

The Monotone Theory

Nader H. Bshouty

Eyal Kushilevitz

Abstract

Here we give the Monotone Theory and show that any boolean function is learnable from membership and equivalence queries in polynomial time in n , its CNF size and its DNF size. In particular, the class of decision trees is learnable in polynomial time.

1 The Monotone Theory

A simple (yet sometimes very powerful) strategy in algorithms design is to try first to develop an algorithm for some special case (hopefully simpler) of the problem. Then, one can try to run this algorithm on instances for the general problem (that do not satisfy the restrictions of the special case). Obviously, we expect the algorithm to fail; the hope, however, is that when the algorithm fails it still provides some information that helps in solving the problem. In this section we use this strategy to try and learn certain classes of DNF formulae. We will try to see what happens when we run Angluin learning algorithm `LearnMonotone`, on (general) DNF formulae.

In its first iteration the algorithm `LearnMonotone` asks the equivalence query `EQ(0)` and receives some counterexample a such that $f(a) = 1$; then, the algorithm flips 1-bits of a to 0-bits as long as it gets assignments on which the value of f is 1. Let us denote this final assignment $a^{(1)}$ and call it a *local minterm* of f . This assignment $a^{(1)}$ therefore satisfies $f(a^{(1)}) = 1$ and if we set any bit i where $a_i^{(1)} = 1$ to zero then the function value is changed to zero. Notice that since f is not necessarily monotone it is not true that if a is a local minterm of f then for every $b < a$ we have $f(b) < f(a)$. For example, if $f(x_1, \dots, x_5)$ is the function $x_1x_2x_3x_4\bar{x}_5 \vee \bar{x}_1\bar{x}_2$ then a possible counterexample in the first iteration is 11110 which is also a local minterm. In addition, the assignment 00110 is smaller than 11110 and still $f(00110) = 1$.

In its second iteration the algorithm `LearnMonotone` asks the equivalence query `EQ($T_{a^{(1)}}$)` where $T_a = \bigwedge_{a_i=1} x_i$. Since the function f is not monotone we may get a *negative* counterexample, that is an assignment a that satisfies $f(a) = 0$ and $h(a) = 1$ (this is in oppose to the case of monotone functions where the analysis is based on the fact that we only get *positive* counterexamples; that is, a counterexample a that satisfies $f(a) = 1$ and $h(a) = 0$). For example, consider $f(x_1, \dots, x_5)$ as above and the local minterm 11110 then the algorithm asks `EQ($x_1x_2x_3x_4$)` and a possible counterexample is 11111 for which f gives 0 but the hypothesis $x_1x_2x_3x_4$ gives 1. One way to overcome this difficulty is to assume that we have a stronger

query oracle that is guaranteed to provide us with positive counterexamples (or answer YES if no such counterexample exists). We will call this stronger type of query a *positive equivalence query* oracle PEQ. Later, we will show how to get rid of this oracle and work with standard equivalence queries. Therefore, we shall use positive equivalence queries instead of equivalence queries and obtain a new algorithm, denoted **LearnMf**, that is shown in Figure 1. This new algorithm since it gets only positive counterexamples collects more and more local minterms of f . Before analyzing what is the output of algorithm **LearnMf**, we show that the number of local minterms is at most the number of terms of f .

claim 1.1 *Let f be a DNF with r terms. Then, the number of local minterms of f is at most r .*

Proof: Assume that f consists of a single term T . Then, it is easy to see that the only minterm of T is the assignment a in which $a_i = 1$ if x_i in T and $a_i = 0$ otherwise (i.e., if either \bar{x}_i is in T or none of x_i, \bar{x}_i is in T).

Next, assume that $f = T_1 \vee T_2 \vee \dots \vee T_r$ where each T_i is a term. Let a be any local minterm of f . By the definition of local minterm, we know that $f(a) = 1$ which implies that $T_j(a) = 1$ for some j . Also by the definition, we know that whenever we flip a 1-bit of a to 0 the value of f is changed to 0. This implies (since f is a disjunction of terms) that the value of T_j is also changed to 0. Therefore, a is a local minterm of T_j . By the first part of the proof, every term has exactly one local minterm. Thus, the number of local minterms is at most the number of terms. \square

<pre> Algorithm LearnMf: Set $S \leftarrow \emptyset$ and $h \leftarrow 0$. While PEQ(h) \neq YES do Let a be a counterexample. Do For $i = 1, 2, \dots, n$ do If $a_i = 1$ then Set b to be a with $b_i = 0$ If $f(b) = 1$ then $a \leftarrow b$ Until IsMinterm(a) $S \leftarrow S \cup \{a\}$ Set $h = \bigvee_{a \in S} T_a$ Output(h) </pre>
--

Figure 1: Algorithm **LearnMf**

Based on the above claim, algorithm **LearnMf** asks at most $\text{size}_{\text{DNF}}(f)$ positive equivalence queries and at most $n^2 \cdot \text{size}_{\text{DNF}}(f)$ membership queries. When the algorithm halts there are no more positive counterexamples to h . That is, $f \implies h$. Notice also that h is monotone. Denote by $\mathcal{M}(f)$ the minimal monotone function that satisfies $f \implies \mathcal{M}(f)$; that is, $f \implies \mathcal{M}(f)$ and

if $f \implies g \implies \mathcal{M}(f)$ and g is monotone then $g = \mathcal{M}(f)$. (For example, if f is monotone then $\mathcal{M}(f) = f$; if $f(00\dots 0) = 1$ then $\mathcal{M}(f)$ is the constant 1.) What we claim is that algorithm `LearnMf` actually learns the function $\mathcal{M}(f)$.

claim 1.2 *Let h be the output of algorithm `LearnMf` when using oracles for a function f . Then, $h = \mathcal{M}(f)$.*

Proof: First we claim that the minimal monotone function that satisfies $f \implies h$ is unique. Otherwise, if there are two distinct minimal monotone functions h_1 and h_2 then $f \implies h_1$ and $f \implies h_2$ and therefore $f \implies (h_1 \wedge h_2) \implies h_1$. In addition, $h_1 \wedge h_2$ is monotone and so $h_1 = h_2$.

Clearly h , the output of algorithm `LearnMf`, is monotone (since T_a is monotone for every a). Now suppose that h is not minimal. Then, there is a monotone function g such that $f \implies g \implies h$ and $g \neq h$. This implies that there is an assignment b such that $g(b) = 0$ and $h(b) = 1$. In particular, since $f \implies g$ we have $f(b) = 0$. If $h(b) = 1$ then, by the construction of h , the assignment b is above some local minterm a of f . Since $g(b) = 0$ and g is monotone we must have $g(a) = 0$. But since a is a local minterm of f it satisfies $f(a) = 1$ in contradiction to the assumption that $f \implies g$. \square

To summarize what we have proved so far:

Lemma 1.1 *There is a learning algorithm that for every function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ learns the function $\mathcal{M}(f)$ using at most $\text{size}_{\text{DNF}}(f)$ positive equivalence queries and at most $n^2 \cdot \text{size}_{\text{DNF}}(f)$ membership queries.*

Notice that all the definitions that we use, such as the notion of minterm and the notion of monotone functions, depend on the order of assignments in the boolean lattice. However, there is nothing special about this particular order and we can do similar things with respect to other “orders” of the assignments. Formally, let $c \in \{0, 1\}^n$. Define the order $a \geq_c b$ if $a \oplus c \geq b \oplus c$, where \oplus denotes bitwise-xor (note that the standard order is obtained for $c = 00\dots 0$ and that the only minimal point for the order \leq_c is the assignment c itself). We say that f is c -monotone if for every $a \geq_c b$ we have $f(a) \geq f(b)$. We define $\mathcal{M}_c(f)$ to be the minimal c -monotone boolean function that satisfies $f \implies \mathcal{M}_c(f)$. To learn $\mathcal{M}_c(f)$ from positive equivalence queries and membership queries consider the function $g(x) = f(x \oplus c)$; we learn $q = \mathcal{M}(g)$ and then we output $q(x \oplus c)$. We define the algorithm `IsMintermc`(a) that tests if an assignment a is a local minterm of f with respect to the order \leq_c (which is a simple modification of algorithm `IsMinterm`). The algorithm `LearnMfc` that learns $\mathcal{M}_c(f)$ is described in Figure 2. Its analysis is very similar to that of algorithm `LearnMf` and it gives:

Lemma 1.2 *There is a learning algorithm for the class DNF that learns the function $\mathcal{M}_c(f)$ using at most $\text{size}_{\text{DNF}}(f)$ positive equivalence queries and at most $n^2 \text{size}_{\text{DNF}}(f)$ membership queries.*

Before we proceed, we provide a short list of some useful properties of the operator $\mathcal{M}_c(f)$:

```

Algorithm LearnMfc:
Define S ← ∅ and h ← 0.
While PEQ(h) ≠ YES do
  Let d be a counterexample.
  a ← d ⊕ c
  Do
    For i = 1, 2, ..., n do
      If ai = 1 then
        Define b to be a with bi = 0
        If f(b ⊕ c) = 1 then a ← b
  Until IsMintermc(a)
  S ← S ∪ {a}
  Define h = ∨a∈S Ta(x ⊕ c)
Output(h)

```

Figure 2: Algorithm LearnMf_c

Fact 1.1 *Let f and g be boolean functions and c be an assignment. We have*

1. f is c -monotone if and only if $f(x \oplus c)$ is monotone.
2. $\mathcal{M}_c(f) = \mathcal{M}(f(x \oplus c))(x \oplus c)$. That is to find the function $\mathcal{M}_c(f)$ find $g = f(x \oplus c)$, then take $q = \mathcal{M}(g)$ the minimal monotone of g and then take $q(x \oplus c)$. (In fact, this was already used by algorithm LearnMf_c.)
3. If $f \implies g$ then $\mathcal{M}_c(f) \implies \mathcal{M}_c(g)$.
4. $\mathcal{M}_c(f \vee g) = \mathcal{M}_c(f) \vee \mathcal{M}_c(g)$.
5. If C is a clause and $C(a) = 0$ then $\mathcal{M}_c(C) = C$.

Recall that our goal is of-course to learn a function f while what we did so far is learning its monotone extension or more generally learning the function $\mathcal{M}_c(f)$ which all we know about is that it “contains” f (that is $f \implies \mathcal{M}_c(f)$). Also, if for example $f(c) = 1$ then $\mathcal{M}(f) \equiv 1$ which is not very useful. Note however that in such a case it might be that by using a different order c we will have $\mathcal{M}_c(f)$ which is non-constant. Indeed, the hope is that maybe if we learn the function $\mathcal{M}_c(f)$ with respect to many orders c we can combine the outcomes (e.g., by taking their intersection) to get back the function f . The following simple fact shows that this might be possible:

Fact 1.2 *For any boolean function f we have*

$$f = \bigwedge_{c \in \{0,1\}^n} \mathcal{M}_c(f).$$

Proof: On one hand, notice that $f \implies \mathcal{M}_c(f)$ for all c and therefore $f \implies \bigwedge_{c \in \{0,1\}^n} \mathcal{M}_c(f)$. Now if $f(c) = 0$ then $\mathcal{M}_c(f)(a) = 0$ (c is the minimal element in the order \leq_c) and therefore $\bigwedge_{c \in \{0,1\}^n} \mathcal{M}_c(f) \implies f$. \square

This fact by itself is still not enough for efficient learning since it requires learning $\mathcal{M}_c(f)$ for all the 2^n possible orders $c \in \{0,1\}^n$. However, it might be that sometimes one can do much better. Let A be a set of assignments and f be a function. We say that A is a *basis* for f if

$$f = \bigwedge_{c \in A} \mathcal{M}_c(f).$$

We say that a set A is a basis for a class of functions \mathcal{C} if A is a basis for every function $f \in \mathcal{C}$. Fact 1.2 shows that for every class there is a basis (although it might be very big). Now, to learn the class \mathcal{C} that has a basis A , we learn $\mathcal{M}_c(f)$ for all $c \in A$ using the algorithm **LearnMf_c** and then output $\bigwedge_{c \in A} \mathcal{M}_c(f)$ which is exactly f itself. This algorithm uses $|A| \cdot \text{size}_{\text{DNF}}(f)$ positive equivalence queries and $|A| \cdot n^2 \cdot \text{size}_{\text{DNF}}(f)$ membership queries.

We now show how to modify this algorithm into a new algorithm **BasisLearn_A** that learns from standard equivalence queries and membership queries. The algorithm is described in Figure 3; here we provide a description of how it works. Let A be a basis for the class \mathcal{C} . Algorithm **BasisLearn_A** executes the algorithms **LearnMf_c** (for all $c \in A$) until each of them either reaches the point where it asks a positive equivalence query **PEQ**(h_c) or it halts with some output hypothesis h_c . Now we do not have positive equivalence queries available. Therefore, instead of actually asking these positive equivalence queries, algorithm **BasisLearn_A** asks a single (standard) equivalence query **EQ**(H) where $H = \bigwedge_c h_c$. Notice that, because algorithm **LearnMf_c** is learning $\mathcal{M}_c(f)$, we have $h_c \implies \mathcal{M}_c(f)$ (see Exercise 1.1) and since A is a basis we get

$$H = \bigwedge_{c \in A} h_c \implies \bigwedge_{c \in A} \mathcal{M}_c(f) = f.$$

Therefore, as an answer to the query **EQ**(H) algorithm **BasisLearn_A** will get a positive counterexample b ; that is, an assignment b such that $H(b) = 0$ and $f(b) = 1$. Now since $H(b) = 0$ we must have $h_c(b) = 0$ for some (at least one) c and therefore b is an appropriate answer to **PEQ**(h_c) (in the algorithm we denote by S the set of all c 's for which b can be given as a positive counterexample). Then, we can continue to run **LearnMf_c** until it produces a new positive equivalence query (or an output) and repeat the same procedure. Once each of the algorithms **LearnMf_c** produces its output h_c , which by the assumption on these algorithms is exactly $\mathcal{M}_c(f)$ we get

$$H = \bigwedge_{c \in A} h_c = \bigwedge_{c \in A} \mathcal{M}_c(f) = f.$$

This implies the following:

Theorem 1.1 *Let \mathcal{C} be a class of boolean functions and let A be a basis for \mathcal{C} . There is an algorithm that learns \mathcal{C} using $|A| \cdot \text{size}_{\text{DNF}}(f)$ equivalence queries and $|A| \cdot n^2 \cdot \text{size}_{\text{DNF}}(f)$ membership queries.*

In the next subsections we give some interesting classes that have a small size basis. The following simple lemma will be useful for this purpose.

Algorithm BasisLearn_A:

$S \leftarrow A$.

While (TRUE) do

 For all $c \in S$ run LearnMf_c until it asks PEQ(h_c) or outputs h_c

 Ask EQ($\bigwedge_{c \in A} h_c$)

 If the answer is YES then halt with output $\bigwedge_{c \in A} h_c$.

 Otherwise, let b be a counterexample and let $S \leftarrow \{c \in A \mid h_c(b) = 0\}$

 For all $c \in S$ provide LearnMf_c the counterexample b

Figure 3: Algorithm BasisLearn_A

Lemma 1.3 *Let f be a boolean function. Let $C_1 \wedge C_2 \wedge \dots \wedge C_r$ be a CNF formula for f . Let A be a set of assignments such that for every $1 \leq j \leq r$ there exists $c \in A$ such that $C_j(c) = 0$. Then A is a basis for f .*

Proof: By fact 1.1, if $C_j(c) = 0$ then $\mathcal{M}_c(f) \implies \mathcal{M}_c(C_j) = C_j$ and therefore $\bigwedge_{c \in A} \mathcal{M}_c(f) \implies \bigwedge_i C_i \implies f$. Finally, since $f \implies \bigwedge_{c \in A} \mathcal{M}_c(f)$ the lemma follows. \square

Exercise 1.1 *Show that in algorithm LearnMf we always have $h \implies \mathcal{M}(f)$.*

Exercise 1.2 *Determine $\mathcal{M}(f)$ for $f(x_1, \dots, x_n) = x_1 \oplus \dots \oplus x_n$.*

Exercise 1.3 *Let $\mathcal{M}^\partial(f)$ be the maximal monotone function such that $\mathcal{M}^\partial(f) \implies f$. Give an algorithm that learns $\mathcal{M}^\partial(f)$ using negative equivalence queries. Analyze the complexity of your algorithm.*

Exercise 1.4 *Prove Lemma 1.2.*

Exercise 1.5 *Prove Fact 1.1.*

Exercise 1.6 *Show that the output of algorithm BasisLearn is a conjunction of DNF formulae.*

Exercise 1.7 *Show that if C_1 and C_2 have bases A_1 and A_2 respectively then $A_1 \cup A_2$ is a basis for the class $C_1 \wedge C_2 = \{f_1 \wedge f_2 \mid f_1 \in C_1, f_2 \in C_2\}$.*

2 Example: The Class $O(\log n)$ -TERM-DNF

As a first example for the power of the monotone theory, we consider the class k -TERM-DNF. For DNF formulae with a constant number of terms (i.e., $k = O(1)$) we saw that this class is

efficiently learnable in the on-line model and hence can be learned efficiently from equivalence queries only. Now we will show that by using both equivalence queries and membership queries even the class $O(\log n)$ -TERM-DNF is learnable. For this, it suffices to show that this class has a polynomial size basis. There are non-trivial constructions of such bases. Here, instead, we show that a random set of polynomially many assignments has a good chance to be a basis for the class $O(\log n)$ -TERM-DNF. Therefore, if we pick such a random set of assignments A and then run algorithm BasisLearn_A we have a good success probability.

Theorem 2.1 *Let A be a random set of*

$$t = 2^k \left(\ln \binom{n}{k} + \ln 2^k + \ln(1/\delta) \right) = O(2^k(k \log n + \log(1/\delta)))$$

assignments in $\{0, 1\}^n$. Then, with probability at least $1 - \delta$ the set A is a basis for the class k -TERM-DNF. (Notice that t , the size of A , is polynomial in n when $k = O(\log n)$.)

Proof: Recall that k -TERM-DNF $\subseteq k$ -CNF. Let $f = C_1 \wedge C_2 \wedge \dots \wedge C_m$ be a CNF representation of some function $f \in k$ -TERM-DNF where each clause C_i is of size at most k . By Lemma 1.3 if for every clause C_i there is an assignment $c \in A$ such that $C_i(c) = 0$ then A is a basis for the function f . Therefore, if for every clause C of size at most k there is $c \in A$ such that $C(c) = 0$ then A is a basis for the class k -TERM-DNF. Equivalently, A is a basis for the class k -TERM-DNF if and only if for every $1 \leq i_1 < i_2 < \dots < i_k \leq n$ and every $z \in \{0, 1\}^k$ there is $c \in A$ such that $(c_{i_1}, \dots, c_{i_k}) = z$. Fix some indices i_1, \dots, i_k and a set of values for them $z \in \{0, 1\}^k$. The probability that t random assignments (i.e., the set A) do not contain an assignment c such that $(c_{i_1}, \dots, c_{i_k}) = z$ is at most $(1 - \frac{1}{2^k})^t$. The probability that t random assignments do not contain an assignment c such that $(c_{i_1}, \dots, c_{i_k}) = z$ for some choice of i_1, \dots, i_k and z is at most $\binom{n}{k} \cdot 2^k$ larger. Therefore, the probability that a random A of size t is not a basis is, by the choice of t , at most

$$\binom{n}{k} \cdot 2^k \cdot (1 - 2^{-k})^t \leq \delta.$$

□

Exercise 2.1 *Let j -Almost MDNF be the class of all DNF formulae that contains at most j Almost-nonmonotone terms. Show that, for constant j , the class j -MDNF has a polynomial size basis.*

3 Learning Decision-Trees

In this section we demonstrate, again, the power of the monotone theory by giving an efficient algorithm that learns the class DT of decision trees. More generally, the algorithm learns the class of all functions that have both a polynomial-size DNF formula and a polynomial-size CNF formula.

Algorithm LearnCDNF:

$A \leftarrow \emptyset$.

While (TRUE) do

Run **BasisLearn_A** until the equivalence queries receives a negative counterexample b .

$A \leftarrow A \cup \{b\}$.

Figure 4: Algorithm LearnCDNF

We give an algorithm called **LearnCDNF** which is described in Figure 4. The main idea here is that instead of using the same basis for the whole class it is sometimes possible to construct a basis for the particular target function f by analyzing the answers to the queries. At any given point during the algorithm, we will have a set A which is a subset of some small basis for f . Then, we will try to run the algorithm **BasisLearn_A** on the target function f using this set A . (We start with $A = \emptyset$.) Recall that if A is a basis then **BasisLearn_A** expects to get only positive counterexamples and that as long as it gets only positive counterexamples it can proceed. Now, suppose that for some equivalence query $\text{EQ}(h)$ the algorithm gets a negative counterexample b (that is, $f(b) = 0$ but $h(b) = 1$). The fact that $f(b) = 0$ means that in any CNF representation $C_1 \wedge C_2 \wedge \dots \wedge C_r$ of f , there is (at least one) i such that $C_i(b) = 0$. Therefore, this assignment b can be used as an element in a basis for f (see Lemma 1.3). We add b to A and again run **BasisLearn_A**. The following claim shows that by adding this b to A we indeed make a significant progress. More precisely, it shows that every negative counterexample b falsify a new clause that no previous negative counterexample did. Therefore, after $\text{size}_{\text{CNF}}(f)$ counterexamples we will have a basis for f . Formally,

claim 3.1 *Let $f = C_1 \wedge C_2 \wedge \dots \wedge C_r$ be any CNF formula for f . Suppose that we have a set A such that for every $i \in \{i_1, \dots, i_t\}$ there is an assignment $c \in A$ that falsify the clause C_i , i.e., $C_i(c) = 0$. Suppose we are running the algorithm **BasisLearn_A** and we get a negative counterexample b . Then, b falsifies a new clause C_j , for some $j \notin \{i_1, \dots, i_t\}$.*

Proof: Recall that algorithm **BasisLearn_A** tries to learn $\mathcal{M}_c(f)$ for each $c \in A$ by using the corresponding algorithm **LearnMf_c**. Let h_c be the hypothesis of **LearnMf_c** when the negative counterexample b is received. Since b is a negative counterexample then $f(b) = 0$ while $H(b) = 1$, where $H = \bigwedge_{c \in A} h_c$. In particular, $h_c(b) = 1$ for all $c \in A$. Since $h_c \implies \mathcal{M}_c(f)$ we also have $\mathcal{M}_c(f)(b) = 1$ for all $c \in A$. Now, let C_i be any clause where $i \in \{i_1, \dots, i_t\}$ and let $c' \in A$ be such that $C_i(c') = 0$. Since $f \implies C_i$, by Fact 1.1 we have

$$1 = \mathcal{M}_{c'}(f)(b) \implies \mathcal{M}_{c'}(C_i)(b) = C_i(b)$$

and therefore $C_i(b) = 1$. Since $f(b) = 0$ we must have $C_i(b) = 0$ for some $i \notin \{i_1, \dots, i_t\}$. \square

Theorem 3.1 *There is a learning algorithm that learns any boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ using $\text{size}_{\text{CNF}}(f) \cdot \text{size}_{\text{DNF}}(f)$ equivalence queries and $\text{size}_{\text{CNF}}(f) \cdot \text{size}_{\text{DNF}}(f) \cdot n^2$ membership queries.*

Exercise 3.1 *Give an efficient algorithm for learning the class DT of decision trees.*

4 OPEN PROBLEMS

1. Show that k -term DNF is learnable from membership and equivalence queries for some $k = \omega(\log n)$.
2. Show that k -Almost MDNF is learnable for some $k = \omega(1)$.