1 Administrative

The first project checkin is due on March 5th.

2 Last Time

Last time we covered Fourier basics, the LMN (Linial/Mansour/Nisan) algorithm, and the KM (Kushilevitz/Mansour) algorithm. Today we finish the KM algorithm and explore its applications, and start a unit on learning monotone Boolean functions using the LMN algorithm.

Recall \( f_{k,S_1}(x) = \sum_{T_2 \subseteq \{k+1,...,n\}} \hat{f}(S_1 \cup T_2) \chi_{T_2}(x) \) and \( \mathbb{E}[f^2_{k,S_1}] = \sum_{T_2 \subseteq \{k+1,...,n\}} \hat{f}(S_1 \cup T_2) \).

**Lemma 1.** Fix any \( f: \{-1,1\}^n \rightarrow \{-1,1\} \). Pretend that we can exactly compute \( \mathbb{E}[f^2_{k,S_1}] \) in unit time, for any given \( k, S_1 \subseteq [k] \). Then given \( \theta > 0 \), there is an algorithm that outputs exactly \( \{ S : |\hat{f}(S)| \geq \theta \} \) in \( \text{poly}(n, \frac{1}{\theta}) \) time.

**Proof.** The algorithm builds a partial binary tree on-the-fly where each level-\( k \) node has address \( (k, S_1) \) and value \( \mathbb{E}[f^2_{k,S_1}] \). Note that the node’s value is the sum of its two children’s value: node \( v \), at location \( (k, S_1) \), has children

- \( (k + 1, S_1) \), with value \( \sum_{T_2 \subseteq \{k+2,...,n\}} \hat{f}(S_1 \cup T_2)^2 \); and
- \( (k + 1, S_1 \cup \{k + 1\}) \), with value \( \sum_{T_2 \subseteq \{k+2,...,n\}} \hat{f}(S_1 \cup \{k + 1\} \cup T_2)^2 \).

In particular, for each \( k \) the sum of all level-\( k \) nodes of the complete binary tree is \( \mathbb{E}[f^2_{0,\emptyset}] = \mathbb{E}[f^2] = 1 \); hence the number of nodes at level \( k \) whose value is greater than \( \theta \) is less than \( \frac{1}{\theta^2} \).

The algorithm then builds the partial tree from the root \( (0, \emptyset) \). When it reaches a node with value \( < \theta^2 \), it calls it “dead” and stops exploring this branch. The algorithm
only continues in “live” nodes. Therefore each level contains at most $\leq \frac{1}{\theta^2}$ live nodes, and the total number of nodes ever explored is $\frac{n}{\theta^2}$ (since there are $n$ levels). Once the whole tree is built, the value at each leaf $(n, S_1)$ at the bottom level is $E[f_{n, S_1}^2] = \hat{f}(S_1)^2$, and these leaves correspond exactly to all $S_1 \subseteq [n]$ such that $E[f_{n, S_1}^2] \geq \theta^2$.

### 3 KM Algorithm

Now we would like to look at “almost-exact” computations, since we (as already observed) cannot get the exact values $E[f_{k, S_1}^2]$, but only approximate them. Note that we have $f_{k, S_1}(x) \leq 1$ for all $x, k \in \{0, \ldots, n\}$ and $S_1 \subseteq [k]$ because $f_{k, S_1}(x) = \mathbb{E}_{y \sim \{-1, 1\}^k}[f(yx)\chi_{S_1}(y)]$. So we estimate $E_x[f_{k, S}(x)^2]$ by picking a uniform random $x$ and estimating $f_{k, S}(x)$ and then taking the square. We are basically using sample points. Towards this end, we can prove the following:

**Lemma 2.** Given MQ$(f), f: \{-1, 1\}^n \rightarrow \{-1, 1\}$, $0 \leq k \leq n, S_1 \subseteq [k]$, $\tau, \delta$, there is a $\text{poly}(n, \frac{1}{\tau}, \log \frac{1}{\delta})$-time algorithm that with probability $\geq 1 - \delta$, outputs $v$ such that

$$|v - E[f_{k, S_1}^2]| \leq \tau.$$

**Proof.** Left as an exercise. \hfill \Box

Using this, the actual analogue of “almost-exact” is the KM algorithm.

**Theorem 3.** Let $f: \{-1, 1\}^n \rightarrow \{-1, 1\}$. Then given $\theta > 0$ and error probability $\delta > 0$, there is an algorithm (KM) that, with probability $1 - \delta$, outputs a collection $S$ of size $O(1/\theta^2)$ such that

- if $S \in S$, then $\left| \hat{f}(S) \right| \geq \frac{\theta}{2}$;
- if $S$ has $\left| \hat{f}(S) \right| \geq 2\theta$, then $S \in S$.

**Proof.** Left as an exercise. \hfill \Box

### 4 Applications of the KM algorithm

What can we use the KM algorithm for? It doesn’t work well for everything; some functions have only small Fourier coefficients. KM works best if most of the Fourier weight is on a “small” set of large coefficients.
Parities As a trivial case, we can use the KM algorithm to learn an unknown parity function: if \( f = \chi_S \) for some unknown \( S \subseteq [n] \), \( f \) has all its Fourier weight on only one non-zero coefficient; setting \( \theta \overset{\text{def}}{=} 0.9 \) will then ensure we find \( S \).

**Inner Product** An example of a function which “defeats” the KM algorithm (by having only very small Fourier coefficients) is the inner product function, defined as

\[
\text{IP}_2 : x \in \{-1, 1\}^{2n} \mapsto (x_1 \land x_2) \land (x_3 \land x_4) \land \cdots \land (x_{2n-1} \land x_{2n})
\]

Since \( x_1 \land x_2 = \frac{1}{2} (1 + x_1 + x_2 - x_1 x_2) \) (recall that True corresponds to \(-1\)), the Fourier representation of \( \text{IP}_2 \) is

\[
\text{IP}_2 = \sum_{S \subseteq [n]} \frac{\pm 1}{2^n} \chi_S
\]

and all coefficients have same tiny weight \( \frac{1}{2^n} \).

As we shall see momentarily, there exists a class of functions for which the Kushilevitz–Mansour algorithm is well-suited: the functions with bounded \( L_1 \) norm.

### 4.1 Learning function with small \( L_1 \) norm

**Definition 4.** Given \( f : \{-1, 1\}^n \to \{-1, 1\} \), define \( L_1(f) \overset{\text{def}}{=} \sum_{S \subseteq [n]} |\hat{f}(S)| \)

**Claim 5.** Given Boolean function \( f \) and \( \epsilon > 0 \), let \( S_\epsilon \overset{\text{def}}{=} \left\{ S : \left| \hat{f}(S) \right| \geq \frac{\epsilon}{L_1(f)} \right\} \). Then

1. \( |S_\epsilon| \leq \frac{L_1(f)^2}{\epsilon} \); and
2. \( f \) is \( \epsilon \)-concentrated on \( S_\epsilon \).

**Proof.** Fix \( S \in S_\epsilon \). Then \( \left| \hat{f}(S) \right| \geq \frac{\epsilon}{L_1(f)} \). We know by definition that

\[
L_1(f) = \sum_{S \subseteq [n]} \left| \hat{f}(S) \right| \geq \sum_{S \in S_\epsilon} \left| \hat{f}(S) \right| \geq |S_\epsilon| \cdot \epsilon
\]

which gives us the first item. For the second, the goal is to show that \( \sum_{S \in S_\epsilon} \hat{f}(S)^2 \leq \epsilon \).

This holds because

\[
\sum_{S \in S_\epsilon} \hat{f}(S)^2 \leq \max_{S \in S_\epsilon} \left| \hat{f}(S) \right| \cdot \sum_{S \in S_\epsilon} \left| \hat{f}(S) \right| < \epsilon
\]

as \( \sum_{S \in S_\epsilon} \left| \hat{f}(S) \right| \leq L_1(f) \) and \( \max_{S \in S_\epsilon} \left| \hat{f}(S) \right| \leq \frac{\epsilon}{L_1(f)} \). \( \square \)
Putting it together, we obtain an algorithm that can learn Boolean functions with small $L_1$ norm:

**Theorem 6.** There exists an algorithm $\text{Learn-L}_1$ which, given $\text{MQ}(f)$, $L = L_1(f)$, $\epsilon, \delta > 0$, outputs with probability $\geq 1 - \delta$ a Boolean function $h$ such that $\text{dist}(f, h) \leq \epsilon$, and runs in time $\text{poly}(n, \frac{1}{\epsilon}, L, \log(\frac{1}{\delta}))$. We can run $\text{LMN}$ algorithm to get $h$.

**Proof.** $\text{Learn-L}_1$ runs the KM algorithm to find a collection $S$ of $\text{poly}(L/\epsilon)$ many Fourier coefficients containing all coefficients in $S_{\epsilon/2}$; $f$ is then $\epsilon/2$-concentrated on $S$, and it suffices to use the LMN algorithm to learn $f$ to accuracy $\epsilon$. □

**Remark 1.** If the algorithm is not given $L$, one can still use it by “guessing” an upperbound on it (repeated guesses), and testing the hypothesis obtained until a good one is found.

**Corollary 7.** We can learn size-$s$ decision trees in time $\text{poly}(n, s, \frac{1}{\epsilon}, \log(\frac{1}{\delta}))$, where the size $s$ denotes the number of leaves in the decision tree.

**Proof.** This directly follows from the following claim:

**Lemma 8.** Let $f : \{-1, 1\}^n \rightarrow \{-1, 1\}$ be computed by a size-$s$ decision tree. Then $L_1(f) \leq s$.

To show this, fix leaf $v$ in the decision tree; call the bit label $b_v \in \{-1, 1\}$. Consider the function $g_v$ that outputs $b_v$ if $x$ reaches the leaf $v$ and 0 otherwise. The path to $v$ is an AND of literals: $\ell_{i_1}, \ell_{i_2}, \ldots, \ell_{i_k}$ where $\ell_{i_j} = \pm x_j$. What is the function $g_v$ then?

$$g_v(x) = b_v \left( \frac{1 + \ell_{i_1}}{2} \right) \left( \frac{1 + \ell_{i_2}}{2} \right) \cdots \left( \frac{1 + \ell_{i_k}}{2} \right) = \sum_{S \subseteq \{i_1, i_2, \ldots, i_k\}} \frac{\pm 1}{2^k} \chi_S(x).$$

Each path function $g_v$ thus has a simple Fourier representation, with each coefficient being $\pm 2^k$. The decision tree is $f = g_{v_1} + \cdots + g_{v_s}$ where $v_1, \ldots, v_s$ are the $s$ leaves, so $L_1(f) \leq \sum L_1(g_{v_i}) \leq s$. □

### 4.2 Sparse Fourier Representations

Another application is learning functions with sparse Fourier representations. Given a Boolean function $f$, let $\text{sparse}(f)$ be the number of its non-zero Fourier coefficients, that is

$$\text{sparse}(f) \overset{\text{def}}{=} \left| \left\{ S \subseteq [n] : \hat{f}(S) \neq 0 \right\} \right|$$

As a first observation, we have the following relation between sparsity and $L_1$ norm:
Fact 9. Fix any \( f : \{-1, 1\}^n \to \{-1, 1\} \). If \( \text{sparse}(f) = s \), then \( L_1(f) \leq \sqrt{s} \).

Proof. Let \( S_1 \ldots S_s \) be the sets corresponding to the non-zero coefficients of \( f \), so that 
\[
L_1(f) = \sum_i \left| \hat{f}(S_i) \right| = \sum_{i=1}^s \left| \hat{f}(S_i) \right|.
\]
By Cauchy–Schwartz,
\[
\sum_{i=1}^s \left| \hat{f}(S_i) \right| \leq \sqrt{\sum_{i=1}^s 1^2} \sqrt{\sum_{i=1}^s \left| \hat{f}(S_i) \right|^2} = \sqrt{s} \cdot 1
\]

Corollary 10. We can learn the class of \( s \)-sparse functions in time \( \text{poly}(s, n, \frac{1}{\epsilon}, \log(\frac{1}{\delta})) \).

Remark 2. There is no general relationship between sparsity and decision tree size. Consider \( \chi_{[n]} \), which is (super) sparse: \( \text{sparse}(\chi_{[n]}) = 1 \), but its decision tree size is \( 2^n \). On the other hand, the function AND\(_n\) has sparsity \( 2^n \), but decision tree size \( n \).

Remark 3. It is known that the KM algorithm can learn poly(\( n \))-term DNF to accuracy \( \epsilon \) in time \( n^{O\left(\log\log(\frac{n}{\epsilon})\right)} \).

5 Monotone Boolean Functions

We will see that the LMN algorithm will learn any monotone function \( f \) in \( n^{O\left(\sqrt{n}/\epsilon\right)} \) time (in particular, without membership queries). Let us first define a monotone Boolean function.

Definition 11. A Boolean function \( f : \{-1, 1\}^n \to \{-1, 1\} \) is said to be monotone if \( x \preceq y \) implies \( f(x) \leq f(y) \). Here, \( x \preceq y \) refers to the partial order on the Boolean hypercube, i.e. \( x \preceq y \) if and only if \( x_i \leq y_i \) for all \( i \in [n] \).

Remark 4. Note that there are a huge number of monotone functions (more than \( 2^\left(\frac{n}{2}\right) \)); to see this, consider (for \( n \) even) any function \( f \) that has \( f(x) = 1 \) when \( \sum_{i=1}^n x_i > 0 \) and \( f(x) = -1 \) when \( \sum_{i=1}^n x_i < 0 \). Any such function is monotone, and there are \( 2^\left(\frac{n}{2}\right) \) such functions since each of the \( \binom{n}{n/2} \) points \( x \) with \( \sum_i x_i = 0 \) can take either value +1 or −1.

So any naive method would result in an exponential time algorithm. We define a key notion to help us deal with monotone functions, which is the concept of influence.
5.1 Influence

Definition 12 (Influence of a Boolean function). Fix $x \in \{-1, 1\}^n$ and $b \in \{-1, 1\}$. Given $f: \{-1, 1\}^n \to \{-1, 1\}$, the influence of $x_i$ on $f$ is

$$\text{Inf}_i[f] \overset{\text{def}}{=} \Pr[ f(x_i \leftarrow 1) \neq f(x_i \leftarrow -1) ] .$$

The total influence of $f$ is $\text{Inf}[f] = \sum_{i=1}^n \text{Inf}_i[f]$. Let’s see a couple examples.

1. For the constant function $f = 1$, $\text{Inf}_i[f] = 0 \forall i$.
2. For the parity function $\chi[n] = f$, $\text{Inf}_i[\chi[n]] = 1 \forall i$
3. Consider majority function $f = \text{MAJ}_n(x) = \text{sign}(x_1 + \cdots + x_n)$, where $n$ is odd. What is the influence of the hidden variable? (Applications in voting). We can see that $\text{Inf}_i[\text{MAJ}_n] = \left(\frac{n-1}{2}\right)/2^n = \Theta(1/\sqrt{n})$, so $\text{Inf}[\text{MAJ}_n] = \Theta(\sqrt{n})$. We’ll see next time that this is in fact the largest possible influence for any monotone Boolean function.

6 Preview for next time

Next time we will talk about the connection between influence and the KM algorithm.

Lemma 13. Consider $f: \{-1, 1\}^n \to \{-1, 1\}$. Then $\text{Inf}_i[f] = \sum_{S \ni i} \hat{f}(S)^2$.

Proof. Define the discrete derivative in the $i$-th direction of $f$ as

$$D_i(f) = \frac{f(x_i \leftarrow 1) - f(x_i \leftarrow -1)}{2}$$

where $x_i \leftarrow b$ denotes $x$ with the $i$-th bit set to $b$. Note that by definition $D_i(f)(x) = 0$ if $x$ is “insensitive” and $\pm 1$ otherwise. By linearity of the Fourier transform, one gets

$$D_i(f(x)) = \frac{1}{2} \left( \sum_{S \subseteq [n]} \hat{f}(S) \chi_S(x) - \sum_{S \subseteq [n]} \hat{f}(S) \chi_S(x_{i \leftarrow 1}) \right) = \sum_{S \ni i} \hat{f}(S) \chi_{S \setminus \{i\}}(x).$$

and then

$$\text{Inf}_i[f] = \Pr[ x_i \leftarrow 1 \neq f(x_i \leftarrow -1) ] = \Pr[ D_i(f(x)) = \pm 1 ] = \mathbb{E}[D_i(f(x))^2] = \sum_{S \ni i} \hat{f}(S)^2 .$$

$\square$
Corollary 14. For any Boolean function $f$, the on the $i$-th variable is the Fourier weight of all the coefficients that contain $i$, and in particular

$$\text{Inf}[f] = \sum_{i=1}^{n} \sum_{S \ni i} \hat{f}(S)^2 = \sum_{S \subseteq [n]} |S| \hat{f}(S)^2.$$