Computer Science 294 Lecture 8 Notes

Daniel Raban

February 9, 2023

1 Analysis of the Goldreich-Levin Algorithm and Learning Juntas

1.1 Description and analysis of the Goldreich-Levin algorithm

Last time we introduced the Goldreich-Levin algorithm, an efficient randomized algorithm that, given θ and query access to $f : \{\pm 1\}^n \to \{\pm 1\}$, outputs with high probability a list $\mathcal{L}$ such that for all S,

$$|\widehat{f}(S)| \geq \theta \implies S \in \mathcal{L}, \qquad S \in \mathcal{L} \implies |\widehat{f}(S)| \geq \theta/2.$$

This lets us find the Fourier coefficients with most of the Fourier mass.

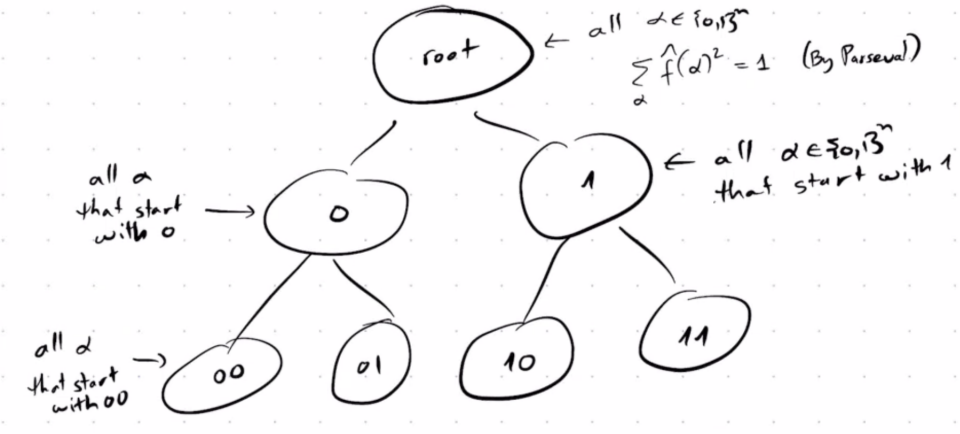
Here is more convenient notation for the proof: We will represent sets as binary vectors. The set $S \subseteq [n]$ corresponds to $\alpha \in \{0,1\}^n$ by

$$\alpha_i = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

for $i = 1, \dots, n$. For example, if $n = 6$ and $S = \{1.3.4\}$, then $\alpha = (1, 0, 1, 1, 0, 0)$. With this notation,

$$f(x) = \sum_{\alpha \in \{0,1\}^n} \widehat{f}(\alpha)\chi_\alpha(x), \qquad \chi_\alpha(x) = \prod_{i:\alpha_i = 1} x_i.$$

Recall that Parseval's identity gives $\sum_{\alpha \in \{0,1\}^n} \widehat{f}(\alpha)^2 = 1$. We want to find all α with $\widehat{f}(\alpha)^2 \geq \theta^2$. How many are there? The proof idea is a divide and conquer strategy. We will start with the root, containing all $\alpha \in \{0,1\}^n$, so $\sum_\alpha \widehat{f}(\alpha)^2 = 1$. Then we partition these coefficients according to their first bit and calculate the sum of the squared coefficients within each bucket. Then we partition these buckets further according to their second bit.



We won't look at the entire binary tree, though. Suppose that for any $k \in \{0, 1, \ldots, n\}$ and any $\beta \in \{0,1\}^k$, we can calculate $\sum_{\alpha:\alpha \text{ starts with } \beta} \widehat{f}(\alpha)^2$. Then, whenever this sum is $\geq \theta^2/2$, we expand node β to its two children $\beta \circ 0$ and $\beta \circ 1$. Otherwise, we prune the tree at this leaf.

First note that every α such that $|\widehat{f}(\alpha)| \geq \theta$ would survive and be one of the leaves. The second observation is that for any layer of this tree, the number of surviving nodes per layer is at most $2/\theta^2$ by Parseval's identity. This tells us that the size of the tree will be $O(n/\theta^2)$, which is linear rather than exponential in n.

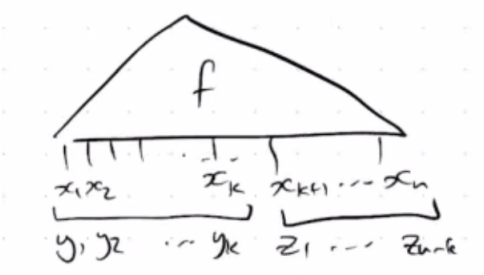
For $\beta \in \{0,1\}^k$, how do we estimate $\sum_{\alpha:\alpha \text{ starts with } \beta} \widehat{f}(\alpha)^2$? It would be nice to have a function $g_\beta : \{\pm 1\}^{n-k} \to \{\pm 1\}$ with

$$g_\beta(z) = \sum_{\gamma \in \{0,1\}^{n-k}} \widehat{f}(\beta \circ \gamma) \chi_\gamma(z)$$

because

$$\mathbb{E}_{Z \sim \{\pm 1\}^{n-k}}[g_\beta(Z)^2] = \sum_{\alpha \in \{0,1\}^{n-k}} \widehat{f}(\beta \circ \gamma)^2.$$

Lets look at f as a function of $x_1, x_2, \ldots, x_k$ and $x_{k+1}, \ldots, x_n$. Let's relabel these as $y_1, y_2, \ldots, y_k$ and $z_1, \ldots, z_{n-k}$.



We can write

$$f(x) = \sum_{\alpha \in \{0,1\}^n} \widehat{f}(\alpha) \prod_{i:\alpha_i=1} x_i,$$

which we now express as

$$\begin{aligned} f(y,z) &= \sum_{\alpha \in \{0,1\}^n} \widehat{f}(\alpha) \prod_{i:\alpha_i=1, i \le k} y_i \prod_{i:\alpha_i=1, i>k} z_{i-k} \\ &= \sum_{\beta' \in \{0,1\}^k, \gamma \in \{0,1\}^{n-k}} \widehat{f}(\beta' \circ \gamma) \chi_{\beta'}(y) \chi_\gamma(z) \end{aligned}$$

Look at $\beta = 0^k$, for example. Then

$$\begin{aligned} g_0(z) &= \mathbb{E}_{Y \sim \{\pm 1\}^k}[f(Y,z)] \\ &= \sum_{\beta' \in \{0,1\}^k, \gamma \in \{0,1\}^{n-k}} \widehat{f}(\beta' \circ \gamma) \underbrace{\mathbb{E}_{Y \sim \{\pm 1\}^k}[\chi_{\beta'}(Y)]}_{=\delta_{\beta'=0^k}} \chi_\gamma(z) \\ &= \sum_{\gamma \in \{0,1\}^{n-k}} \widehat{f}(0^k \circ \gamma) \chi_\gamma(z). \end{aligned}$$

More generally,

$$\begin{aligned} g_\beta(z) &= \mathbb{E}_{Y \sim \{\pm 1\}^k}[f(Y,z)\chi_\beta(y)] \\ &= \sum_{\beta' \in \{0,1\}^k, \gamma \in \{0,1\}^{n-k}} \widehat{f}(\beta' \circ \gamma) \underbrace{\mathbb{E}_{Y \sim \{\pm 1\}^k}[\chi_{\beta'}(Y)\chi_\beta(Y)]}_{=\langle \chi_\beta, \chi_{\beta'} \rangle} \chi_\gamma(z) \\ &= \sum_{\gamma \in \{0,1\}^{n-k}} \widehat{f}(\beta \circ \gamma) \chi_\gamma(z). \end{aligned}$$

Recall that we want to estimate $\mathbb{E}_{Z \sim \{\pm 1\}^{n-k}}[g_\beta(Z)^2] = \sum_{\gamma \in \{0,1\}^{n-k}} \widehat{f}(\beta \circ \gamma)^2$. How do we estimate this when we have query access to f? We have

$$\mathbb{E}_Z[g_\beta(Z)^2] = \mathbb{E}_Z[(\mathbb{E}_Y[f(Y,Z)\chi_\beta(Y)])^2]$$

$$= \mathbb{E}_{Z,Y,Y'}[f(Y,Z)\chi_\beta(Y)f(Y',Z)\chi_\beta(Y')]$$

This is the expected value of a $\{\pm 1\}$-valued random variable, so it can be estimated with probability $1-\delta$ up to accuracy ε using $m = O(\log(1/\delta)/\varepsilon^2)$ queries. Note that we need to be able to query f to make sure we have the values of f at two points which have the same last $n-k$ bits.

Here is the overall Goldreich-Levin algorithm:

1. For $k = 0, 1, \ldots, n$,
   (a) For every alive $\beta \in \{0,1\}^k$, estimate with probability $\geq 1-\delta$ the quantity $\sum_\gamma \widehat{f}(\beta \circ \gamma)^2$ up to accuracy $\varepsilon = \theta^2/4$. If this quantity is smaller than $\theta^2/2$, prune. Otherwise, add $\beta \circ 0$ and $\beta \circ 1$ to the queue.
   (b) If more than $8/\theta^2$ strings are in the queue, abort.
2. Output the queue.

Here is a more formal proof that the algorithm works.

Proof. Let $\{E_i\}_{i=1}^n$ be the events where all estimates in the i-th iteration were ε-accurate. If $E_1, \ldots, E_n$ all happen, then we did not abort, since for every $\beta \in \{0,1\}^k$ in the k-th iteration that was not pruned had

$$\sum_\gamma \widehat{f}(\beta \circ \gamma)^2 \geq \theta^2/2 - \theta^2/4 = \theta^2/4.$$

By Parseval's identity, at most $4/\theta^2$ leaves are not pruned, and each of these leaves has two children. By a union bound,

$$\mathbb{P}(E_i \mid E_1, \ldots, E_{i=1}) \geq 1 - \delta\frac{8}{\theta^2},$$

so

$$\mathbb{P}(E_1, \ldots, E_n) \geq 1 - \delta 8/\theta^2 \cdot n.$$

This has runtime $\text{poly}(n, 1/\theta, \log(1/\delta))$. □

1.2 PAC learning for k-juntas

Let's return to PAC learning, this time for learning k-juntas. Recall that $f : \{\pm 1\}^n \to \{\pm 1\}$ is a **k-junta** if it depends only on k of the coordinates, i.e. there exists a $g : \{\pm 1\}^n \to \{\pm 1\}$ and coordinates $i_1, i_2, \ldots, i_k \in [n]$ such that $f(x) = g(x_{i_1}, x_{i_2}, \ldots, x_{i_k})$ for all x.

Learning juntas in the query model is relatively easy, since we can just flip one coordinate to see if that coordinate is relevant. So we will focus on learning from random labeled examples. The naive algorithm is to check by brute force, which takes roughly

$n^k \operatorname{poly}(n, 2^k)$ time, where the n^k is an upper bound for $\binom{n}{k}$. We will follow an idea by Mossel, O'Donnell, and Servedio.

The first claim is that "finding one relevant variable is enough." Suppose given a k-junta f, you can find a relevant variable in time $T(n,k)$. Then you can find all relevant coordinates in time $T(n,k)2^k$. The idea is to recurse. After finding 1 coordinate x_i. For $b \in \{\pm 1\}$, go over all examples $x : x_i = b$ and find the relevant coordinates there recursively. This gives a runtime

$$T(n,k) + 2T(n,k-1) + 4T(n,k-2) + \cdots \leq 2^k T(n,k).$$

Proposition 1.1. *If f is a k-junta, then any Fourier coefficient is $\widehat{f}(S) = \text{integer}/2^k$.*

Proof. The key point is that g has the same Fourier coefficients as f.

$$\begin{aligned}\widehat{g}(T) &= \mathbb{E}_{X \in \{\pm 1\}^k}[g(X)\chi_T(X)] \\ &= \frac{1}{2^k} \underbrace{\sum_{x \in \{\pm 1\}^k} g(x)\chi_Y(x)}_{\text{integer}}.\end{aligned}$$

□

We can thus use the following low degree algorithm:

For $i = 1, 2, 3, \ldots,$

For all sets $S \subseteq [n]$ of size i,

Estimate $\widehat{f}(S)$ up to accuracy $\frac{1}{4} \cdot \frac{1}{2^k}$ to get $\widetilde{f}(S)$.

If $|\widetilde{f}(S)| \geq \frac{1}{2} \cdot \frac{1}{2^k}$, return S.

The runtime is $\approx n^t$, where t is the smallest number such that there exists an $s : |s| = t$ and $\widehat{f}(S) \neq 0$. The worst case fo the low degree algorithm is

$$f(x) = \text{PARITY}_k(x_{i_1}, \ldots, x_{i_k}),$$

(ignoring constant functions). Can we handle the worst case better?

Over $\mathbb{F}_2$, this function is

$$\begin{aligned}f(x) &= x_{i_1} + \cdots + x_{i_k} \pmod 2 \\ &= \sum_{i=1}^{n} \alpha_i x_i \pmod 2,\end{aligned}$$

where $\alpha_{i_1} = \alpha_{i_2} = \cdots \alpha_{i_k} = 1$ and all the rest are 0. Our goal is to find α, given random examples $(x^{(1)}, f(x^{(1)})), \ldots, (x^{(m)}, f(x^{(m)}))$. We can solve this system of linear equations by Gaussian elimination:

$$\begin{bmatrix} - & x^{(1)} & - \\ - & x^{(2)} & - \\ & \vdots & \\ - & x^{(m)} & - \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} f(x^{(1)}) \\ f(x^{(2)}) \\ \vdots \\ f(x^{(m)}) \end{bmatrix}$$

When $m = O(n)$, with high probability, this is a linear system with full rank. This guarantees a unique solution. This has runtime n^ω, where $2 \leq \omega \leq 2.372$ is the matrix multiplication exponent.

Let's generailze this idea: Parity functions have degree 1 over $\mathbb{F}_2$. What about functions with degree 2?

$$f(x) = \sum_{i<j} \alpha_{i,j} x_i x_j + \sum_{i=1}^{n} \alpha_i x_i + \alpha_0 \cdot 1.$$

$$\underbrace{\begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} & x_1^{(1)}x_2^{(1)} & x_1^{(1)}x_3^{(1)} & \cdots \\ & & & & & & & \\ & & & & & & & \end{bmatrix}}_{1+n+\binom{n}{2} \text{ columns}} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \\ \alpha_{1,2} \\ \alpha_{1,3} \\ \vdots \\ \alpha_{n-1,n} \end{bmatrix} = \begin{bmatrix} f(x^{(1)}) \\ f(x^{(2)}) \\ \vdots \\ f(x^{(m)}) \end{bmatrix}$$

This matrix has $1 + n + \binom{n}{2}$ columns, and solving this system of linear equations takes time $(n^2)^\omega$.

More generally, for functions with degree d,

$$f(x) = \sum_{S \subseteq [n], |S| \leq d} \alpha_S \prod_{i \in S} x_i \pmod 2.$$

$$\underbrace{\begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} & x_1^{(1)}x_2^{(1)} & x_1^{(1)}x_3^{(1)} & \cdots \\ & & & & & & & \\ & & & & & & & \end{bmatrix}}_{1+n+\binom{n}{2}+\cdots+\binom{n}{d} \text{ columns}} \begin{bmatrix} \alpha_\varnothing \\ \alpha_{\{1\}} \\ \alpha_{\{2\}} \\ \vdots \\ \alpha_S \\ \vdots \end{bmatrix} = \begin{bmatrix} f(x^{(1)}) \\ f(x^{(2)}) \\ \vdots \\ f(x^{(m)}) \end{bmatrix}$$

This matrix has $1 + n + \binom{n}{2} + \cdots + \binom{n}{d}$ columns, and solving this system of linear equations takes time $(n^d)^\omega$.

Let's compare these two algorithms:

- Low degree algorithm: Runs in time $\approx n^t$ if there exists an S such that $1 \leq |S| \leq t$ and $\widehat{f}(S) \neq 0$.
- Gaussian elimination: Runs in time $\approx n^{d\omega}$ if f has degree $\leq d$ as a multilinear polynomial over $\mathbb{F}_2$.

The main observation of Mossel, O'Donnell, and Servedio is that for any k-junta, one of these algorithms runs much faster than n^k. If for all S such that $|S| < t$, $\widehat{f}(S) = 0$, then $\deg_{\mathbb{F}_2}(f) \leq k - r$. Pick $t = \frac{\omega}{\omega+1} \cdot k \approx 0.7k$ to get runtime $n^t + n^{(k-t)\omega} \approx n^{\omega/(\omega+1)\cdot k}$. The proof of this observation will be given as a guided exercise in homework 3.

In 2012, Gregory Valiant[1] showed that you can speed up the low degree algorithm faster than n^t.

[1] Gregory Valiant is actually the son of Leslie Valiant who wrote the original PAC learning paper!