6.5420 Randomness and Computation Apr 21, 2026

Lecture 18

Lecturer: Ronitt Rubinfeld *Scribe: Sijin Peng*

1 Fourier Analysis Recap

Define $\chi_S(x) = \prod_{i \in S} x_i$ for $x \in \{-1,1\}^n$ and $S \subseteq [n]$. Further, define the inner product $\langle f, g \rangle = \frac{1}{2^n} \sum_{x \in \{-1,1\}^n} f(x)g(x)$ for $f, g : \{-1,1\}^n \to \mathbb{R}$. We can always write a function f over the Boolean cube $\{-1,1\}^n$ as a linear combination of χ_S for all $S \subseteq [n]$. Specifically,

$$f(x) = \sum_{S \subseteq [n]} \hat{f}(S)\chi_S(x), \tag{1}$$

where $\hat{f}(S) = \langle f, \chi_S \rangle$ and $\hat{f}(S) = 1 - 2\Pr_{x \sim \{-1,1\}^n}[f(x) \neq \chi_S(x)] = 2\Pr_{x \sim \{-1,1\}^n}[f(x) = \chi_S(x)] - 1$ when f is a Boolean function.

Plancherel's Identity:

$$\langle f, g \rangle = \sum_{S \subseteq [n]} \hat{f}(S)\hat{g}(S). \tag{2}$$

Parseval's Theorem for Boolean Functions: For $f : \{\pm 1\}^n \to \{\pm 1\}$

$$1 = \langle f, f \rangle = \sum_{S \subseteq [n]} \hat{f}(S)^2. \tag{3}$$

Lemma 1. *For a Boolean function f, there is an algorithm that approximates $\hat{f}(S)$ for a given $S \subseteq [n]$ to within an ϵ additive error with probability $1 - \delta$ using $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ uniform samples.*

2 Learning Heavy Fourier Coefficients (with Queries)

The algorithm we will discuss in this section originates from a 1991 paper by Kushilevitz and Mansour [2]. The main idea is similar to the famous Goldreich-Levin theorem [1] in cryptography.

2.1 The Setup

Given a Boolean function f, which is not necessarily Fourier-concentrated, we want to identify its heavy Fourier coefficients. Specifically, given a parameter θ, the goal of the algorithm in this section is to output a family $\mathcal{F}$ of subsets of $[n]$, such that:

- Not Missing Out: If $\hat{f}(S) \geq \theta$, then $S \in \mathcal{F}$;
- No Junk: If $S \in \mathcal{F}$, then $\hat{f}(S) \geq \frac{\theta}{2}$.

It is currently unknown how to approach this efficiently using only uniform samples. However, what if the algorithm can query f at any input?

A preliminary observation is that if $\hat{f}(S) \geq \frac{3}{4} + \epsilon$, then we can use the self-correcting procedure discussed in previous lectures to extract S. Can we do better?

2.2 The Approach: Exhaustive Search With Good Pruning

The high-level idea of the algorithm is straightforward: exhaust all possible Fourier coefficients using n decision steps by deciding on the inclusion of x_i at level i. This search process naturally generates a full binary tree of depth $n+1$, where each leaf corresponds to a Fourier coefficient of f. An example search tree is given in the following figure.

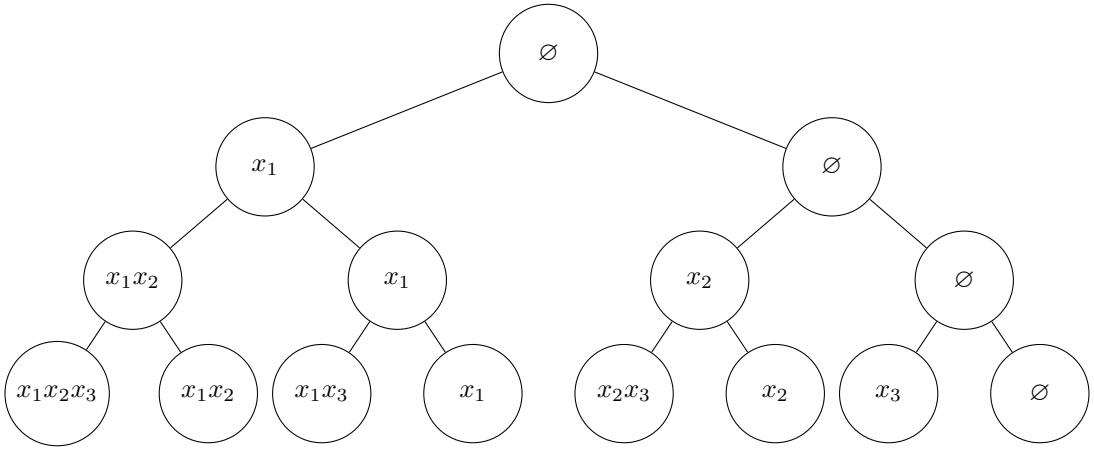

Figure 1: A search tree of depth 3

A naive exhaustive search takes $\Omega(2^n)$ time, which is prohibitively slow. However, if each vertex could estimate whether a heavy leaf exists within its subtree, we could prune useless branches and achieve a significantly better running time.

Next, we introduce the quantity used for pruning: fix the current search level $k \in [0, n]$ and the current search node $S_1 \subseteq [k]$. We define a function f_{k,S_1}, which maps from $\{-1,1\}^{n-k}$ to $\mathbb{R}$, such that

$$f_{k,S_1}(x) = \sum_{T_2 \subseteq [k+1,n]} \hat{f}(S_1 \cup T_2)\chi_{T_2}(x).^1 \tag{4}$$

Put another way, for a level k, each of the 2^k nodes receives a different function f_{k,S_1} which, roughly speaking, is the "projection" of f onto the first k coordinates, and does not depend on the first k coordinates at all.

Let us perform some sanity checks on this definition:

- When $k = 0$, $S_1 = \varnothing$, and $f_{0,\varnothing}(x) = \sum_{T_2 \subseteq [n]} \hat{f}(T_2)\chi_{T_2}(x) = f(x)$;
- When $k = n$, $[k+1, n] = \varnothing$ and $f_{n,S_1}(x) = \hat{f}(S_1)\chi_{\varnothing}(x) = \hat{f}(S_1)$.

With the definition of f_{k,S_1}, we apply the following pruning strategy: when we are at level k and node S_1, if $\mathbb{E}_x[f^2_{k,S_1}(x)] < \frac{3\theta^2}{4}$, we immediately stop searching that subtree and backtrack. Conversely, if $\mathbb{E}_x[f^2_{k,S_1}(x)] \geq \frac{3\theta^2}{4}$, we proceed deeper and search its children.

2.3 Correctness

To prove the correctness of the pruning, we must address the following three questions:

1. Can we compute $\mathbb{E}_x[f^2_{k,S_1}(x)]$ efficiently?

[1] We will use subscript 1 for prefixes and subscript 2 for suffixes.

2. Can we get to all heavy leaves with this pruning? Do we get junk?

3. How many paths do we take with this pruning? Do we still visit an exponential number of vertices?

2.3.1 Answer to Question 3: Not Too Many Paths!

Lemma 2. *For $f : \{\pm 1\}^n \to \{\pm 1\}$ and an arbitrary parameter $\gamma > 0$,*

- *There are at most $\frac{1}{\gamma^2}$ Fourier coefficients whose absolute value is at least γ;*
- *For all $0 \le k \le n$, there are at most $\frac{1}{\gamma^2}$ functions f_{k,S_1} such that $\mathbb{E}_x[f^2_{k,S_1}(x)] \ge \frac{1}{\gamma^2}$.*

We proved this claim in the previous lecture. The first part of the claim is a direct consequence of the Boolean Parseval's theorem, while the second part is implied by the following equality:

$$\mathbb{E}_x[f_{k,S_1}(x)^2] = \sum_{T_2 \subseteq \{k+1 \dots n\}} \hat{f}(S_1 \cup T_2)^2. \tag{5}$$

Lemma 2 implies that, assuming a perfect estimator of $\mathbb{E}_x[f^2_{k,S_1}(x)]$, all but $O(\frac{1}{\theta^2})$ nodes in the search tree will be pruned at each level. As a consequence, we will visit $O(\frac{n}{\theta^2})$ nodes in total.

2.3.2 Answer to Question 2: Never Miss Out Heavy Leaves!

Lemma 3. *For any $0 \le k \le n$ and $S_1 \subseteq [k]$, if $\exists T_2 \subseteq [k+1, n]$ s.t. $|\hat{f}(S_1 \cup T_2)| \ge \theta$, then $\mathbb{E}_x[f^2_{k,S_1}(x)] \ge \theta^2$.*

Proof. This is a direct corollary of eq. (5). □

Consequently, we can find all heavy leaves without spending too much time when the estimation is perfect. However, we cannot yet be certain that all found leaves are indeed heavy. This can be easily resolved by filtering out small coefficients before generating the final output, using Lemma 1.

2.3.3 Answer to Question 1: Efficient Estimation

An initial approach to estimating $\mathbb{E}_x[f_{k,S_1}(x)^2] = \sum_{T_2 \subseteq \{k+1 \dots n\}} \hat{f}(S_1 \cup T_2)^2$ might be to estimate each $\hat{f}(S_1 \cup T_2)$ individually. This is highly inefficient. Another potential challenge is that heavy $\hat{f}(S_1 \cup T_2)$ values can differ greatly depending on how S_1 is fixed, so nodes at the same level do not share much useful information about f_{k,S_1}. Nevertheless, the following lemma demonstrates that we can efficiently estimate $\mathbb{E}_x[f_{k,S_1}(x)^2]$.

Lemma 4. *For $0 \le k \le n$, $S_1 \subseteq [k]$, and $x \in \{-1, 1\}^{n-k}$,*

$$f_{k,S_1}(x) = \mathbb{E}_{y \in \{\pm 1\}^k}[f(y \| x) \chi_{S_1}(y)], \tag{6}$$

where $y \| x$ is the concatenation of y and x.

Put another way, to estimate $f_{k,S_1}(x)$, we randomly generate the first k bits, say y, and check whether $f(y\|x)$ agrees with $\chi_{S_1}(y)$.

Proof. For a set $T \subseteq [n]$, we can always partition it into $T_1 = T \cap [k]$ and $T_2 = T \cap [k+1, n]$. Then we have

$$\begin{aligned} f(yx) &= \sum_{T \subseteq [n]} \hat{f}(T) \chi_T(yx) \\ &= \sum_{T_1 \subseteq [k]} \sum_{T_2 \subseteq [k+1,n]} \hat{f}(T_1 \cup T_2) \chi_{T_1}(y) \chi_{T_2}(x), \end{aligned}$$

and from the fact that $\mathbb{E}_y[\chi_{S_1}(y)\chi_{T_1}(y)]$ equals zero when $S_1 \neq T_1$ and one when $S_1 = T_1$, we get

$$\begin{aligned}
\mathbb{E}_y[f(yx)\chi_{S_1}(y)] =& \mathbb{E}_y\left[\sum_{T_1\subseteq[k]}\sum_{T_2\subseteq[k+1,n]}\hat{f}(T_1\cup T_2)\chi_{T_1}(y)\chi_{T_2}(x)\chi_{S_1}(y)\right] \\
=& \sum_{T_1\subseteq[k]}\sum_{T_2\subseteq[k+1,n]}\hat{f}(T_1\cup T_2)\chi_{T_2}(x)\mathbb{E}_y[\chi_{S_1}(y)\chi_{T_1}(y)] \\
=& \sum_{T_2\subseteq[k+1,n]}\hat{f}(S_1\cup T_2)\chi_{T_2}(x) \\
=& f_{k,S_1}(x).
\end{aligned}$$

□

As a result, to estimate $\mathbb{E}_x[f_{k,S_1}(x)^2]$, we have

$$\begin{aligned}
\mathbb{E}_x[f_{k,S_1}(x)^2] =& \mathbb{E}_x[\mathbb{E}_y[f(yx)\chi_{S_1}(y)]^2] \\
=& \mathbb{E}_{x,y_1,y_2}[f(y_1x)\chi_{S_1}(y_1)f(y_2x)\chi_{S_1}(y_2)],
\end{aligned}$$

and thus, it suffices for the algorithm to average $f(y_1x)\chi_{S_1}(y_1)f(y_2x)\chi_{S_1}(y_2)$ over uniform random samples of (x, y_1, y_2). According to the Chernoff bound, we can obtain an estimation of $\mathbb{E}_x[f_{k,S_1}(x)^2]$ to within an additive error of $\frac{\theta^2}{4}$ with probability $1-\frac{1}{n^2}$ using $\mathrm{poly}(n, \frac{1}{\theta})$ queries.

2.4 Wrapping Up

The pseudocode for the KM algorithm is given in Algorithm 1.

Algorithm 1 Procedure $\mathrm{KM}(f, \theta, k, S_1)$

if $k = n$ **then**
  Use Lemma 1 to estimate $\hat{f}(S_1)$ with $\delta = \frac{1}{n^2}$, $\epsilon = \frac{\theta}{4}$, and output S_1 when the estimation is at least $\frac{3}{4}\theta$.
else
  Use Lemma 4 to estimate $\mathbb{E}_x[f_{k,S_1}(x)^2]$ with an additive error of $\frac{\theta^2}{4}$ with probability $1-\frac{1}{n^2}$.
  if The estimation is at least $\frac{3\theta^2}{4}$ **then**
    $\mathrm{KM}(f, \theta, k+1, S_1)$
    $\mathrm{KM}(f, \theta, k+1, S_1 \cup \{k+1\})$
  end if
end if

Theorem 5. *With probability $1-O(\frac{1}{n})$, Algorithm 1 outputs a family $\mathcal{F}$ of subsets of $[n]$ using $\mathrm{poly}(n, \frac{1}{\theta})$ queries, such that*

- *Not Missing Out: If $\hat{f}(S) \geq \theta$, then $S \in \mathcal{F}$;*
- *No Junk: If $S \in \mathcal{F}$, then $\hat{f}(S) \geq \frac{\theta}{2}$.*

Proof. The "not missing out" property is guaranteed by Lemma 3, and the "no junk" property is guaranteed by the final check when $k = n$.

Assuming the estimation is always correct, only nodes (k, S_1) such that $\mathbb{E}_x[f_{k,S_1}(x)^2] \geq \frac{3\theta^2}{4} - \frac{\theta^2}{4} = \frac{\theta^2}{2}$ will recurse. According to Lemma 2, we will visit $O(\frac{n}{\theta^2})$ vertices in total when the estimations are all correct, which holds with probability $1-O(\frac{1}{n})$ by the union bound.

Finally, the runtime is bounded because we only visit $O(\frac{n}{\theta^2})$ vertices, and the estimation at each node uses $\mathrm{poly}(n, \frac{1}{\theta})$ queries. □

2.5 Applications of the KM Algorithm

The KM algorithm can be used to learn two types of Boolean functions:

- Functions of small L_1-norm: When $L_1(f) = \sum_S |\hat{f}(S)|$ is small, we can set $\theta = \frac{\epsilon}{L_1(f)}$ and approximate f using all large Fourier coefficients output by the KM algorithm. If we do not know $L_1(f)$, we can use a binary lifting approach to guess the value of $L_1(f)$ (1, 2, 4, 8, ...) and select the best hypothesis among all guesses.
- Decision Trees: It can be shown that a decision tree f with m nodes has $L_1(f) \leq m$. Furthermore, a depth-d decision tree has all of its Fourier coefficients at least 2^{-d}, so we can reconstruct the exact function f using $\text{poly}(n, 2^d)$ samples.

For more details on these applications, one can refer to [2].

3 Next Lecture: Monotone Functions

We define a partial order over the Boolean cube: For $x, y \in \{0,1\}^n$, $x \leq y$ if $x_i \leq y_i$ for all $1 \leq i \leq n$. A function f is *monotone* over the Boolean cube if $x \leq y$ implies $f(x) \leq f(y)$.

In the homework, we will explore a "strong learning" algorithm for monotone Boolean functions, which outputs a function that is ϵ-close to the underlying function over the uniform distribution. However, it requires $2^{\tilde{O}(\sqrt{n})}$ queries, which is far from polynomial. In the next lecture, we will present an algorithm that learns slightly better than random guessing using a polynomial number of samples.

References

[1] Oded Goldreich and Leonid A Levin. A hard-core predicate for all one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 25–32, 1989.

[2] Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the fourier spectrum. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 455–464, 1991.