Learning Decision Trees using the Fourier Spectrum[*]

Eyal Kushilevitz[†] Yishay Mansour[‡]

Abstract

This work gives a *polynomial* time algorithm for learning *decision trees* with respect to the uniform distribution. (This algorithm uses *membership queries.*) The decision tree model that is considered is an extension of the traditional boolean decision tree model that allows linear operations in each node (i.e., summation of a subset of the input variables over $GF(2)$).

This paper shows how to learn in polynomial time any function that can be approximated (in norm L_2) by a polynomially sparse function (i.e., a function with only polynomially many non-zero Fourier coefficients). The authors demonstrate that any function f whose L_1-norm (i.e., the sum of absolute value of the Fourier coefficients) is polynomial can be approximated by a polynomially sparse function, and prove that boolean decision trees with linear operations are a subset of this class of functions. Moreover, it is shown that the functions with polynomial L_1-norm can be learned *deterministically.*

The algorithm can also *exactly identify* a decision tree of depth d in time polynomial in 2^d and n. This result implies that trees of logarithmic depth can be identified in polynomial time.

1 Introduction

In recent years much effort has been devoted to providing a theoretical basis for machine learning. These efforts involved formalization of learning models and algorithms, with a special emphasis on polynomial running time algorithms (see [Val84, Ang87]). This work further extends our understanding of the learning tasks that can be performed in polynomial time.

Recent work by [LMN89] has established the connection between the Fourier spectrum and learnability. They presented a quasi-polynomial-time (i.e. $O(n^{poly-log(n)})$) algorithm for learning

[*]A preliminary version appeared in the Proceedings of the 23^{rd} Annual ACM Symposium on Theory of Computing, pages 455-464, May 1991.

[†]Dept. of Computer Science, Technion, Haifa 32000, Israel. Current address: Aiken Computation Laboratory, Harvard University, Cambridge MA 02138.

[‡]Aiken Computation Laboratory, Harvard University, Cambridge MA 02138. Partially supported by ONR N00014-85-K-0445

the class AC^0 (polynomial size constant depth circuits), where the quality of the approximation is judged with respect to the uniform distribution (and n is the number of variables). Their main result is an interesting property of the representation of the Fourier transform of AC^0 circuits. Using this property, they derive the learning algorithm for this class of functions. [FJS91] has extended the result to apply also to mutually independent distributions (i.e., product distributions) with a similar running time (i.e. quasi-polynomial time). In [AM91] polynomial time algorithms are given for learning both decision *lists* and μ-decision trees (a boolean decision tree in which each variable appears only *once*) with respect to the uniform distribution. As in [LMN89] these algorithms make use of special properties of the Fourier coefficients and approximate the target function by observing examples drawn according to the uniform distribution. More information about Fourier transform over finite groups is found in [Dia88].

In this work we show another interesting application of the Fourier representation that is applied to achieve learnability. The learning model allows membership queries, where the learner can query the (unknown) function on any input. Our main result is a polynomial-time algorithm for learning functions computed by *boolean decision trees* with *linear operations* (over $GF(2)$). In these trees each node computes a summation (modulo 2) of a subset of the n boolean input variables, and branches according to whether the sum is zero or one. Clearly, this is an extension of the traditional boolean decision-tree model, since we can still test single variables. On the other hand, we can test in a single operation the parity of all the input variables, compared with a lower bound of 2^n nodes in the traditional model (see [BHO90]).

An interesting consequence of our construction is that one can *exactly* find the Fourier transform representation of boolean decision trees with linear operations in time $poly(n, 2^d)$, where d is the depth of the tree. This implies that we find a function that is *identical* to the tree for any boolean input. A corollary of this result is that decision trees with logarithmic depth can be exactly identified in polynomial time. (Note that enumeration, even of constant depth trees, would require exponential time (due to the linear operation); even eliminating the linear operations and constraining each node to contain a single variable, the number of trees of depth d is $\Omega(n^d)$.)

Our main result – the learning algorithm for decision trees – is achieved by combining the following three results:

- *The algorithmic tool* – We present a randomized polynomial time algorithm that performs the following task. The algorithm receives as an input a boolean function f that can be approximated by a polynomially sparse function g (a function with a polynomial number of non-zero Fourier coefficients) such that the expected error square (i.e. $E(f-g)^2$) is bounded by ε. The algorithm finds some polynomially sparse function h that approximates f, such that $E(f-h)^2 = O(\varepsilon)$. The algorithm we develop here is based on the ideas of [GL89].

- We consider the class of functions $\mathcal{F} = \{f : L_1(f) \leq poly(n)\}$, where $L_1(f)$ is the L_1-norm of the coefficients (i.e., the sum of the absolute value of the coefficients). We show that in order to achieve an approximation of a function $f \in \mathcal{F}$ within ε, it is sufficient to consider only coefficients larger than $\frac{\varepsilon}{L_1(f)}$ (there are at most $(\frac{L_1(f)}{\varepsilon})^2$ such coefficients). Therefore, every function in the class $\mathcal{F}$ can be approximated by a polynomially sparse function and therefore can be learned in polynomial time by our algorithm.

- We prove that the L_1-norm of the coefficients of a decision tree is bounded by the number of nodes in the tree. Therefore, polynomial size decision trees are in the class $\mathcal{F}$. It follows that every polynomial size decision tree with linear operations can be learned in polynomial time.

Furthermore, for functions in the class $\mathcal{F}$ we show how to derandomize the learning algorithm. The derandomization uses constructions of "small", "almost unbiased" probability spaces, called λ – bias distributions [NN90, AGHP90]. (For a formal definition of λ – bias probability distributions see Section 4.1.) Thus, we derive a *deterministic* polynomial time algorithm for learning decision trees.

Our technique sheds a new light on the possibilities of using λ – bias distributions for derandomization. We show that the deviation of the expected value of a function f with respect to the uniform distribution and a λ – bias distribution is bounded by $\lambda \cdot L_1(f)$. One nice example where this bound comes in handy is for showing that the deviation of the AND of a subset of the n variables is bounded by 3λ. (This is since $L_1(AND) \leq 3$, independent of the subset of variables or its size.)

1.1 Relations to Other Works

Our result could be contrasted with the result of [EH89], where an $O(n^{\log m})$ algorithm is given for learning decision trees in the PAC model, where n is the number of variables and m is the number of nodes in the tree. Their algorithm learns traditional boolean decision trees with respect to an arbitrary distribution, and uses only examples drawn from that distribution. Therefore, it learns in a weaker model. On the other hand, it runs in time $O(n^{\log m})$ compared to the polynomial time of our algorithm. Also, our algorithm handles a stronger model of boolean decision trees, which include linear operations, while the algorithm of [EH89] does not seem to extend to such a model. In [Han90] a polynomial-time algorithm was presented for learning μ-decision trees using membership queries and equivalence queries, and in [Han91] a polynomial time algorithm was presented for learning decision trees in which each variable appears at most a constant number of times. (Again, these results do not address linear operations.)

Recently, Bellare [Bel92] was able to extend a few of our results concerning decision trees, and show how to derive an upper bound on the sum of the Fourier coefficients as a function of the predicates in the nodes. He also extends the learning algorithm to the case of product distributions, and show that if the L_1-norm of f (with respect to a product distribution μ) is polynomially bounded then it can be learned (with respect to μ) in polynomial time. Unfortunately, this result falls short of showing that decision-trees are learnable with respect to product distributions, since there are functions (e.g., the AND function) that have a small size decision tree but their L_1-norm is exponential with respect to some product distributions.

Following our work, it has been shown [Man92] how to learn DNF formulas, with respect to the uniform distribution, in $O(n^{\log\log n})$ time. The main contribution of that work is made by bounding the number of "large" coefficients in the Fourier expansion of such a function by $O(n^{\log\log n})$. Then, the algorithm of this paper is used to recover them.

In the work of [RB91] the same learning model was considered (i.e., using membership queries and testing the hypothesis with respect to the uniform distribution). They show that any polynomial over $GF(2)$ with polynomial number of terms can be learned in polynomial time in such a model. The class of polynomials with polynomial number of terms (considered in [RB91]) and the class of boolean decision trees with linear operations (considered in our work) are incomparable. On the one hand, the inner-product function has a small polynomial but does not have a small decision tree. On the other hand, consider a boolean decision list with $\log n$ nodes, where each node computes the sum of $\Omega(n)$ variables. Representing such a decision list by a polynomial may require $\Omega(n^{\log n})$ terms.

The power of polynomial size boolean decision trees with linear operations is also incomparable to AC^0 circuits (which are the target of the learning algorithm of [LMN89]). Such trees can compute parity, which cannot be approximated by AC^0 circuits (see [FSS84, Ajt83, Yao85, Has86]). We show that for boolean decision trees with linear operations the L_1-norm is bounded by the number of nodes; therefore, computing a polynomial-size DNF that has an exponential L_1-norm would require an exponential number of nodes (see [BS90] for a construction of such a DNF).

The class $\mathcal{F}$ of boolean functions whose L_1-norm is polynomially bounded was also studied in [Bru90, BS90, SB91]. They showed that any such function f can be approximated by a sparse polynomial of a certain form. Note, however, that their notion of approximation is different than ours. Another type of approximation for boolean functions was recently suggested in [ABFR91] (and then studied by others). In that work boolean functions are approximated by the sign of a low-degree polynomial over the integers.

1.2 Organization

The rest of this paper is organized as follows. Section 2 has the definitions of Fourier transform, decision trees and the learning model. Section 3 includes the procedure that finds the approximating sparse function. In section 4 we prove the properties of functions with small L_1-norm. In section 5 we prove the results about boolean decision trees with linear operations. Finally, in section 6 we discuss some extensions and mention some open problems.

2 Preliminaries

In this section we give the definition of Fourier transform and recall some known properties of it (section 2.1). Then, we formally define the model of decision trees which is used in this work (section 2.2). We end by describing the membership-queries learning model which is used in this work (section 2.3).

2.1 Fourier Transform

Let $f : \{0,1\}^n \to \Re$ be a real function. Denote by $E[f]$ the expected value of $f(x)$ with respect to the uniform distribution on x, i.e., $E[f] = \frac{1}{2^n}\sum_{x\in\{0,1\}^n} f(x)$. The set of all real functions on the cube Z_2^n is a 2^n-dimensional real vector space with an inner product defined by:

$$< g, f > \triangleq \frac{1}{2^n} \sum_{x \in \{0,1\}^n} g(x) f(x) = E[gf].$$

The *norm* of a function f is defined by $\|f\|_2 \triangleq \sqrt{< f, f >} = E[f^2]$.

Define a basis for the linear space of real functions on the cube Z_2^n, using the *characters* of Z_2^n as follows: For each $z \in \{0,1\}^n$, define the basis function χ_z:

$$\chi_z(x_1, \cdots, x_n) \triangleq \begin{cases} +1 & \text{if } \sum_i z_i x_i \bmod 2 = 0 \\ -1 & \text{if } \sum_i z_i x_i \bmod 2 = 1 \end{cases}$$

The following properties of these functions can be verified easily:

- For every two vectors $z_1, z_2 \in \{0,1\}^n$: $\chi_{z_1}\chi_{z_2} = \chi_{z_1 \oplus z_2}$, where $\oplus$ denotes bitwise exclusive-or.
- The family of functions $\{\chi_z : z \in \{0,1\}^n\}$ forms an orthonormal basis. That is, (1) any function $f(x)$ on the cube Z_2^n can be uniquely expressed as $\sum_z \hat{f}(z)\chi_z(x)$, where $\hat{f}(z)$ are real constants; and (2) if $z_1 \neq z_2$, then $< \chi_{z_1}, \chi_{z_2} >= 0$, and for every z, $< \chi_z, \chi_z >= 1$.

The Fourier transform of f is just the expansion of f as a linear combination of the χ_z's. Since the χ_z's are an orthonormal basis, Fourier coefficients are found via

$$\hat{f}(z) =< f, \chi_z >= E[f\chi_z].$$

The orthonormality of the basis implies *Parseval's identity*:

$$E[f^2] = \|f\|_2^2 = \sum_{z \in \{0,1\}^n} \hat{f}^2(z).$$

Note that if for every $x \in Z_2^n$, $|f(x)| \leq 1$, then $\|f\|_2 \leq 1$ and therefore for every $z \in \{0,1\}^n$, $|\hat{f}(z)| \leq 1$. Finally, we define $L_1(f)$ as the L_1 norm of the coefficients of f, i.e. $L_1(f) \triangleq \sum_z |\hat{f}(z)|$.

2.2 Boolean decision trees

In this section we give a precise definition of the *decision tree* model used in this work. This model is much stronger than the traditional decision tree model.

A *boolean decision tree* T consists of a labeled binary tree. Each inner node v of the tree is labeled by a set $S_v \subseteq \{1, \ldots, n\}$, and has two outgoing edges. Every leaf of the tree is labeled by either $+1$ or -1. (Throughout this paper a function is called *boolean* if its range is $\{+1, -1\}$.)

Given an input, $x = (x_1, \ldots, x_n)$, the decision tree defines a *computation*. The computation traverses a path from the root to a leaf and assigns values to the nodes on the path in the following way. The computation starts at the root of the tree T. When the computation arrives at an inner node v, labeled by S_v, it assigns the node v the value $\sum_{i \in S_v} x_i \bmod 2$, which we denote by $val(v)$. If $val(v) = 1$ then the computation continues to the right son of v, otherwise it continues to the left son. The computation terminates at a leaf u and outputs the label of u (which is also the value of the leaf). The value of the tree on an input is the value of the output of the computation.

Note that if, for example, $|S_v| = 1$ then the meaning of the operation is testing the value of a single variable which is the only permitted operation in the traditional decision tree model. If, for example, $|S_v| = 2$ then the meaning of the operation is testing whether the two corresponding variables are equal, and if $|S_v| = n$ then in a single operation we have a test for the parity of all variables. In the traditional decision tree model computing the parity of all the variables requires 2^n nodes.

2.3 Learning Model

The learner in our model uses only membership queries. That is, it can query the unknown function f on any input $x \in \{0,1\}^n$ and receive $f(x)$. After performing a finite number of membership queries, the learner outputs an hypothesis h. The error of an hypothesis h, with respect to the function f, is defined to be $error(f,h) \triangleq Prob[f(x) \neq h(x)]$, where x is distributed uniformly over $\{0,1\}^n$.

A randomized algorithm A *learns* a class of functions $\mathcal{F}$ if for every $f \in \mathcal{F}$ and $\varepsilon, \delta > 0$ the algorithm outputs an hypothesis $h = A(f, \varepsilon, \delta)$ such that

$$Prob[error(f,h) \geq \varepsilon] \ \leq \ \delta.$$

The algorithm A learns in *polynomial time* if its running time is polynomial in n, $1/\varepsilon$, and $\log 1/\delta$.

We also discuss deterministic learning algorithms. An algorithm A *deterministically learns* a class of functions $\mathcal{F}$ if for every $f \in \mathcal{F}$ and $\varepsilon > 0$ the algorithm outputs an hypothesis $h = A(f, \varepsilon)$ such that

$$error(f,h) \leq \varepsilon.$$

The algorithm A learns in *deterministic polynomial time* if its running time is polynomial in n and $1/\varepsilon$. Note that in a deterministic algorithm we do not have a parameter δ. That is, the algorithm always succeeds in finding a "good" hypothesis.

A (real) function g *ε-approximates* f (in norm L_2) if $E[(f(x) - g(x))^2] \leq \varepsilon$. In the case that f is a boolean function, we can convert a real prediction function g to a boolean prediction by predicting the sign of g. In such a case, if $f(x) \neq sign(g(x))$ then $|f(x) - g(x)| \geq 1$, which implies

$$Prob[f(x) \neq sign(g(x))] \leq E[(f(x) - g(x))^2] \leq \varepsilon.$$

Thus, we have

Claim 2.1 *If g ε-approximates a boolean function f, then*

$$Prob[f(x) \neq sign(g(x))] \leq \varepsilon.$$

3 Approximation by sparse functions

In this section we show how to find an approximation by a sparse function. The main result in this section is that if f can be ε-approximated by some polynomially-sparse function g then there is a randomized polynomial time procedure that finds some function h that $O(\varepsilon)$-approximates f. (A function g is *t-sparse* if it has at most t Fourier coefficients that are not zero.)

The first step is to show that if f can be approximated by a polynomially sparse function g, it can be approximated by a polynomially sparse function that has only "large" coefficients. We remark that we do not make a "direct" use of g (e.g., by approximating g instead of approximating f) but only use its existence in the analysis.

Lemma 3.1 *If f can be approximated by a t-sparse function g such that $E[(f-g)^2] \leq \varepsilon$, then there exists a t-sparse function h such that $E[(f-h)^2] \leq \varepsilon + O(\varepsilon^2/t)$ and all the non-zero coefficients of h are at least ε/t.*

Proof: Let $g(x) = \sum_{i=1}^{t} \hat{g}(z_i)\chi_{z_i}(x)$. Note that the Fourier coefficients of the function $f-g$ are exactly $\hat{f}(z) - \hat{g}(z)$. Therefore, by Parseval's equality,

$$E[(f-g)^2] = \sum_z (\hat{f}(z) - \hat{g}(z))^2.$$

Thus, requiring that $\hat{g}(z_i) = \hat{f}(z_i)$ can only reduce the expected error squared. Therefore, without loss of generality, the non-zero coefficients of g are the coefficients of f, i.e. $g(x) = \sum_{i=1}^{t} \hat{f}(z_i)\chi_{z_i}(x)$. Let h be the function obtained from g by replacing the "small" coefficients by 0. Namely,

$$h(x) = \sum_{\hat{f}(z_i) \geq \varepsilon/t} \hat{f}(z_i)\chi_{z_i}(x).$$

We now show that $E[(f-h)^2] \leq \varepsilon + O(\varepsilon^2/t)$. Consider the expression,

$$E[(f-h)^2] - E[(f-g)^2].$$

By the above arguments, this is equal to

$$\sum_z (\hat{f}(z) - \hat{h}(z))^2 - \sum_z (\hat{f}(z) - \hat{g}(z))^2 = \sum_{\hat{f}(z_i) < \varepsilon/t} (\hat{f}(z_i) - \hat{h}(z_i))^2 = \sum_{\hat{f}(z_i) < \varepsilon/t} \hat{f}^2(z_i) < (\frac{\varepsilon}{t})^2 t = \varepsilon^2/t.$$

Since $E[(f-g)^2] \leq \varepsilon$, the lemma follows. □

The above lemma has reduced the problem of approximating f by a t-sparse function to the problem of finding all the coefficients of f that are greater than a threshold of ε/t. Note that the function h defined above does not necessarily contain all the coefficients of f that are greater than ε/t, but only those that appear also in g. However, adding these coefficients to h will clearly make h a better approximation for f. In any case, the number of these coefficients, as follows from Lemma 3.4 below, cannot be too high.

In the remainder of this section we show a randomized polynomial time procedure that given a function f and a threshold θ outputs (with prob. $1-\delta$) all the coefficients for which $|\hat{f}(z)| \geq \theta$.

The procedure runs in polynomial time in n, $1/\theta$ and $\log 1/\delta$. This procedure is based on the ideas of [GL89], although the context is completely different.

Let $f(x) = \sum_{z\in\{0,1\}^n} \hat{f}(z)\chi_z(x)$. For every $\alpha \in \{0,1\}^k$, we define the function $f_\alpha : \{0,1\}^{n-k} \to \Re$ as follows:

$$f_\alpha(x) \triangleq \sum_{\beta\in\{0,1\}^{n-k}} \hat{f}(\alpha\beta)\chi_\beta(x).$$

In other words, the function $f_\alpha(x)$ includes all the coefficients $\hat{f}(z)$ of f such that z starts with α (and all the other coefficients are 0). This immediately gives the key idea for how to find the large coefficients of f: find (recursively) the large coefficients of f_0 and f_1. Note that during the learning process we can only query for the value of the target function f in certain points. Therefore, we first have to show that $f_\alpha(x)$ can be efficiently computed using such queries to f. Actually, we need not compute the exact value of $f_\alpha(x)$ but just need to approximate it. The following lemma gives an equivalent formulation of f_α, which is computationally much more appealing:

Lemma 3.2 *For any function f, any $1 \le k < n$, any $\alpha \in \{0,1\}^k$, and any $x \in \{0,1\}^{n-k}$,*

$$f_\alpha(x) = E_{y\in\{0,1\}^k}[f(yx)\chi_\alpha(y)].$$

This formulation implies that even though we do not know how to compute the value of $f_\alpha(x)$ we can approximate it, by approximating the above expectation.

Proof: Let $f(yx) = \sum_z \hat{f}(z)\chi_z(yx)$. Note that if $z = z_1z_2$, where $z_1 \in \{0,1\}^k$, then $\chi_z(yx) = \chi_{z_1}(y)\chi_{z_2}(x)$. Therefore,

$$\begin{aligned} E_y[f(yx)\chi_\alpha(y)] &= E_y\left[\left(\sum_{z_1}\sum_{z_2}\hat{f}(z_1z_2)\chi_{z_1}(y)\chi_{z_2}(x)\right)\chi_\alpha(y)\right] \\ &= \sum_{z_1}\sum_{z_2}\hat{f}(z_1z_2)\chi_{z_2}(x)E_y[\chi_{z_1}(y)\chi_\alpha(y)], \end{aligned}$$

where y and z_1 denote strings in $\{0,1\}^k$ and z_2 denotes strings in $\{0,1\}^{n-k}$. By the orthonormality of the basis, (see Section 2.1) it follows that $E_y[\chi_{z_1}(y)\chi_\alpha(y)]$ (which is the same as $<\chi_{z_1}, \chi_\alpha>$) equals 0 if $z_1 \ne \alpha$, and equals 1 if $z_1 = \alpha$. Therefore, only the terms with $z_1 = \alpha$ contributes in the sum. Thus, the last term equals

$$\sum_{z_2\in\{0,1\}^{n-k}} \hat{f}(\alpha z_2)\chi_{z_2}(x) = f_\alpha(x).$$

□

Since both $|f(x)| = 1$ and $|\chi_\alpha(y)| = 1$ we derive the following corollary on the value of $f_\alpha(x)$.

Corollary 3.3 *For any boolean function f, any $1 \le k < n$, any $\alpha \in \{0,1\}^k$, and any $x \in \{0,1\}^{n-k}$,*

$$|f_\alpha(x)| \le 1.$$

```
SUBROUTINE Coef($\alpha$)
        IF $E[f_\alpha^2] \geq \theta^2$ THEN
                IF $|\alpha| = n$ THEN OUTPUT $\alpha$
                        ELSE Coef($\alpha 0$); Coef($\alpha 1$);
```

Figure 1: Subroutine Coef

We showed how to decompose a function f into functions f_α, $\alpha \in \{0,1\}^k$, such that each coefficient of f appears in a unique f_α. Recall that our aim is to find the coefficients $\hat{f}(z)$ such that $|\hat{f}(z)| \geq \theta$. The next lemma claims that this cannot hold for "too many" values of z, and that the property $E[f_\alpha^2] \geq \theta^2$ cannot hold for "many" α (of length k) simultaneously.

Lemma 3.4 *Let f be a* boolean *function, and $\theta > 0$. Then,*

1. *At most $1/\theta^2$ values of z satisfy $|\hat{f}(z)| \geq \theta$.*
2. *For any $1 \leq k < n$, at most $1/\theta^2$ functions f_α with $\alpha \in \{0,1\}^k$ satisfy $E[f_\alpha^2] \geq \theta^2$.*

Proof: By the assumption that f is a boolean function combined with Parseval's equality, we get

$$\sum_{z \in \{0,1\}^n} \hat{f}^2(z) = E[f^2] = 1.$$

Therefore, (1) immediately follows. Similarly, using the definition of f_α,

$$E[f_\alpha^2] = \sum_{\beta \in \{0,1\}^{n-k}} \hat{f}^2(\alpha\beta).$$

Thus, if $|\hat{f}(\alpha\beta)| \geq \theta$, for some $\beta \in \{0,1\}^{n-k}$, then $E[f_\alpha^2] \geq \theta^2$. By the above two equalities the following holds,

$$\sum_{\alpha \in \{0,1\}^k} E[f_\alpha^2] = E[f^2] = 1.$$

Therefore, at most $1/\theta^2$ functions f_α have $E[f_\alpha^2] \geq \theta^2$, which completes the proof of (2). □

The Algorithm

By now the algorithm for finding the large coefficients of a function f should be rather obvious. It is described by the recursive subroutine Coef, which appears in Figure 1. We start the algorithm by calling Coef(λ), where λ is the empty string.

As mentioned earlier in this section, we know that each coefficient of f_α appears in exactly one of $f_{\alpha 0}$ and $f_{\alpha 1}$, and also that if $|\hat{f}(\alpha\beta)| \geq \theta$, for some $\beta \in \{0,1\}^{n-k}$, then $E[f_\alpha^2] \geq \theta^2$ (note that when $|\alpha| = n$, then $E[f_\alpha^2] = \hat{f}^2(\alpha)$). Therefore the correctness of the algorithm follows; namely, the algorithm outputs all the required coefficients.

By lemma 3.4, we also know that the number of α's for which $E[f_\alpha^2] \geq \theta^2$ is bounded by $1/\theta^2$, for each length of α. Thus, the total number of recursive calls is bounded by $O(n/\theta^2)$.

```
SUBROUTINE Coef($\alpha$)
            $B_\alpha$ = Approx($\alpha$) /* $B_\alpha$ is approximating $E[f_\alpha^2]$.*/
            IF $B_\alpha \geq \theta^2/2$ THEN
                        IF $|\alpha| = n$ THEN OUTPUT $\alpha$
                                    ELSE Coef($\alpha$0); Coef($\alpha$1);
```

Figure 2: The Modification of Subroutine `Coef`

Detailed analysis

We are still not done, since this algorithm assumes that we can compute $E[f_\alpha^2]$ exactly, something that is not achievable in polynomial time. On the other hand, we can approximate $E[f_\alpha^2]$ very accurately in polynomial time. Therefore we modify Subroutine `Coef`: instead of testing whether $E[f_\alpha^2] \geq \theta^2$ we approximate $E[f_\alpha^2]$ and test whether the approximated value is greater than $\theta^2/2$ (see Figure 2).

The approximation of $E[f_\alpha^2]$ is such that with very high probability the error in the approximation is small. That is, with high probability, every coefficient satisfying $|\hat{f}(z)| \geq \theta$ will be output, which guarantees the correctness condition of the algorithm. Also, this approximation guarantees that with high probability, no coefficient satisfying $|\hat{f}(z)| \leq \theta/2$ will be output, which bounds (by lemma 3.4) the number of coefficients the algorithm outputs to at most $4/\theta^2$. Moreover, it implies that for every k at most $4/\theta^2$ strings $\alpha \in \{0,1\}^k$ of length k will pass the test of the subroutine, which bounds the number of calls to the recursive subroutine by $O(n/\theta^2)$. What we are left with is to bound the computation required to approximate $E[f_\alpha^2]$.

Let m_1, m_2 be parameters (to be fixed later). We approximate $E[f_\alpha^2(x)]$ as follows.

```
SUBROUTINE Approx($\alpha$)
Choose at random $x_i \in \{0,1\}^{n-k}$, for $1 \leq i \leq m_1$.
For each $x_i$
        Choose at random $y_{i,j} \in \{0,1\}^k$, for $1 \leq j \leq m_2$.
        Let $A_i = \frac{1}{m_2}\sum_{j=1}^{m_2} f(y_{i,j}x_i)\chi_\alpha(y_{i,j})$.
                                        /* $A_i$ is approximating */
                                        /* $f_\alpha(x_i) = E_y[f(yx_i)\chi_\alpha(y)]$.*/
Let $B_\alpha = \frac{1}{m_1}\sum_{j=1}^{m_1} A_i^2$.
                                        /* $B_\alpha$ is approximating $E_x[f_\alpha^2(x)]$.*/
RETURN $B_\alpha$.
```

The value of B_α is the approximation to $E[f_\alpha^2(x)]$. We now need to find the "right" values of m_1 and m_2, such that B_α will be a "good" approximation to $E[f_\alpha^2(x)]$. That is, with high probability, if $E[f_\alpha^2] \geq \theta^2$ then $B_\alpha \geq \theta^2/2$ and if $E[f_\alpha^2] \leq \theta^2/4$ then $B_\alpha < \theta^2/2$.

To prove that B_α is a "good" approximation of $E_x[f_\alpha^2(x)]$, we first prove that B_α would be a "good" approximation of $E_x[f_\alpha^2(x)]$ if we compute it with the real values of $f_\alpha(x_i)$. Then we show that the A_i's are "good" approximations for $f_\alpha(x_i)$. Finally, we show that even if we compute B_α with the A_i's (instead of $f_\alpha(x_i)$) it is still a "good" approximation of $E_x[f_\alpha^2(x)]$. For the proof we use Chernoff bounds (see [HR89]):

Lemma 3.5 (Chernoff) *Let $X_1, \ldots, X_m$ be independent, identically distributed random variables, such that $E[X_i] = p$ and $S_m = \sum_{i=1}^m X_i$.*

- *If $X_i \in [0,1]$ then*
$$Prob\left[(1-\varepsilon)\cdot p \le \frac{S_m}{m} \le (1+\varepsilon)\cdot p\right] \ge 1 - 2e^{-\varepsilon^2 mp/3}$$
- *If $X_i \in [-1,+1]$ then*
$$Prob\left[|\frac{S_m}{m} - p| \ge \lambda \cdot \frac{1}{\sqrt{m}}\right] \le 2e^{-\lambda^2/2}.$$

Using this bound, we claim that by choosing at random m_1 values x_i and computing the average $f_\alpha^2(x_i)$, we get a value that is very close to $E[f_\alpha^2]$.

Lemma 3.6 *Let $B'_\alpha = \frac{1}{m_1}\sum_{j=1}^{m_1} f_\alpha^2(x_i)$, where $x_i \in \{0,1\}^{n-k}$, $1 \le i \le m_1$, are chosen uniformly at random. Then,*
$$Prob\left[\frac{3}{4}E[f_\alpha^2] \le B'_\alpha \le \frac{5}{4}E[f_\alpha^2]\right] \ge 1 - 2e^{-\frac{m_1}{48}\cdot E[f_\alpha^2]}.$$

Proof: Follows immediately from the first part of Lemma 3.5 with $\varepsilon = \frac{1}{4}$ (and $p = E[f_\alpha^2]$). □

The next lemma claims that A_i is a "good" approximation for $f_\alpha(x_i)$. It is based on the identity of Lemma 3.2 (i.e., $f_\alpha(x_i) = E_y[f(yx_i)\chi_\alpha(y)]$).

Lemma 3.7 *For any value of x_i,*
$$Prob\left[|A_i - f_\alpha(x_i)| \ge \theta^2/16\right] \le 2e^{-\theta^4 m_2/2^9}$$

Proof: Follows immediately from the second part of Lemma 3.5 with $\lambda = \frac{\theta^2\sqrt{m_2}}{16}$. □

Intuitively, if we approximate each $f_\alpha(x_i)$ well, the difference between B_α (which uses the approximate values) and B'_α (which uses the true values) should be small. The following lemma formalizes this intuition.

Lemma 3.8 *If $|f_\alpha(x_i) - A_i| \le \frac{\theta^2}{16}$, for $1 \le i \le m_1$, then $|B_\alpha - B'_\alpha| \le \frac{2\theta^2}{16}$.*

Proof: From Corollary 3.3 it follows that $|f_\alpha(x_i)| \le 1$. From the definition of the A_i's it follows that $|A_i| \le 1$. Therefore,
$$|B_\alpha - B'_\alpha| = |\frac{1}{m_1}\sum_{i=1}^{m_1}(f_\alpha^2(x_i) - A_i^2)| \le \frac{1}{m_1}\sum_{i=1}^{m_1}|f_\alpha(x_i) - A_i|\cdot|f_\alpha(x_i) + A_i| \le \frac{1}{m_1}\sum_{i=1}^{m_1}\frac{\theta^2}{16}\cdot 2 = \frac{2\theta^2}{16}$$
□

Using the above lemmas, we now fix the values of m_1 and m_2 so that B_α will be a "good" approximation for $E[f_\alpha^2]$.

Lemma 3.9 *Let $m_1 = \Theta(\frac{1}{\theta^2}\log\frac{n}{\delta\theta^2})$ and $m_2 = \Theta(\frac{1}{\theta^4}\log\frac{n\cdot m_1}{\delta\theta^2})$. With probability $1-\delta$ the procedure* `Coef` *outputs all the coefficients z, such that $|\hat{f}(z)| \geq \theta$, and does not output any coefficient z, such that $|\hat{f}(z)| \leq \theta/2$,*

Proof: As shown above we have $O(n/\theta^2)$ calls to the subroutine `Coef` (and the same number of calls to the subroutine `Approx`). To guarantee a total error probability of δ we choose m_1 and m_2 so that the probability of error in each of the calls is no more than $\frac{\delta\theta^2}{n}$. Also recall that if $z = \alpha\beta$, for some $\alpha \in \{0,1\}^k$ and $\beta \in \{0,1\}^{n-k}$, and if $|\hat{f}(z)| \geq \theta$, then $E[f_\alpha^2] \geq \theta^2$.

Consider an α such that $E[f_\alpha^2] \geq \theta^2$. By Lemma 3.6, with probability at least $1 - 2e^{-m_1\theta^2/48}$, the value of $B'_\alpha \geq \frac{3}{4}\theta^2$. By Lemma 3.7, with probability $1 - 2m_1 e^{-\theta^4 m_2/2^9}$, all the values of A_i ($1 \leq i \leq m_1$) satisfy $|f_\alpha(x_i) - A_i| \leq \theta^2/16$. In this case, using Lemma 3.8, $B_\alpha \geq \frac{10}{16}\theta^2 > \frac{\theta^2}{2}$.

Consider an α such that $E[f_\alpha^2] \leq \theta^2/4$. Note that B_α is monotonic in $E[f_\alpha^2]$ and therefore it is enough to consider the case $E[f_\alpha^2] = \theta^2/4$. By Lemma 3.6, with probability $1 - 2e^{-m_1\theta^2/192}$, the value of $B'_\alpha \leq \frac{5}{16}\theta^2$. By Lemma 3.7, with probability $1 - 2m_1 e^{-\theta^4 m_2/2^9}$, all the values of A_i ($1 \leq i \leq m_1$) satisfy $|f_\alpha(x_i) - A_i| \leq \theta^2/16$. In this case, using Lemma 3.8, $B_\alpha \leq \frac{7}{16}\theta^2 < \frac{\theta^2}{2}$.

So far we have shown that the algorithm performs the "right" recursive calls. This implies that, with probability $1-\delta$, the number of recursive calls is at most $4n/\theta^2$. It also implies that all the required coefficients will be output. Now we need to show that in such a case no coefficient z, such that $|\hat{f}(z)| \leq \theta/2$ is output. Note that the probability of outputting such a coefficient is at least the probability that we made one "wrong" recursive call, and this probability is bounded by δ. □

Once the procedure outputs the list of vectors, $z_1, \ldots, z_\ell$, we can approximate each coefficient $\hat{f}(z_i)$. Let the approximate value be γ_i. (Since by definition, $\hat{f}(z_i) = E[f\chi_{z_i}]$ then Lemma 3.5 guarantees that a "small" sample will give with a high probability a "good" approximation for all these coefficients.) The prediction hypothesis is $h(x) = \sum_{i=1}^{\ell}\gamma_i\chi_{z_i}(x)$. To conclude, the algorithm has the following performance:

Theorem 3.10 *There is a randomized algorithm, that for any boolean function f, any $\delta > 0$, and any $\theta > 0$ outputs a list of vectors $\alpha_i \in \{0,1\}^n$ such that*

- *with probability $\geq 1-\delta$ the list contains every vector α for which $|\hat{f}(\alpha)| \geq \theta$ and does not contain any vector α for which $|\hat{f}(\alpha)| \leq \theta/2$. (This implies that the list may contain at most $4/\theta^2$ vectors.)*
- *the algorithm runs in time polynomial in n, $1/\theta$ and $\log 1/\delta$.*

To summarize, in this section we have shown that if f can be ε-approximated by a t-sparse function, then it is sufficient to find all its coefficients larger than $\varepsilon/t = \theta$. Therefore we have established the following theorem.

Theorem 3.11 *Let f be a boolean function such that there exists a t-sparse function g that ε-approximates f (in norm L_2). Then there exists a randomized algorithm, that on input f and $\delta > 0$ outputs a function h, such that with probability $1-\delta$ the function h $O(\varepsilon)$-approximates, in norm L_2, the input function f. The algorithm runs in time polynomial in n, t, $1/\varepsilon$ and $\log 1/\delta$.*

4 Functions with small L_1 norm

In this section we show that a function whose sum of the absolute value of the coefficients is "small" has a "small" number of "large" coefficients that "almost" determine the function. Therefore, in order to get a good approximation of the function, it is sufficient to approximate those coefficients. Saying it differently, we show that functions with "small" L_1 norm can be approximated by sparse functions.

Let f be a boolean function, and recall that $L_1(f) \triangleq \sum_z |\hat{f}(z)|$. The following lemma shows that it is sufficient to approximate a small number of the (large) coefficients of f.

Lemma 4.1 *Let $\varepsilon > 0$. Let $S = \{z \; : \; |\hat{f}(z)| \geq \varepsilon / L_1(f)\}$, and let $g(x) = \sum_{z \in S} \hat{f}(z)\chi_z(x)$. Then*

$$E[(f-g)^2] \leq \varepsilon$$

Proof: By the definition of g, we have

$$(f-g)(x) = \sum_z (\hat{f}(z) - \hat{g}(z))\chi_z(x) = \sum_{z \notin S} \hat{f}(z)\chi_z(x).$$

Therefore, using Parseval's identity, we have

$$E[(f-g)^2] = \sum_{z \notin S} \hat{f}^2(z).$$

This is clearly bounded above by

$$\left(\max_{z \notin S} |\hat{f}(z)|\right) \cdot \left(\sum_{z \in \{0,1\}^n} |\hat{f}(z)|\right) \quad \leq \quad \frac{\varepsilon}{L_1(f)} \cdot L_1(f) = \varepsilon.$$

□

This implies that if we can find all the coefficients that are greater, in absolute value, than $\varepsilon / L_1(f)$, we can approximate f. The procedure in the previous section gives a way to find all such coefficients in time $poly(n, L_1(f), 1/\varepsilon, \log 1/\delta)$. Note that in order to use subroutine `Coef` we need to know the value of $L_1(f)$. If this is not the case we can search for an upper bound on it. This will add a multiplicative factor of $O(\log L_1(f))$ to the time complexity. We have established the following theorem.

Theorem 4.2 *There is a randomized algorithm, that for any boolean function f, and $\varepsilon, \delta > 0$, outputs a function g such that $Prob[E[(f-g)^2] \leq \varepsilon] \geq 1 - \delta$, and the algorithm runs in time polynomial in n, $L_1(f)$, $1/\varepsilon$ and $\log 1/\delta$.*

4.1 Derandomization

For functions with a "small" L_1 norm we can efficiently derandomize the algorithm. One drawback of the derandomization is that it requires that we have a bound on the L_1-norm, since we cannot test hypotheses using randomization as before. The main idea in the derandomization is the usage of λ-bias distributions. The notion of a λ-bias distribution was first suggested by [NN90], and other constructions were given later by [AGHP90]. One way to formalize the notion of λ-bias is the following.

Definition 4.1 *Every distribution μ over $\{0,1\}^n$ can be considered as a real function $\mu(x) = \sum_z \hat{\mu}(z)\chi_z(x)$. A distribution $\mu(x)$ is λ-bias if for any $z \neq \vec{0}$, $|\hat{\mu}(z)| \leq \lambda 2^{-n}$.*

Note that the uniform distribution $u(x) = \frac{1}{2^n}$ has $\hat{u}(z) = 0$, for $z \neq \vec{0}$, and therefore it is 0-bias. Also for any distribution μ,

$$\hat{\mu}(\vec{0}) =< \mu, \chi_0 >= E[\mu] = \frac{1}{2^n}\sum_x \mu(x) = \frac{1}{2^n}.$$

One of the applications of λ-bias distributions is to derandomize algorithms. The derandomization of an algorithm is done by showing that the output of the algorithm when its coin tosses are chosen from a uniform distribution, and the output of the algorithm when its coin tosses are chosen from a λ-bias distribution are very similar. If this holds, then the deterministic algorithm is the following: (1) enumerate all the strings in the λ-bias distribution, (2) for each such string compute the value of the randomized algorithm, and (3) output the average of the replies in step (2). To have an *efficient* derandomization we would like that the sample space of the λ-bias probability distribution would be enumerable "efficiently" (in particular, it has to be "small").

Theorem 4.3 ([NN90, AGHP90]) *There are λ-bias distributions whose sample spaces are of size $(\frac{n}{\lambda})^2$ and are constructible in polynomial time.*

Using the definition (and basic properties) of the Fourier transform we show the following identity.

Lemma 4.4 *For any function f and any distribution μ,*

$$E_\mu[f] = \hat{f}(\vec{0}) + 2^n \sum_{z \neq \vec{0}} \hat{\mu}(z)\hat{f}(z).$$

Proof: By the definitions,

$$E_\mu[f] = \sum_x \mu(x)f(x) = \sum_x \left(\sum_z \hat{\mu}(z)\chi_z(x) \sum_{z'} \hat{f}(z')\chi_{z'}(x)\right).$$

Clearly, if $z = z'$ then $\sum_x \chi_z(x)\chi_{z'}(x) = 2^n$, and if $z \neq z'$ then $\sum_x \chi_z(x)\chi_{z'}(x) = \sum_x \chi_{z\oplus z'}(x) = 0$. Therefore the above sum equals

$$2^n \sum_z \hat{\mu}(z)\hat{f}(z).$$

As $\hat{\mu}(0) = \frac{1}{2^n}$, the lemma follows. □

Our goal now is to show that the algorithm behaves "similarly" when its coin tosses are chosen from the uniform distribution, u, or from a λ-bias distribution, μ. We show it by proving that the A_i's and the B_α computed by Subroutine `Approx` are "similar". The main tool for this is the following lemma.

Lemma 4.5 *Let f be any function, u be the uniform distribution, and μ be a λ-bias distribution, then*

$$|E_\mu[f] - E_u[f]| \leq \lambda L_1(f).$$

Proof: By definition, $E_u[f] = \hat{f}(\vec{0})$. From Lemma 4.4 we have

$$E_\mu[f] = \hat{f}(\vec{0}) + 2^n \sum_{z\neq\vec{0}} \hat{\mu}(z)\hat{f}(z).$$

The definition of λ-bias distributions ensures that $|\hat{\mu}(z)| \leq \lambda/2^n$, therefore we get

$$|E_\mu[f] - E_u[f]| \leq |\hat{f}(\vec{0}) + 2^n \sum_{z\neq\vec{0}} \hat{\mu}(z)\hat{f}(z) - \hat{f}(\vec{0})| \leq 2^n \sum_{z\neq\vec{0}} \frac{\lambda}{2^n}|\hat{f}(z)| = \lambda L_1(f),$$

which completes the proof. □

Lemma 4.6 *Let $h(y) = f(yx)$, for some fixed $x \in \{0,1\}^k$. Then, $L_1(h) \leq L_1(f)$.*

Proof: The proof is by induction k. Let $k = 1$. Then $h(y) = f(yb)$, where $b \in \{0,1\}$. One can verify easily that if $b = 0$, then $\hat{h}(z) = \hat{f}(z0) + \hat{f}(z1)$, and if $b = 1$, then $\hat{h}(z) = \hat{f}(z0) - \hat{f}(z1)$. In both cases $L_1(h) \leq L_1(f)$.

The induction step follows from the fact that we can restrict the function bit after bit. □

This implies that we can compute the A_i's (the inner loop of subroutine `Approx`) with λ-bias distributions.

Lemma 4.7 *Let u be the uniform distribution, and μ be a λ-bias distribution on $\{0,1\}^k$. For any function f, $k \leq n$, $\alpha \in \{0,1\}^k$, and $x \in \{0,1\}^{n-k}$,*

$$|E_{y\in\mu}[f(yx)\chi_\alpha(y)] - E_{y\in u}[f(yx)\chi_\alpha(y)]| \leq \lambda L_1(f).$$

Proof: Let $h_x(y) \triangleq f(yx)$ and $g_x(y) \triangleq h_x(y)\chi_\alpha(y)$. By Lemma 4.6 $L_1(h_x) \leq L_1(f)$. First, we show that $L_1(g_x) = L_1(h_x)$ by showing that they have the same set of coefficients:

$$\hat{g_x}(z) =< g_x, \chi_z >=< h_x\chi_\alpha, \chi_z >=< h_x, \chi_\alpha\chi_z >=< h_x, \chi_{\alpha\oplus z} >= \hat{h_x}(\alpha \oplus z).$$

This implies that $L_1(g_x) \leq L_1(f)$. By Lemma 4.5,

$$|E_\mu[g_x] - E_u[g_x]| \leq \lambda L_1(g_x) \leq \lambda L_1(f).$$

$\Box$

We now show a few basic relations about the L_1 norm of the coefficients of a function, so that we can show that $L_1(f_\alpha^2) \leq L_1^2(f)$.

Claim 4.8 *For any function f and $\alpha \in \{0,1\}^k$, $k \leq n$, then $L_1(f_\alpha) \leq L_1(f)$.*

This is because f_α, by definition, includes only a subset of the coefficients of f. The second claim establishes a relation between the L_1 norm of two functions and the L_1 norm of their product.

Claim 4.9 *For any functions g and h, $L_1(gh) \leq L_1(g)L_1(h)$.*

Proof: Note that

$$g(x)h(x) = \sum_{z_1,z_2\in\{0,1\}^n} \hat{g}(z_1)\hat{h}(z_2)\chi_{z_1}(x)\chi_{z_2}(x) = \sum_{z_3}\left(\sum_{z_1}\hat{g}(z_1)\hat{h}(z_3\oplus z_1)\right)\chi_{z_3}(x).$$

(To see the last transformation take z_3 to be $z_1 \oplus z_2$.) Therefore,

$$L_1(gh) = \sum_{z_3}|\sum_{z_1}\hat{g}(z_1)\hat{h}(z_3\oplus z_1)| \leq \sum_{z_1,z_2}|\hat{g}(z_1)||\hat{h}(z_2)| = L_1(g)L_1(h).$$

$\Box$

We use the above two claims to bound $L_1(f_\alpha^2)$.

Claim 4.10 *For any function f and $\alpha \in \{0,1\}^k$, $k \leq n$, then,*

$$L_1(f_\alpha^2) \leq L_1^2(f_\alpha) \leq L_1^2(f).$$

This implies that we can compute B_α (the outer loop of subroutine `Approx`) using only λ-bias distributions.

Lemma 4.11 *For any function f, and $\alpha \in \{0,1\}^k$, $k \leq n$,*

$$|E_\mu[f_\alpha^2] - E_u[f_\alpha^2]| \leq \lambda L_1^2(f).$$

Proof: Combine Lemma 4.5 with Claim 4.10. $\Box$

Lemma 4.11 can be used to derandomize the outer loop, by choosing $\lambda = \varepsilon/L_1^2(f)$. Lemma 4.7 can be used to derandomize the inner loop, by choosing $\lambda = \varepsilon/L_1(f)$. This implies that we have established the following theorem.

Theorem 4.12 *There is a deterministic algorithm, that receives as an input a boolean function f, $L_1(f)$ and $\varepsilon > 0$, and outputs a function g such that $E[(f-g)^2] \leq \varepsilon$, and the algorithm runs in time polynomial in n, $L_1(f)$, and $1/\varepsilon$.*

5 Decision trees

We consider decision trees, whose input is n boolean variables, and the branching in each node is based on a linear combination (over $GF(2)$) of a subset of the variables (as described in section 2.2). In this section we show that for any function corresponding to such a boolean decision tree, the sum of the absolute values of its coefficients (i.e., the L_1 norm) is bounded by m, the number of nodes in the tree. This implies, using the result in the previous section, that such decision trees can be approximated in polynomial time.

Lemma 5.1 *Let f be computed by a decision tree with m nodes, then $L_1(f) \leq m$.*

Proof: A non-redundant decision tree is a tree in which for every leaf there is some input that ends up in that leaf. By the claim of the lemma, f has a decision tree with m nodes, therefore there is a non-redundant decision tree T_f, with at most m nodes, that computes f. Denote by $\texttt{leaf}(T_f)$ the set of leaves in the tree T_f, and by $d(v)$ the depth of node v. That is, $d(v)$ is the number of nodes on the path from the root to node v, not including v.

We claim that every node at depth d has exactly a 2^{-d} fraction of the inputs reaching it. The inputs that reach a node at depth d pass through d internal nodes; therefore, they satisfy a set of d linear constraints over $GF(2)$. Each such linear constraint is satisfied by exactly $\frac{1}{2}$ of the inputs. Since T_f is non-redundant, the linear constraints are linearly independent. Therefore, the fraction of the inputs that satisfy all the d constraints is 2^{-d}.

By the definition of a decision tree each input reaches a unique leaf. Let $I(v)$ be the set of all the inputs that reach leaf v. Then for every z,

$$\hat{f}(z) = < f, \chi_z > = E[f\chi_z] \quad = \sum_{v \in \texttt{leaf}(T_f)} 2^{-d(v)} val(v) E_{x \in I(v)}[\chi_z(x)].$$

In the following we show that $|E_{x \in I(v)}[\chi_z(x)]| = 1$ for exactly $2^{d(v)}$ values of z, and zero for the rest. This implies that each leaf can contribute at most one to the value of $L_1(f)$.

Consider a leaf v. Any input $x \in I(v)$ satisfies $d(v)$ linear constraints, i.e.

$$\begin{aligned} x \odot y_1 &= b_1 \\ x \odot y_2 &= b_2 \\ &\vdots \\ x \odot y_{d(v)} &= b_{d(v)}, \end{aligned}$$

where $\odot$ denotes the inner product of two n-bit vectors $x, y \in \{0,1\}^n$, i.e. $x \odot y = \sum_i x_i y_i \bmod 2$.

The argument has two parts, depending on whether or not z is a linear combination of the y_i's. If z is a linear combination of the y_i's then clearly the value of $x \odot z$ is fixed, for every $x \in I(v)$. Since the value of $x \odot z$ is fixed, by definition, the value of $\chi_z(x)$ is fixed to either $+1$ or -1, hence $|E_{x \in I(v)}[\chi_z(x)]| = 1$. Since the tree is non-redundant there are exactly $2^{d(v)}$

vectors that are a linear combination of the y_i's. (Note that we consider $z = \vec{0}$ as a linear combination of the y_i's.) On the other hand, if $z \neq \vec{0}$ is not a linear combination of the y_i's then the number of $x \in I(v)$ satisfying $x \odot z = 0$ is the same as the number of $x \in I(v)$ satisfying $x \odot z = 1$. Therefore, in this case $E_{x\in I(v)}[\chi_z(x)] = 0$. Combining the two claims, we have that $\sum_z |E_{x\in I(v)}[\chi_z]| = 2^{d(v)}$.

Intuitively, each leaf v contributes to at most $2^{d(v)}$ coefficients, and to each coefficient it contributes $2^{-d(v)}$. This implies that leaf v contributes at most one to the sum of the absolute value of all the coefficients. Therefore, $L_1(f)$ is bounded by the number of leaves which is at most m. The following calculations shows this formally:

$$\begin{aligned}
L_1(f) = \sum_{z\in\{0,1\}^n} \left|\hat{f}(z)\right| = \sum_{z\in\{0,1\}^n} |E[f\chi_z]| &= \sum_{z\in\{0,1\}^n} \left| \sum_{v\in\texttt{leaf}(T_f)} 2^{-d(v)} val(v) E_{x\in I(v)}[\chi_z] \right| \\
&\leq \sum_{z\in\{0,1\}^n} \sum_{v\in\texttt{leaf}(T_f)} 2^{-d(v)} \left|E_{x\in I(v)}[\chi_z]\right| \\
&= \sum_{v\in\texttt{leaf}(T_f)} 2^{-d(v)} \sum_{z\in\{0,1\}^n} \left|E_{x\in I(v)}[\chi_z]\right| \\
&= \sum_{v\in\texttt{leaf}(T_f)} 2^{-d(v)} 2^{d(v)} \\
&= |\texttt{leaf}(T_f)| \leq m.
\end{aligned}$$

□

Combining Lemma 5.1 and Claim 2.1 with Theorem 4.12, we get the following theorem:

Theorem 5.2 *There is a polynomial time (deterministic) algorithm, that for any boolean function f that can be represented by an m node decision tree with linear operations, and for any $\varepsilon > 0$, outputs a function g such that*

$$Prob[f \neq sign(g)] \leq \varepsilon,$$

and the algorithm runs in time polynomial in n, m, and $1/\varepsilon$.

We now show that the bound given in Lemma 5.1 is tight. Consider the inner product function on inputs of $n = 2\ell$ variables:

$$f(x_1, \ldots, x_{2\ell}) = (-1)^{x_1 x_{\ell+1} \oplus \cdots \oplus x_\ell x_{2\ell}} = \prod_{i=1}^{\ell} (-1)^{x_i x_{\ell+i}}.$$

Let $h_i(x_1, \ldots, x_{2\ell}) = (-1)^{x_i x_{\ell+i}}$, for $1 \leq i \leq \ell$. Clearly, $f = \prod_{i=1}^{\ell} h_i$. The Fourier transform of h_i is

$$h_i(x) = \frac{1}{2} + \frac{1}{2}\chi_{I(i)}(x) + \frac{1}{2}\chi_{I(i+\ell)}(x) - \frac{1}{2}\chi_{I(i,i+\ell)}(x),$$

where $I(S)$ is the indicator vector of the set S, e.g. $I(\{j\})$ is equal to the vector who has the jth coordinate one and all the other coordinates zero. From the expansion of h_i it is clear that $|\hat{f}(z)| = 2^{-\ell}$, for any z, and therefore $L_1(f) = 2^{2\ell}2^{-\ell} = 2^{\ell}$. We will show that there is a decision tree with linear operations of size $O(2^{\ell})$ that computes f.

The following is a description of the decision tree that computes f. The first ℓ levels of the decision tree form a complete binary tree. In each node of level i $(1 \leq i \leq \ell)$ we test $x_i \oplus x_{\ell+i}$. For every leaf v of the tree, let $b_1^v, \ldots, b_\ell^v$ be the sequence of the replies to the queries $x_i \oplus x_{\ell+i}$, along the path from the root of the tree to v. Let, $S_v = \{i | b_i^v = 0\}$. We now test for the parity of all x_i's with $i \in S_v$. Let the value of the computation be the value of the parity. The tree has only depth $\ell + 2$, and hence only $O(2^{\ell})$ nodes. The reason that it computes the inner product correctly is the following. If $b_i^v = 1$, then exactly one of $x_i, x_{\ell+i}$ is 0 and in particular $x_i x_{\ell+i} = 0$. This implies that the i-th term in the inner product is zero, and therefore we can ignore it. If $b_i^v = 0$, then either both $x_i, x_{\ell+i}$ are 0 or both $x_i, x_{\ell+i}$ are 1. In both cases, $x_i x_{\ell+i} = x_i$. Therefore, instead of considering the value of the i-th term (i.e., $x_i x_{\ell+i}$), we can consider the variable x_i. Therefore the parity of S_v is the parity of all the relevant terms.

Exact Reconstruction

We show that a boolean decision trees with linear operations can be recovered exactly in time polynomial in n and 2^d, where d is the depth of the tree.

It follows from the proof of Lemma 5.1 that all the coefficients of a tree of depth d, can be written as $k/2^d$, where k is an *integer* in the range $[-2^d, +2^d]$. The idea is to first find a good approximation of *all* the non-zero coefficients and then, using the above fact, to compute them *exactly.*

By Theorem 4.12, we have a *deterministic* algorithm that for every function f and $\varepsilon > 0$ outputs a function g such that

$$\sum_z (\hat{f}(z) - \hat{g}(z))^2 = E[(f-g)^2] \leq \varepsilon$$

in time polynomial in $n, L_1(f)$ and $1/\varepsilon$. In particular, it follows that $|\hat{f}(z) - \hat{g}(z)| \leq \sqrt{\varepsilon}$, for every z. We use this algorithm, with $\varepsilon < (\frac{1}{2}\frac{1}{2^d})^2$, which ensures that g satisfies $|\hat{f}(z) - \hat{g}(z)| < \frac{1}{2}\frac{1}{2^d}$, for every z. Since the real coefficient is of the form $k/2^d$, where k is *integer*, the difference between possible values that a coefficient can have is $1/2^d$; since the error is smaller than $\frac{1}{2}\frac{1}{2^d}$, by rounding we find the exact coefficient.

This implies that we recovered all the Fourier coefficients of the function *exactly.* Therefore, we found a function whose Fourier transform is *identical* to the tree's Fourier transform, this implies that the two functions are identical. By the choice of ε and as $L_1(f) \leq m \leq 2^{d+1}$, the running time of the algorithm is polynomial in n and 2^d. Thus, we have established the following theorem,

Theorem 5.3 *There is a (deterministic) polynomial time algorithm, that for any boolean function f that can be represented by a depth d decision tree with linear operations, outputs a function* *g such that*

$$\forall x \in Z_2^n \quad g(x) = f(x)$$

and the algorithm runs in time polynomial in n, and 2^d.

An interesting special case is when the depth of the tree is logarithmic in n. In such a case, the algorithm will run in *polynomial* time.

6 Extensions and Open Problems

The characterization of the decision trees can be extended easily to boolean functions of the form $f : \{0,1,\ldots,k-1\}^n \to \{0,1\}$ that can be computed by a polynomial-size k-ary decision tree. Namely, a tree in which each inner node v has k outgoing edges. When the computation arrives the node v, labeled by $S_v \in \{0,1,\ldots,k-1\}^n$, it assigns this node the value $\sum_{i=1}^n S_i \cdot x_i \bmod k$, and the computation continues to the appropriate child of v. For extending the results to such functions and decision trees we have to define the appropriate characters and modify the proofs accordingly. For each $z \in \{0,1,\ldots,k-1\}^n$, define the basis function χ_z:

$$\chi_z(x_1,\cdots,x_n) \stackrel{\triangle}{=} w^{z_1\cdot x_1+\ldots+z_n\cdot x_n},$$

where $w = e^{\frac{2\pi i}{k}}$ is the root of unity of order k. In this case, a straightforward extension of our proof for $k = 2$ shows that the sum of the magnitudes of the coefficients is bounded by the number of leaves.

Another issue is decision trees with real outputs, where the leaves have real values from the interval $[0, M]$, i.e. $f : \{0,1\}^n \to [0, M]$. In a similar way to the boolean case, one can show that any function f that has a real decision tree with m leaves then $L_1(f) \le mM$. In this case the running time of the learning algorithm would be polynomial in M.

An open problem related to this work is to find other classes of functions that can be learned in polynomial-time. In particular, it is very interesting whether functions that can be represented by a polynomial-size DNF formula can be learned in polynomial-time. One possible direction to resolve this open problem is to show that for any polynomial size DNF there is a polynomially sparse function that approximates it in L_2. So far we have not found any counter examples to this claim.

While our algorithm can be derandomized in the case of functions with polynomial L_1-norm, it is an open problem to derandomize it in the more general case of functions that can be approximated by polynomially sparse functions.

References

[ABFR91] James Aspnes, Richard Beigel, Merrick Furst, and Steven Rudich. The expressive power of voting polynomials. In *Proceedings of the 23^rd Annual ACM Symposium on Theory of Computing*, pages 402–409, May 1991.

[AGHP90] Noga Alon, Oded Goldreich, Johan Hastad, and Rene Peralta. Simple constructions of almost k-wisw independent random variables. In 31st *Annual Symposium on Foundations of Computer Science, St. Louis, Missouri*, pages 544–553, October 1990.

[Ajt83] M. Ajtai. $\sum_1^1$-formulae on finite structure. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[AM91] William Aiello and Milena Mihail. Learning the fourier spectrum of probabilistic lists and trees. In *Proceedings SODA 91*. ACM, Jan 1991.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

[Bel92] Mihir Bellare. A technique for upper bounding the spectral norm with applications to learning. In 5th *Annual Workshop on Computational Learning Theory*, pages 62–70, July 1992.

[BHO90] Y. Brandman, J. Hennessy, and A. Orlitsky. A spectral lower bound technique for the size of decision trees and two level circuits. *IEEE Trans. on Computers.*, 39(2):282–287, 1990.

[Bru90] J. Bruck. Harmonic analysis of polynomial threshold functions. *Siam J. on Disc. Math.*, 3(2):168–177, May 1990.

[BS90] J. Bruck and R. Smolensky. Polynomial threshold functions, AC^0 functions and spectral norms. In 31st *Annual Symposium on Foundations of Computer Science, St. Louis*, pages 632–641, October 1990.

[Dia88] Persi Diaconis. *The use of Group representation in probability and statistics.* 1988.

[EH89] Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, September 1989.

[FJS91] Merrick L. Furst, Jeffrey C. Jackson, and Sean W. Smith. Improved learning of ac^0 functions. In 4th *Annual Workshop on Computational Learning Theory*, pages 317–325, August 1991.

[FSS84] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984.

[GL89] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proc.* 21*st ACM Symposium on Theory of Computing*, pages 25–32. ACM, 1989.

[Han90] Thomas Hancock. Identifying μ-decision trees with queries. In *3rd COLT*, pages 23–37, August 1990.

[Han91] Thomas Hancock. Learning 2μ DNF and $k\mu$ decision trees. In *4th COLT*, pages 199–209, August 1991.

[Has86] J. Hastad. *Computational limitations for small depth circuits.* MIT Press, 1986. Ph.D. thesis.

[HR89] Torben Hagerup and Christine Rub. A guided tour to chernoff bounds. *Info. Proc. Lett.*, 33:305–308, 1989.

[LMN89] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, fourier transform and learnability. In 30^{th} *Annual Symposium on Foundations of Computer Science, Reseach Triangle Park, NC*, pages 574–579, October 1989.

[Man92] Yishay Mansour. An $o(n^{\log\log n})$ learning algorihm for DNF under the uniform distribution. In 5^{th} *Annual Workshop on Computational Learning Theory*, pages 53–61, July 1992.

[NN90] Joseph Naor and Moni Naor. Small bias probability spaces: efficient construction and applications. In *Proceedings of the* 22^{nd} *Annual ACM Symposium on Theory of Computing, Baltimore, Maryland*, pages 213–223, May 1990.

[RB91] M. Ron Roth and Gyora M. Benedek. Interpolation and approximation of sparse multivariate plynomials over GF(2). *Siam J. on Comp.*, 20(2):291–314, April 1991.

[SB91] Kai-Yeung Sui and Jehoshua Bruck. On the power of threshold circuits with small weights. *Siam J. on Disc. Math.*, 4(3):423–435, Aug 1991.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[Yao85] A. C. Yao. Separating the polynomial-time hierarchy by oracles. In 26^{th} *Annual Symposium on Foundations of Computer Science, Portland, Oregon*, pages 1–10, October 1985.