6.842 Randomness and Computation March 19, 2022

Lecture 14

Lecturer: Ronitt Rubinfeld *Scribe: Isabelle Quaye*

1 Introduction

In this lecture, we discuss linearity testing, self-correction and introduce the Boolean cube.

2 Problem setup and definition

We begin by defining terms associated with linearity testing:

Definition 1 (*Linear function)* *Given a function f that maps from a finite set G to another finite set H, we say f is linear if $\forall\ x, y\ \ f(x) +_H f(y) = f(x +_G y)$.*

Here, we define $+_H$ to be addition as defined within the set H and $+_G$ to be addition as defined within the set G. Examples of such linear functions is the family of functions parameterised by a $f_a(x) = ax \bmod m, \forall x \in Z_m$ [†]. In this example $G = H$. This is linear because $f_a(x_1) + f_a(x_2) = ax_1 + ax_2 = a(x_1 + x_2) = f_a(x_1 + x_2)$

We also define a closely related concept called ϵ-linear:

Definition 2 (ϵ-*Linear function)* *Given a function f that maps from a finite set G to another finite set H, we say f is ϵ-linear if there exists a linear function g such that $Pr_{x \in G}[f(x) = g(x)] \geq 1 - \epsilon$.*

Given these established definitions, we want to understand how we could test if a function is linear. Since both G and H are finite, we could simply run the check on all possible combinations of x and y. However, G and H could turn out to be very large. We seek a more tractable way to do this.

3 Testing for linearity

As seen in previous lectures, one way to deal with intractability is to introduce randomness. To this end, we present the following randomized algorithm:

```
Do k times:
     Pick random x, y in G
     Test f(x+y) = f(x) + f(y)
```

Figure 1: Call this algorithm A_1

An important question here, is what should k be? That is, how many times should we run this algorithm so that we can effectively test if f is linear? This is important because it is possible that for some specific k we choose, the function f could pass the linearity test at all those points but could be far from linear. In that case, we want to be able to express how useful A_1 is. To answer these questions, we make some useful observations and discuss the concept of self-correction.

[†]Recall that $Z_m = [0, ..., m-1]$

3.1 Useful Observations

Suppose we choose some a, y and we want to know $Pr_{x \in_u G}[y = a + x]$. That is, we want to know the probability that for some a and y, $y = a + x$ for any x chosen uniformly at random from G. But with some rearranging, we realise that we can rewrite this probability as follows:

$$Pr_{x \in_u G}[y = a + x] = Pr_{x \in_u G}[x = y - a]$$

But since x chosen uniformly at random from G, the probability x equals any value is simply $\frac{1}{|G|}$. Therefore, it follows that if x is picked uniformly at random from G, then $y = a + x$ is also uniformly distributed in G.
This is an important observation and comes up in many places. Consider another example where $G = Z_2^{n\dagger}$ and addition under this set is bitwise i.e. $(a_1, ..., a_n)+(b_1, ..., b_n) = (a_1+b_1, a_2+b_2, ..., a_{n-1}+b_{n-1}, a_n+b_n)$. Then if we have $a = (0, 1, 1, 0)$ and $x = (b_1, b_2, b_3, b_4)$, under the assumption that x was picked uniformly at random from G, then the result $(0 + b_1, 1 + b_2, 1 + b_3, 0 + b_4)$ is also uniformly distributed.

3.2 Self correction aka "random self-reducibility"

Suppose you are given a function $f : G \to G$ * such that there exists a linear function $g : G \to G$ and $Pr_{x \in G}[f(x) = g(x)] \geq \frac{7}{8}$. Some important notes on this statement:

- This definition is closely related to the definition of ϵ-linear.($\epsilon = \frac{1}{8}$ here)
- We do not know what f looks like; it is handed to us as a blackbox whose functional form is unknown. However, by making calls to f, we can approximate what g is.
- There are likely to be multiple such g but we focusing on show that we can approximate one such g with only $O(\log \frac{1}{\beta})$ calls to f but with probability of error at most β.

The idea that we can approximate a linear function g from another function f that may or may not be linear gives rise to the idea of a *self-corrector*. The idea behind a self-corrector is that we would ideally like to compute $g(x)$ but we do not know what it is and do not have blackbox access to it either. However, there exists a function $f(x)$ which usually agrees with $g(x)$ (at least $\frac{7}{8}$ of the time). We can therefore use calls to $f(x)$ to estimate $g(x)$. The algorithm is as follows:

```
For i=1 to c*log(1/β):
     Pick y uniformly at random from G
     Record answer = f(y) + f(x-y) for this round
Output the most common answer from all rounds.
```

Figure 2: $A_2(x)$: Algorithm runs on the same input x for each of the $c \log \frac{1}{\beta}$ rounds

We make our first claim for the algorithm:
Claim 1 : $Pr[A_2(x) = g(x)] \geq 1 - \beta$.
Proof: Recall that

$$Pr[f(y) \neq g(y)] \leq \frac{1}{8}$$

†This notation simply means that G consists of vectors of length n whose entries are either 0 or 1.
*Could also do this definition in terms of $G \to H$

$$Pr[f(x-y) \neq g(x-y)] \leq \frac{1}{8}$$

from our definition above. Using union bound on these two probabilities, we can write that :

$$Pr[f(y) = g(y) \& f(x-y) = g(x-y)] \geq 1 - \left(\frac{1}{8} + \frac{1}{8}\right) = \frac{3}{4}$$

.
Since $Pr[f(y) + f(x-y) = g(y) + g(x-y)] \geq Pr[f(y) = g(y)\,\&\, f(x-y) = g(x-y)] \geq \frac{3}{4}$ [†],
we can write that:

$$Pr[f(y) + f(x-y) = g(y) + g(x-y)] \geq \frac{3}{4}$$

But notice that $f(y)+f(x-y)$ is the recorded answer for a single round in A_2 and $g(x) = g(y)+g(x-y)$. Therefore, this tell us the probability that the answer we get on any round is indeed $g(x)$. To analyze this over all the rounds, we use Chernoff bounds where the X_is are indicator variables for round i and is 1 iff in that round the recorded answer is indeed equal to $g(x)$ and 0 otherwise

$$Pr[\text{failure}] = Pr[X < \frac{1}{2} * \mu] < e^{-\frac{3k}{32}} \quad , \quad \mu = \frac{3k}{4}$$

Since we want an error of at most β, it follows that:

$$\beta = e^{-\frac{3k}{32}}$$

Rearranging yields:

$$k = \frac{32}{3} \ln\left(\frac{1}{\beta}\right)$$

so we indeed require $O(\log(\frac{1}{\beta})$ rounds to achieve an error of at most β.

3.3 Linearity Testing is non-trivial

The motivation for discussing self-correction and linearity testing is that if a given function is ϵ-linear and we can test for it, we can hopefully correct for it as we did in the previous subsection because we know the Pr[failure] However, developing tests for linearity is notoriously difficult. To see this, consider the same linearity testing scheme in Figure 1 and suppose we have a function $f : Z_m \to Z_m$. We have that f is defined as follows:

$$f(x) = \begin{cases} 1 & x \equiv 1 \mod 3 \\ 0 & x \equiv 0 \mod 3 \\ -1 & x \equiv 2 \mod 3 \end{cases}$$

A graph of the function is found in Figure 3. From the graph, we see that the best linear function $g(x) = 0$ that best approximates $f(x)$ will only align with at most $\frac{1}{3}$ of the points in $f(x)$. Therefore, we say that $g(x)$ is $\frac{2}{3}$-far from linear.
Notice that when $g(x) = 0$, our algorithm A_1 only fails[‡] anytime $x \equiv y \equiv 1 \mod 3$ and whenever $x \equiv y \equiv 2 \mod 3$ but it passes for all other x, y. Therefore with our algorithm, the function f succeeds on a large number of x, y even though it is clearly far from linear. We call this example the **Coppersmith's Example** and it is an example of the worst-case.

[†]One way to see this is to consider probability from a counting perspective. The number of ways that $f(y)+f(x-y) = g(y)+g(x-y)$ includes the case where $f(y) = g(y) f(x-y) = g(x-y)$. As a result, the probability of the former occurring is larger than just the latter.

[‡]Here fails means that the test on the third line with f fails

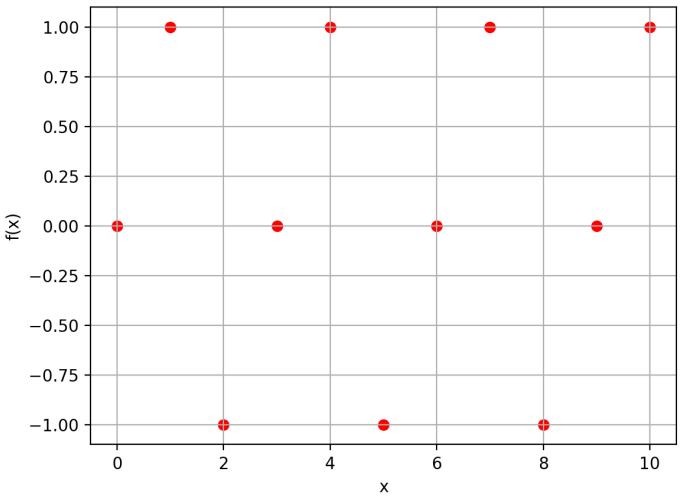

Figure 3: "tough" function $f(x)$ from Coppersmith's Example

3.4 Rejection probability

As seen in the previous subsection, there are cases where our algorithm does not perform well for certain choices of f and x, y. As a result, our algorithm isn't guaranteed to return the correct answer all the time. To this end, we define δ_f to be the rejection probability of linearity testing which is the probability that for any x, y $f(x) + f(y) \neq f(x + y)$. In the Coppersmith example, $\delta_f = \frac{2}{9}$ for a function f that is $\frac{2}{3}$-far from any $g(x)$. Since the Coppersmith Example is a worst case scenario, $\frac{2}{9}$ is in fact a threshold for defining an important theorem:

Theorem 1: If $\delta_f < \frac{2}{9}$, then f must be δ_f-close to linear [§]

Given this theorem about the rejection probability, we can set k from A_1 to $\frac{1}{\delta_f}$. The reasoning here is that if a function is far from linear, then there is some probability of rejection and so we exploit that probability to find the number of times we need to repeat the algorithm A_1 until we encounter a failing x, y.

4 Linearity Testing for Boolean functions

In this section, we restrict ourselves to looking at linearity testing for Boolean functions i.e. $f : G \to \{0, 1\}$. to be able to analyze this case, we need a few important definitions and tools like Fourier analysis over Boolean cube. We define these tools and definitions today and will finish up Fourier analysis over Boolean cube next lecture.

[§]Or equivalently $(1 - \delta_f)$-far from linear.

4.1 Boolean cube

The Boolean cube of size n is best described by example:

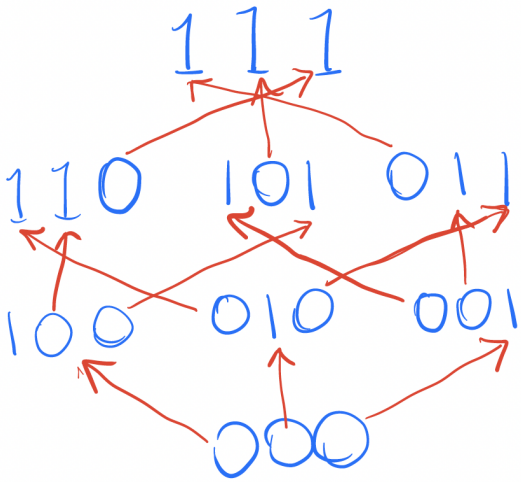

Figure 4: Boolean cube for $n = 3$.

It consists of i 0s and $n - i$ 1s at each level i[¶]. It is funnel shaped at both ends with 111 at the top and 000 at the bottom. The arrows are drawn between levels to connect bits that differ by a single bit flip from 0 to 1.

4.2 Inner product and notation change

We define the *inner product* on two n-bit binary digits x, y as:

$$x \cdot y = \sum_{i=1}^{n} x_i y_i \mod 2$$

From here, we define a set of linear functions on the domain $\{0,1\}^n$:

$$L_a(x) = ax$$

for a fixed $a \in \{0,1\}^n$. Since these functions are parameterized by a and there are 2^n choices for a, there are also 2^n of these functions. However, using our inner product notation, we can rewrite this as:

$$L_a(x) = L_A(x) = \sum_{i \in A} x_i \mod 2$$

where A are the set of indices i where $a_i = 1$

The change of notation we propose is for the function f. Before we had $f : \{0,1\}^n \to \{0,1\}$. Now we change f such that $f : \{+1,-1\}^n \to \{+1,-1\}$. Essentially, we are letting $0 \to +1$ and $1 \to -1$. This means we have that:

$$a \to (-1)^a \quad (A)$$

[¶]Count levels from the top downward as you would a recursion tree

$$a + b \to (-1)^{(a+b)} = (-1)^a(-1)^b \quad (B)$$

Essentially by making this change, we are mapping addition to multiplication[‖]. This also changes our linearity condition:

$$f(a) + f(b) = f(a + b) \to f(a)f(b) = f(a * b)$$

As a result, A_1 changes also. It's condition for passing the test is as follows:

$$f(x)f(y)f(x * y) = \begin{cases} 1 & \text{test accepts} \\ -1 & \text{test rejects} \end{cases}$$

We can redefine this test for a passed round as follows:

$$\frac{1 - f(x)f(y)f(x * y)}{2} = \begin{cases} 0 & \text{test accepts} \\ 1 & \text{otherwise} \end{cases}$$

We also redefine the rejection probability from $\delta_f = Pr_{x,y}[f(x) + f(y) \neq f(x + y)]$ to $\delta_f = Pr_{x,y}[f(x)f(y) \neq f(x * y)]$

5 Conclusion

In the next lecture, we look at Fourier analysis on the Boolean cube which will help us get stronger bounds than we achieved with Theorem 1.

[‖] We see this clearly in (B)