6.842 Randomness and Computation April 2, 2012

Lecture 14

Lecturer: Ronitt Rubinfeld *Scribe: Aaron Sidford*

1 Review

In the last class we saw an algorithm to learn a function where very little of the Fourier coeffecient "mass" is on large sets. Formally, we defined a quantity called *Fourier Concentration* and gave an $O\left(\frac{n^d}{\epsilon}\log\frac{n^d}{\epsilon}\right)$ sample uniform learning distribution algorithm to learn a function with Fourier concentration d. We called this algorithm the low degree algorithm as it learns a function by generating a low Fourier degree to approximation to that function.

Definition 1 (Fourier Concentration) *$f:\{\pm1\}^n\to\mathbb{R}$ has $\alpha(\epsilon,n)$-fourier concentration if*

$$\sum_{S\subseteq[n]\ s.t.\ |S|\geq\alpha(\epsilon,n)}\hat{f}(S)^2\leq\epsilon$$

Remark [Boolean Fourier Concentration] By boolean Parseval's theorem we know that a boolean function $f:\{\pm1\}^n\to\{\pm1\}$ has Fourier coeffecients satisfying $\sum_{S\subseteq[n]}\hat{f}^2(S)=1$. Therefore, we have that if f has $\alpha(\epsilon,n)$-fourier concentration then

$$\sum_{S\subseteq[n]\text{ s.t. }|S|<\alpha(\epsilon,n)}\hat{f}(S)^2=\sum_{S\subseteq[n]}\hat{f}^2(S)-\sum_{S\subseteq[n]\text{ s.t. }|S|\geq\alpha(\epsilon,n)}\hat{f}(S)^2\geq1-\epsilon$$

Theorem 2 (Low Degree Algorithm) *If function family C has Fourier concentration d then there is an $O\left(\frac{n^d}{\epsilon}\log\frac{n^d}{\epsilon}\right)$ sample uniform distribution learning algorithm for C.*

2 Today

Today we will discuss two more applications of the low degree algorithm.

First we will look at constant depth circuits. We will cite a theorem regarding the Fourier concentration of functions computable by constant depth circuits and we will use this theorem as well as the low degree algorithm to show how we can learn constant depth circuits.

Second we will begin to develop the theory need to learn functions of halfspaces. To do this we will explore the notion of noise sensitivity which will ultimately allow us to bound the Fourier concentration of these types of functions and learn them through the low degree algorithm.

3 Learning Constant Depth Circuits

3.1 Basic Definitions

Informally, a circuit is a set of boolean input variables and boolean constants (i.e. T ("true") and F ("false")) connected to $\wedge$ ("and"), $\vee$ ("or"), and $\neg$ ("not") operators or "gates." Each gate receives as input one or more input variables, constants, or operators and sends its output to one or more other operators. One of these entities is designated to be the circuit output. Given circuit input (i.e. settings of the variables) the circuit output is determined by each gate computing it's output once it's input has been determined until the circuit output is determined. For this process to be well defined, we restrict the graph where the nodes are variables, constants, and gates and the edges demarcate which entity sends its output to which gate to be a directed acyclic graph (DAG).

Definition 3 (Boolean Circuit) *A boolean circuit C is a directed acyclic graph (DAG) where the nodes are each one of $\wedge$, $\vee$, $\neg$, T, F, and variable.*

We will be primarily interested in the functions computable by circuits satisfying certain parameters. In particular we will be interested in constraining the *size*, *depth*, and *fan-in* of circuits. The size refers to the number of gates in the circuit, the depth refers to the longest chain of gate dependencies in the circuit, and the fan-in refers to how many inputs a single gate can have.

Definition 4 (Circuit Size) *The* size *of a circuit C is the number of nodes in the DAG for that circuit.*

Definition 5 (Circuit Depth) *For a circuit C its* depth *is the length of the longest path in the DAG.*

Definition 6 (Fan In) *The* fan in *of a gate is simply the number of input that that gate has and the* fan in *of a circuit or family of circuits is the maximum number of inputs that any gate can have.*

3.2 Circuit Complexity

Today we will study two general family of functions computable by certain types of circuts:

- $AC0$, $AC1$, ..., ACk
- $NC0$, $NC1$, ..., NCk

Each family consists of functions computable by circuits of polynomial size and bounded depth. However, the families differ in the allowed fan-in of these circuits. In particular we define ACi and NCi as follows.

Definition 7 (ACi) *For all non-negative integers i, the circuit family ACi is the set of functions of n input variables computable by circuits of depth $O(\log^i(n))$, size $O(\text{poly}(n))$, and unbounded fan-in.*

Definition 8 (NCi) *For all non-negative integers i, the circuit family NCi is the set of functions of n input variables computable by circuits of depth $O(\log^i(n))$, size $O(\text{poly}(n))$, and constant bounded fan-in.*

Remark The limitation that ACi and NCi be of polynomial size is essential for the depth restriction to be meaningful. Without the polynomial size bound every function would be in $AC0$ since we could simply write out the DNF formula for the function (i.e. the "or" of the "and" of each precise input setting that makes the function true).

Remark $AC0$ and $NC0$ are circuits of constant depth. However, in $NC0$ the function's output cannot even be dependent on all the input since the fan in restriction means that it takes logarithmic levels for the output to be dependent on all the input.

3.3 Complexity Hierarchy

There is a natural hierarchy within the circuit families.

Theorem 9

$$NC0 \subseteq AC0 \subseteq NC1 \subseteq AC1 \subseteq \ldots$$

Proof First, since the definition of NCi is strictly more restrictive the the definition of ACi we have

$$\forall i \ : \ NCi \subseteq ACi$$

Furthermore, note that an "and" or "or" gate with d inputs is equivalent to a balanced tree of depth $O(\log d)$ of "and" or "or" gates on the same input. Therefore, given a circuit in ACi on n inputs we

can simply replace every "and" and "or" gate with such a balanced binary tree of "and" or "or" gates multiplying the depth by $O(\log n)$. Therefore we have just shown that

$$\forall i \ : \ ACi \subseteq NC(i+1)$$

Combining these two equations yields the result. ■

3.4 Switching Lemma

There is a strong separation result within the complexity hierarchy. The parity function is in $NC1$ however the parity function is in not in $AC0$. Therefore $AC0 \subseteq NC1$ but $NC1 \nsubseteq AC0$ so the inclusion $AC0 \subset NC1$ is proper.

To see that the parity function is in $NC1$ we first note that we can compute the xor of two input's using a constant number of circuits (i.e. $a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$). Therefore, since parity is the xor of all the input and since xor is a symmetric and associative operation we can implement parity of n inputs in $NC1$ by making a balanced binary tree of xors on n inputs obtaining a $NC1$ circuit of size $O(n)$, depth $O(\log(n))$, and fan in of 2.

However, the proof that parity is not in $AC0$ is much more complicated. In fact, whether or not parity was in $AC0$ was an open question for a while until Furst, Saxe, and Sipser first proved that parity is in fact not in $AC0$. Over time this result was improved in order to derive better bounds on the size and depth of a circuit needed to compute certain functions. Ultimately, the findings were distilled into Hastad's switching lemma, which was used by Linial, Mansour and Nisan to give the characterization below

Theorem 10 (Hastad, Linial, Mansour, Nisan) *For any function f computable a circuit C of size $\leq s$ and depth $\leq d$ we have*

$$\sum_{S \subseteq [n] \ s.t. \ |S|>t} \hat{f}^2(S) \leq \epsilon$$

for $t = O\left(\log \frac{s}{\epsilon}\right)^{d-1}$

Thus, this theorem says that functions computable by circuits of bounded size and depth have bounded Fourier concentration. We will not give a proof of this theorem, however we will show how it proves that parity is not in $AC0$ and we will use it to show how functions in $AC0$ can be learned by the low degree algorithm.

3.4.1 Parity $\notin AC0$

Let f be the parity function on n. We know that $f = \chi_{[n]}$ and therefore $\hat{f}([n]) = 1$ and $\hat{f}(S) = 0$ for $S \neq [n]$. Therefore,

$$\sum_{S \subseteq [n] \text{ s.t. } |S|>n-1} \hat{f}^2(S) = \hat{f}^2([n]) = 1$$

However, if a circuit C of unbounded fan in and size $s = \text{poly}(n)$ and depth d we know that by the switching lemma for $\epsilon = \frac{1}{2}$ and $t = O\left(\log \frac{s}{\epsilon}\right)^{d-1} = O(\log(n))^{d-1}$ we have

$$\sum_{S \subseteq [n] \text{ s.t. } |S|>t} \hat{f}^2(S) \leq \frac{1}{2}$$

Therefore, if C computes the parity function, combining these results implies $O(\log(n))^{d-1} = t > n-1$. Consequently $d = \Omega\left(\frac{\log n}{\log \log n}\right)$ so C does not have constant depth. Therefore, parity $\notin AC0$ as desired.

3.4.2 Learning Constant Depth Circuits

If a function f on n inputs is in $AC0$ then we know that it is computable by a circuit C of size $s = \text{poly}(n)$ and constant depth d. Therefore, by the switching lemma we have that for $t = O(\log \frac{s}{\epsilon})^{d-1} = O(\log \frac{n}{\epsilon})^{d-1}$ and it is the case that

$$\sum_{S \subseteq [n] \text{ s.t. } |S|>t} \hat{f}^2(S) \leq \epsilon$$

Therefore, by this Fourier concentration bound we can apply the low degree algorithm to learn f from the uniform distribution with $\tilde{O}\left(n^{\log^{d-1} \frac{n}{\epsilon}}\right)$ samples.

It is interesting to note how quickly these techniques can be used to get a bound on learning. The bound isn't polynomial but it is meaningful. Also, it is important to note that this bound can be improved to something that looks like $n^{O(\log \log n)}$ for functions using techniques of Jackson.

4 Noise Sensitivity of a Function

Now, we will switch topics slightly and discuss another general property of a function related to its learnability. We will discuss the *noise sensitivity* of a function. Informally, we will consider the process of switching bits of input with probability ϵ and we will measure how likely this modification to the input is to change the functions output. Intuitively, we will think of voting as a very noise insensitive function whereas parity is a very noise sensitive function.

4.1 Motivation

The killer application of noise sensitivity that we will consider is learning *halfspaces* also known as *linear threshold functions* (LTF). These are an important class of functions that receive a lot of attention in the theory community.

Definition 11 (Halfspace or Linear Threshold Function(LTF)) *A* halfspace *is boolean function* $h : \{\pm 1\}^n \to \{\pm 1\}$ *given by*

$$h(x) = \text{sign}(w \cdot x - \theta)$$

Where $w \in \mathbb{R}^n$, $\theta \in \mathbb{R}$ *and*

$$\text{sign}(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

We can think of a halfspace as a majority vote weighted by w on whether or not the input exceeds θ.

In this lecture and in the following lecture we will study the noise sensitivity of halfspaces and we will make connection between the noise sensitivity of a function and its fourier concentration. Ultimately, we will derive a bound on the Fourier concentration of a halfspace and then use the low-degree algorithm to learn it. In summary, we will use the machinery we develop today to later prove the theorem and corollary given below.

Theorem 12 *If* h *is halfspace over* $\{\pm 1\}^n$ *then* h *has Fourier concentration* $\alpha(\epsilon) = O\left(\frac{1}{\epsilon^2}\right)$ *i.e.*

$$\sum_{|S| \geq \frac{c}{\epsilon^2}} \hat{f}^2(S) \leq \epsilon$$

Corollary 13 *The low degree algorithm learns halfspaces under uniform distribution with* $\tilde{O}\left(n^{O(1/\epsilon^2)}\right)$ *samples.*

Remark There is $O\left(n^{1/5}\right)$ sample learning algorithm but we will not go over the technique.

4.2 Defining Noise Sensitivity

Here we formally define the noise sensitivity of a function. First we define a *noise operator* to formalize the notion of adding noise to the input and then we use this to define the noise sensitivity of a function.

Definition 14 (Noise Operator) *For* $0 < \epsilon < \frac{1}{2}$ *the* Noise Operator, $N_\epsilon(x)$, *is the operation of randomly flipping each bit of* x *independently with probability* ϵ.

Definition 15 (Noise Sensitivity) *For* $0 < \epsilon < \frac{1}{2}$ *the* noise sensitivity, $\text{ns}_\epsilon(f)$ *is defined as follows*

$$\text{ns}_\epsilon(f) = \Pr_{x \in \{\pm 1\}^n, N_\epsilon(x)}[f(x) \neq f(N_\epsilon(x))]$$

Therefore, the noise sensitivity of a function is simply the probability that the noise operator changes the output of f on a random input.

4.3 Examples

Here we compute the noise sensitivity of a variety of functions

4.3.1 Example 1: $f(x) = x_1$

Since f only depends on the first bit and since $NS_\epsilon(x)$ changes x with probability ϵ we have that the probability the noise operator changes f's output on any input is ϵ and therefore

$$\text{ns}_\epsilon(f) = \epsilon$$

4.3.2 Example 2: f is the "and" function of k inputs

For boolean $f : \{\pm 1\}^n \to \{\pm 1\}$ that is the function of k input we first break $\Pr_{x \in \{\pm 1\}^n, N_\epsilon(x)}[f(x) \neq f(N_\epsilon(x))]$ into the two cases that can cause $f(x) \neq f(N_\epsilon(x))$ and we have

$$\text{ns}_\epsilon(f) = \Pr\left[f(x) = F \wedge f(N_\epsilon(x)) = T\right] + \Pr\left[f(x) = T \wedge f(N_\epsilon(x)) = F\right]$$

However, since x is uniformly distributed we have that $N_\epsilon(x)$ is uniformly distributed and therefore the two cases are identical. Applying this fact and some more math we get the following

$$\begin{aligned}
ns_\epsilon(f) &= 2 \cdot \Pr[f(x) = T \wedge f(N_\epsilon(x)) = F] && \text{(Earlier reasoning)} \\
&= 2 \cdot \Pr\left[f(x) = T\right] \cdot \Pr\left[f(N_\epsilon(x)) = F \mid f(x) = T\right] && \text{(Conditional probability)} \\
&= \frac{2}{2^k} \Pr\left[f(N_\epsilon(x)) = F | f(x) = T\right] && (\Pr[f(x) = T] = \Pr[\text{all } k \text{ input are true}]) \\
&= \frac{2}{2^k} \left(1 - \Pr\left[f(N_\epsilon(x)) = T | f(x) = T\right]\right) \\
&= \frac{2}{2^k} \left(1 - (1-\epsilon)^k\right)
\end{aligned}$$

Where the last step comes from the fact that for $\Pr[f(N_\epsilon(x)) = T | f(x) = T]$ is the probability that the noise operator does not switch any of the k "and" input values.

Note how $(1-\epsilon)^k$ is either very small or close to $1 - \epsilon$ depending on k. This make sense since for $k = 1$ this is precisely example 1 and we get $\text{ns}_\epsilon(f) = \epsilon$ and for large k the function is almost always 0.

4.3.3 Example 3: $f(x) = Maj(x_1, \ldots, x_n)$

Claim 16

$$ns_\epsilon(f) = O\left(\sqrt{\epsilon}\right)$$

We won't actually prove this in full detail but we will give a proof sketch that should provide intuition for why this is true.

There are multiple ways to view the majority function. One way, is to view the majority function as simply computing the sum of the input and that outputting $+1$ or -1 depending on whether or not the sum is positive or negative.

However, when the input to the majority function is uniform on $\{\pm 1\}^n$ we can also view the majority function as a random walk on the line. Think of the majority function as starting at point 0 and then processing the input $x_1, \ldots, x_n$ in order. If $x_i = 1$ the random walk moves to the right one unit and if $x_i = -1$ the random walk moves to the left one unit. If the random walk terminates at a positive coordinate the majority function outputs 1 and if the random walk terminates at a negative coordinate then the majority function outputs -1.

One of the benefits of viewing the majority function as a random walk like this is that we know that its expected distance from start is $\sqrt{n}$ and we know that the random walk is very likely to be near this $\sqrt{n}$ distance from 0.

Now, we can view the majority function of the noise operator as taking the above random walk and then continuing the random walk on an expected $n\epsilon$ bits where each move takes 2 steps rather than 1 (Since the noise operator switches an expected ϵn input bits and each switch changes the majority sum by 2).

By the same arguments we see that the noise operator is expected to displace the majority random walk by $O\left(\sqrt{n\epsilon}\right)$. However, if the majority random walk ends up at distance $\sqrt{n}$ from 0 then as long as the noise operator does not displace the majority random walk by more than $\sqrt{n}$ then we have that the functions value does not change. Therefore, if we could loosely apply Markov's inequality we could say

$$\mathrm{ns}_\epsilon = \Pr[N_\epsilon \text{ changes output}] \leq \Pr\left[N_\epsilon \text{ displacement} > \sqrt{n}\right] \leq \frac{2\sqrt{n\epsilon}}{\sqrt{n}} = O(\sqrt{\epsilon})$$

and the argument would be complete. However, this argument is cheating all over the place. The majority random walk doesn't always end at distance $\sqrt{n}$ and there is a range of probabilities for different distance the walk can terminate at. Also, the noise operator doesn't always modify exactly ϵn bits and which direction the noise operator walk goes may be a function of how far the original walk goes. However, with much more careful analysis it is possible to obtain the desired result.

4.3.4 Example 4: Any LTF

Here we just cite the following the theorem of Peres.

Theorem 17 (Peres)

$$ns_\epsilon(LTF) < 8.8\sqrt{\epsilon}$$

Remark This result is the asymptotically best possible result, i.e we can show that $ns_\epsilon(LTF) = \theta(\sqrt{\epsilon})$ or that there is a family of LTF's, f, where $\mathrm{ns}_\epsilon(f) = \Omega(\sqrt{\epsilon})$. However, we will not prove this result here.

Remark Note that Examples 1, 2, and 3 were all LTFs. To see this, look at the definition of a LTF. For each example there is a choice of w and θ that makes the halfspace equal the function.

- Example 1: Set $w_1 = 1$, $w_j = 0$ for $j \neq 1$, and $\theta = 0$.
- Example 2: Set $w_i = 1$ if i is one of the k input variables, $w_i = 0$ otherwise, and $\theta = k$.
- Example 3: Set w to be the all ones vector and $\theta = 0$.

4.3.5 Example 5: Parity Functions

Consider $f = \chi_S(x)$ for $|s| = k$. (*Note that for $k = 1$ we have that f is a LTF.*) By definition of noise sensitivity we have that

$$ns_\epsilon(f) = \Pr[\text{odd \# bits in } S \text{ flipped}] = \binom{k}{1}(1-\epsilon)^{k-1}\epsilon + \binom{k}{3}(1-\epsilon)^{k-3}\epsilon^3 + \dots$$

Now using that

$$1^k = ((1-\epsilon)+\epsilon)^k = (1-\epsilon)^k + \binom{k}{1}(1-\epsilon)^{k-1}\epsilon + \binom{k}{2}(1-\epsilon)^{k-2}\epsilon^2 + \dots$$

subtracting

$$(1-2\epsilon)^k = ((1-\epsilon)-\epsilon)^k = (1-\epsilon)^k - \binom{k}{1}(1-\epsilon)^{k-1}\epsilon + \binom{k}{2}(1-\epsilon)^{k-2}\epsilon^2 + \dots$$

and dividing by 2 we get

$$\frac{1-(1-2\epsilon)^k}{2} = \mathrm{ns}_\epsilon(f)$$

5 Next Lecture

Next lecture we will show a relationship between noise sensitivity of a function and Fourier concentration of a function. This will give us a very powerful tool and a learning algorithm for halfspaces and arbitrary functions of k halfspaces.