedings of the 32nd Symposium on Foundations of Computer Science*, 1991.

[AP292] Howard Aizenstein and Leonard Pitt. Exact learning of read-k disjoint DNF and not-so-disjoint DNF. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 71–76, August 1992.

[Bl92] Avrim Blum. Rank-r decision trees are a subclass of r-decision lists *Information Processing Letters*, **43**, pages 183–185, 1992.

[B93] Nader H. Bshouty, Exact learning concepts via the monotone theory. In preparation.

[BHH92a] Nader H. Bshouty, Thomas R. Hancock, and Lisa Hellerstein. Learning arithmetic read-once formulas. In *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing*, May 1992.

[BHH92b] Nader H. Bshouty, Thomas R. Hancock, and Lisa Hellerstein. Learning Boolean read-once formulas with arbitrary symmetric and constant fan-in gates. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 1–15, August 1992.

[BHHK91] Nader H. Bshouty, Thomas R. Hancock, Lisa Hellerstein, and Marek Karpinski. Read-once threshold formulas, justifying assignments, and transformations. Technical report, International Computer Science Institute TR-92-020, 1991.

[BR92] Avrim Blum and Steven Rudich. Fast learning of k-term DNF formulas with queries. In *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing*, pages 382–389, May 1992.

[CZ93] Gérard D. Cohen and Gilles Zémor, Intersecting codes and independent families. Draft, 1993.

[EH89] Andrzej Ehrenfeucht, and David Haussler. Learning decision tree from random examples. *Information and Computation*, 82, pages 231–246, 1989.

[H93] Thomas R. Hancock. Learning $k\mu$ decision trees on the uniform distribution. In *Proceedings of the Sixth Annual Workshop on Computational Learning Theory*, pages 352-360 July, 1993.

[KM91] Eyal Kushilevitz, and Yishay Mansour. Learning decision trees using Fourier spectrum. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, 1991.

[LMN93] Nathan Linial, Yishay Mansour and Noam Nisan. Constant depth circuits, Fourier Transform, and Learnability. In *Proceedings of the Thirtieth Annual Symposium Foundation of Computer Science*, 1989.

[L88] Nick Littlestone. Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[PR94] Krishnan Pillaipakkamnatt and Vijay Raghavan, On the limits of proper learnability of subclasses of DNF formulas. *Proceeding of the Seventh ACM Workshop on Computational Learning Theory,* 1994.

[PR94] Krishnan Pillaipakkamnatt and Vijay Raghavan, Read twice DNF formulas are properly learnable. TR-CS-93-59, Vanderbilt University.

[R87] Ronald L. Rivest, Learning decision lists. *Machine Learning*, **2**, pages 229–246, 1987.

[SS93] Robert E. Schapire and Linda M. Sellie, Learning sparse multivariate polynomials over a field with queries and counterexamples, *Proceeding of the Sixth ACM Workshop on Computational Learning Theory,* 1993.

[V84] Leslie Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.