CS 579: Computational Complexity Fall 2025

Lecture 04 - September 04

Prof. Fernando Granha Jeronimo *Super Scribes: Pranav Rajpal & Andrei Staicu*

1 Previous Lecture

Recall that the Hadamard code is the set of evaluation points of linear functions l_S over $\mathbb{F}_2^n$,

$$\mathcal{H}_n = \{\text{Eval}(l_S) \mid S \subset [n]\} \subset \mathbb{F}_2^N, \quad N = 2^n \tag{1}$$

Hadamard code has the following properties: it has relative distance $\delta(\mathcal{H}_n) = 1/2$, rate $r(\mathcal{H}_n) = \frac{\log_2 N}{N}$, is a linear code, and is locally testable.

The local testing procedure is as follows; sample $x, y \in \mathbb{F}_2^n$ uniformly and accept if $f(x+y) = f(x) + f(y)$. We proved last lecture that if ϵ is the minimum relative hamming distance from f to a linear function then:

$$\Pr(\text{BLR test passes}) \leq 1 - \epsilon = 1 - \left(\min_{S \subset [n]} \delta(\chi_S, f)\right) \tag{2}$$

Therefore, if the test passes with high probability then we can say that ϵ is small and so f is close to linear, and conversely functions that are close to linear are more likely to pass the BLR test.

2 Hadamard Code is Locally Decodable and Correctable

As with local testability, a code being locally decodable or locally correctable is in the eye of the beholder (the locality or number of queries can be considered in various parameter regime: constant, logarithmic, sub-linear, etc). We will use the following informal definition:

Definition 2.1. *A code $C \subseteq \Sigma^n$ is **locally correctable** [1] if, when some $x \in \Sigma^n$ is ε-close to some $c' \in C$ (i.e. $\delta(x, c') \leq \varepsilon$), so the fraction of bits flipped between c' and x is at most ε), then any arbitrary symbol in the nearest codeword c' can be recovered with high probability using relatively few queries to x.*

As an example of a locally correctable code, we can show that Hadamard codes, in addition to being locally testable, are also locally correctable codes.

Specifically, we're given some arbitrary string $\text{Eval}(f) \in \mathbb{F}_2^N$ that represents the evaluation vector of a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$, and we know that $\delta(f, f_S) \leq \epsilon$ for some linear function f_S. We are also given some $x \in \mathbb{F}_2^n$ and we want to compute $f_S(x)$. We can do that using the following procedure:

a) Sample $y \in \mathbb{F}_2^n$ randomly.

b) Query $f(x+y)$ and $f(y)$ and return $f(x+y) + f(y)$.

By the union bound, the event that either $f(x+y)$ or $f(y)$ are corrupted happens with probability at most 2ϵ since each is corrupted with probability at most ϵ. Thus with probability at least $1 - 2\epsilon$ we have that $f(x+y) = f_S(x+y)$ and $f(y) = f_S(y)$, meaning that the returned value is $f_S(x+y) + f_S(y) = f_S(x)$. Importantly, that analysis holds even when the locations of the errors in f and the location of x are chosen adversarially, as opposed to just returning $f(x)$ directly, which would have a probability of failure of 1 if an adversary intentionally queried a location where the input vector had been corrupted.

Notice that given any x, this procedure can be repeated multiple times and, since the value of y is sampled independently on each run of that algorithm, we can return a majority vote of the values returned from each run to "boost" the probability of the correct value being returned. That lets us turn this into an algorithm that succeeds with high probability by repeating the test to make the probability of failure arbitrarily small.

Definition 2.2. *A code C is* ***locally decodable*** *[2, 4] if, for some message y that gets encoded to a codeword $c \in C$, observing a corrupted codeword $x \in \Sigma^n$ that is ϵ close to c (i.e. $\delta(x, c) = \epsilon$), one can efficiently recover any index y_i of the message y with high probability by only reading relatively few entries of x.*

We can show that Hadamard codes are also locally decodable codes by designing a fairly simple algorithm to recover the message (which would be represented by the set S in the function f_S) for a corrupted version f of the corresponding codeword.

The idea is that, now that we have shown that Hadamard codes are locally correctable, we can use our above algorithm as a black box to read out the value of $f_S(x)$ for any arbitrary x. Specifically, the decoding procedure involves the locally correctable queries: if $f_S(x)$, then we can query f at e_i (the vector with a single 1 at entry i and 0s everywhere else) to see if $i \in S$. For that, we are using the fact that

$$f_S(e_i) = 1 \iff i \in S \tag{3}$$

since XORing 0 with anything does not change its value, so the only entry that can have an impact on the value of $f_S(e_i)$ is entry i.

Using the local decodability of the Hadamard code, we can learn the linear function by decoding every message symbol $i \in [n]$, then in $n = \log N$ steps we can recover the entire message S. In this case, note that learning is more expensive than locally testing, correcting, or decoding.

3 Boolean Fourier Analysis and Spectral Graph Theory

Definition 3.1. *The* Boolean hypercube *is the following (family of) graph(s):*

$$B^n = (\mathbb{F}_2^n, \{(x, y) \mid \Delta(x, y) = 1\}) \tag{4}$$

The Boolean hypercube can be equivalently defined in the language of group theory, which will be very useful in shedding light on its algebraic structure, as we shall see shortly. Specifically, it can be viewed as a Cayley graph, which is a graph created from a group H and a generating set $T \subseteq H$ of that group where each group element in H becomes a vertex in the graph, and for every generating set element $t \in T$ and every group element $g \in H$, we add an edge from the vertex for g to the vertex for the group element tg. In our case, the group here is $\mathbb{Z}_2^n$ over addition (which we will write as $\mathbb{F}_2^n$ for convenience) with generators $T = \{e_1, \ldots, e_n\}$, where $e_i \in \mathbb{F}_2^n$ is a vector with a 1 in position i and 0s everywhere else, which is written as $\mathrm{Cay}(\mathbb{Z}_2^n, T)$.

Definition 3.2. *An adjacency matrix $A \in \mathbb{R}^{N \times N}$ of a graph $G = (V, E)$ with $|V| = N$ is defined by:*

$$A_{uv} = \begin{cases} 1 & \text{if } (u, v) \in E \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

If G is d-regular (i.e. every vertex has degree d), then the normalized adjacency matrix $\hat{A}$ is defined as $\frac{1}{d}A$.

Proposition 3.3. *For all χ_S, χ_S is an eigenvector of $\hat{A}$, the normalized adjacency matrix of the Boolean hypercube graph, and the corresponding eigenvalue would be*

$$1 - \frac{2|S|}{n} \tag{6}$$

Proof. We can first simplify the value of entry x in the result of $\hat{A}\chi_S$:

$$
\begin{aligned}
(\hat{A}\chi_S)_x &= \sum_{y\in\mathbb{F}_2^n} \hat{A}_{x,y}\chi_S(y) \\
&= \frac{1}{n}\sum_{y\in\mathbb{F}_2^n} A_{x,y}\chi_S(y) \\
&= \frac{1}{n}\sum_{i=1}^{n} \chi_S(x+e_i) && A_{x,y}\neq 0 \iff y = x+e_i \\
&= \frac{1}{n}\sum_{i=1}^{n} \chi_S(x)\chi_S(e_i) && \chi_S \text{ is a homomorphism} \\
&= \left(\frac{1}{n}\sum_{i=1}^{n} \chi_S(e_i)\right)\chi_S(x)
\end{aligned}
$$

From that, we can see that the sum that $\text{Eval}(\chi_S)_x$ is multiplied by is only dependent on S, not on the value of x, which means that for a fixed S that expression is a constant (i.e. it's an eigenvalue of the vector $\text{Eval}(\chi_S)$).

We can then simplify the expression for that eigenvalue (which we will call λ_S) using the fact that $\chi_S(e_i) = -1$ if and only if $i \in S$ (since all other entries of e_i are 0 so regardless of whether those components are included in S they will not affect the value of the product). Using that, we have that

$$
\begin{aligned}
\lambda_S &= \frac{1}{n}\sum_{i=1}^{n} \chi_S(e_i) \\
&= \frac{1}{n}\sum_{i=1}^{n} (-1)^{\mathbf{1}_{i\in S}} \\
&= \frac{1}{n}\left(\sum_{i\in S} -1 + \sum_{i\notin S} 1\right) \\
&= \frac{1}{n}(-|S| + (n-|S|)) \\
&= \frac{n-2|S|}{n}
\end{aligned}
$$

Thus the eigenvalue is

$$\lambda_S = 1 - \frac{2|S|}{n} \tag{7}$$

□

The reason that we are discussing eigenvalues of the Boolean hypercube here is that the eigenvalues of any given graph give us some insight into the structure of the graph itself, as can be seen from the following statements.

Fact 3.4. *For a d-regular graph, $\hat{A}$ has eigenvalue -1 iff it is bipartite.*

Since $\lambda_{[n]} = -1$, this fact shows that the Boolean cube is bipartite. Indeed, vertices with even Hamming weight connect only to vertices with odd Hamming weight.

Definition 3.5 (Spectral Gap)**.** *The spectral gap of a graph G is the difference between the 2 largest eigenvalues of the normalized adjacency matrix in absolute value.*

The spectral gap of the Boolean cube is:

$$\frac{2}{n} = \frac{2}{\log_2 N} \tag{8}$$

The spectral gap can be connected to how fast (some) probability distribution across vertices of G spreads out into a uniform distribution when treating G as a Markov chain [3]. Although some care is needed, can you find a distribution on the vertices of the Boolean hypercube that does not converge to the uniform distribution?

4 Complexity of Boolean Functions

Part of the reason that we are interested in boolean Fourier analysis is that it is a useful tool for analyzing the complexity of boolean functions naturally arising from many computational models such as various circuits models, decision trees, etc. In order to do that, we will start here by defining some notions of complexity for boolean functions that might be interesting to look at.

Definition 4.1. *The* sensitivity *[5] of a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ at some input $x \in \mathbb{F}_2^n$ is the number of bit positions in x where flipping a single bit changes the output, or more formally:*

$$S(f, x) = |\{i \in [n] \mid f(x) \neq f(x + e_i)\}| \tag{9}$$

The maximum sensitivity $S(f)$ *would then be the largest sensitivity of any input:*

$$S(f) = \max_x S(f, x) \tag{10}$$

Definition 4.2. *If $f = \sum_S \hat{f}(S)\chi_S$, then the Boolean degree of f is the size of the largest set S that shows up in the Fourier series version of f:*

$$\deg(f) = \max\{|S| \mid \hat{f}(S) \neq 0\} \tag{11}$$

Equivalently, if we view f as mapping from $\{\pm 1\}^n$ to $\{\pm 1\}$ (using the mapping $x \mapsto (-1)^x$ to convert from $\mathbb{F}_2$ to $\{\pm 1\}$), then the Boolean degree is the degree of the polynomial created when writing f as a Fourier expansion (since in that case $\chi_S(x) = \prod_{i \in S} x_i$ would be a polynomial of degree $|S|$).

4.1 Decision Trees for Boolean Functions

A *decision tree* for $f : \{0,1\}^n \to \{0,1\}$ is a rooted binary tree where:

- Each internal node is labeled with a variable x_i.
- Each edge corresponds to an assignment $x_i = 0$ or $x_i = 1$.
- Each leaf is labeled with an output value 0 or 1.

On input $x \in \{0,1\}^n$, the computation follows a path from the root according to the values of queried variables until reaching a leaf, whose label is the output $f(x)$.

The *decision tree complexity* of f, denoted $D(f)$, is the minimum depth of a decision tree computing f.

We encourage you to prove the following simple lemma related the Fourier degree and the decision tree complexity of a Boolean function.

Lemma 4.3. *For every Boolean function $f : \{-1,1\}^n \to \{-1,1\}$,*

$$\deg(f) \ \leq \ D(f).$$

References

[1] "Locally correctable code (LCC)". In: *The Error Correction Zoo*. Ed. by Victor V. Albert and Philippe Faist. 2023. URL: https://errorcorrectionzoo.org/c/lcc.

[2] "Locally decodable code (LDC)". In: *The Error Correction Zoo*. Ed. by Victor V. Albert and Philippe Faist. 2023. URL: https://errorcorrectionzoo.org/c/ldc.

[3] Sanjeev Arora and Boaz Barak. "Pseudorandom constructions: Expanders and extractors". In: *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009, pp. 421–459. ISBN: 9780511804090. DOI: 10.1017/CBO9780511804090.024. URL: https://doi.org/10.1017/CBO9780511804090.024.

[4] Tali Kaufman and Michael Viderman. "Locally Testable vs. Locally Decodable Codes". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Ed. by Maria Serna et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 670–682. ISBN: 978-3-642-15369-3. DOI: 10.1007/978-3-642-15369-3_50. URL: https://doi.org/10.1007/978-3-642-15369-3_50.

[5] Claire Kenyon and Samuel Kutin. "Sensitivity, block sensitivity, and ℓ-block sensitivity of boolean functions". In: *Information and Computation* 189.1 (2004), pp. 43–53. ISSN: 0890-5401. DOI: https://doi.org/10.1016/j.ic.2002.12.001. URL: https://www.sciencedirect.com/science/article/pii/S0890540103002530.