1 Last time and today

- Finish proof of Siegenthaler’s Theorem (junta learning)
- Learning PAR under noise: we’ll give a $O(2^n / \log n)$ algorithm.
- Uniform learning with membership queries

Relevant Readings:
- Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model JACM., vol. 50, iss. 4 2003 pp. 506-519

2 Finishing the proof of Siegenthaler’s Theorem

Recall:

**Theorem 1.** Let $f : \{-1, 1\}^r \rightarrow \{-1, 1\}$ be $d$th-order correlation immune (i.e. $\hat{f}(S) = 0$ for $0 < |S| \leq d$). Then the GF($2$) degree of $f$ is less than or equal to $r - d$.

Why do we care? This means that any junta can either be represented as a real polynomial of low degree, or a GF($2$) polynomial of low degree.

**Proof.**

$$f(x) = \hat{f}(\emptyset) + \sum_{d < |S| \leq r} \hat{f}(S)\chi_S$$

Now define

$$h(x) = f(x) \oplus PAR(x_1, \ldots, x_r)$$
$$= \hat{f}(\emptyset)x_1 \cdots x_n + \sum_{0 \leq |T| < r - d} \hat{f}([r] \setminus T)\chi_T$$

What can we say about $h$? There are two cases:
Claim 2. (Easy) Suppose $\hat{f}(\emptyset) = 0$. Then $h$ is of $GF(2)$ degree $< r - d - 1$.

Consider a new representation of $f$, $f'$, over $\{0, 1\}$, and a corresponding $h'$. We can do this with

$$h'(y_1, \ldots, y_r) = \frac{1}{2} - \frac{h(1 - 2y_i, \ldots, 1 - 2y_r)}{2}$$

Then it is certainly true that $\text{deg}(h') < r - d - 1$ (our transformation from $h$ to $h'$ is degree-preserving). But, because we’re in $GF(2)$, we can use this representation to get a low-degree representation of $f$. Observe that over $GF(2)$, $h' = f' + x_1 + \cdots + x_r$ (this final sum computes the parity function over $GF(2)$). So the degree of $f'$ as a $GF(2)$ polynomial is at most $r - d$.

Claim 3. (Hard) Suppose $\hat{f}(\emptyset) \neq 0$. Then $h$ is still of $GF(2)$ degree $< r - d - 1$.

Now we have to worry about the first term. Notice now that $\text{deg}(h) = r$, and so $\text{deg}(h') = r$. We will show that

$$\text{deg}_{GF(2)}(h'_{GF(2)}) \leq r - d$$

First, we’ll argue that all monomials of deg. $> r - d$ in $h'$ have even coefficients.

In $h'$, the only term with deg. $\geq r - d$ is $\hat{f}(\emptyset)x_1x_2\cdots x_r$. The whole contribution of this term is

$$\frac{-\hat{f}(\emptyset)}{2} \prod_{i=1}^{r}(1 - 2y_i)$$

$$= \frac{-\hat{f}(\emptyset)}{2} \sum_{S \subseteq [r]} (-2)^{|S|} y_s$$

Fix $S' = \{1, 2, \ldots, r - d\}$. What is the coefficient of $y_{S'}$ in $h'$? It’s certainly 0 from the non-$\emptyset$ term, and from above it’s

$$\frac{-\hat{f}(\emptyset)}{2} (-2)^{r-d}$$

from the $\emptyset$ term. We have previously shown that this second quantity is an integer, so the coefficient of any $|S'| \leq r - d$ has an integer coefficient.

Now consider a set $S$ of size $> r - d$ (i.e. $= r - d + t$ for some $t > 0$). The coefficient of $h'$ on $y_s$ is

$$\frac{-\hat{f}(\emptyset)}{2} (-2)^{|S|} = \left( \frac{-\hat{f}(\emptyset)}{2} (-2)^{r-d} \right) 2^t$$
by simple algebra. Because the inside term is an integer (from above) and it’s multiplied
by a power of 2, the whole coefficient must be a multiple of 2.

Now we’re done: the even integers are congruent to 0 in GF(2), so \( h'_{GF(2)} \) has a 0
coefficient for every \(|S| > r - d| \). Consequently \( \deg(h'_{GF(2)}) \leq r - d \).

This completes our analysis for both cases.

3 Learning PAR

We’re going to continue looking at uniform-distribution learning, but with a new twist:
after every oracle query, with probability \( 1 - \eta \), the returned example is mislabeled.
(This is called “Random Classification Noise”, or RCN) How does this affect our ex-
stisting results for uniform distribution learning?

Notice first that the Low-Degree Algorithm remains unaffected. It can be re-
formulated in the statistical query model, which is robust to RCN; alternatively, given
\( f : \{-1, 1 \}^n \to \{-1, 1\} \), let \( f_\eta : \{-1, 1\}^n \to \{-1, 1\} \) be

\[
 f_\eta(x) = \begin{cases} 
 f(x) & \text{with probability } 1 - \eta \\
 -f(x) & \text{with probability } \eta 
\end{cases}
\]

Then

\[
 \hat{f}_\eta = (1 - 2\eta)f(s) \forall S
\]

so we can continue using LDA.

But what about the parity function? \( \{\text{PAR}_S\}_{S \subseteq [n]} \) is a hard class to learn. Think of
our usual system-of-equations approach to finding parities: with \( \eta \) classification noise,
an \( \eta \)-fraction of the equations are wrong on the right hand side, and we have to identify
them in order to successfully solve our system of equations.

[A digression: why do we care about learning noisy parities in the first place?
It turns out \text{PAR} is a hardness primitive in some cryptosystems, and has practical
applications to linear coding theory. Finally, a result by [FGKP] tells us that good
noisy parity learning gives us good algorithms for a host of other problems: junta
learning, decision tree learning and DNF learning!]

3.1 A very naïve approach

Let’s just try to estimate all the Fourier coefficients. If we draw only \( n \) samples, then
with high probability we get a good estimate of all \( 2^n \) FCs. However, the algorithm
will require \( 2^n \) time to try each different coefficient.
3.2 A slightly less naïve approach

As before, let’s draw \( m = O(n) \) samples (in this case, \( n + o(n) \)). With high probability, approximately \( \eta m \) examples are noisy, and \( 1 - \eta m \) examples are uncorrupted. Let’s just guess the \( \eta m \) noisy examples to throw away, and use a standard parity learning algorithm.

To test the various combinations, we’ll need to consider \( \binom{m}{\eta m} \) discards. This is slightly better than before, but not much—it still requires \( 2^{\Theta(1)} m \) time to run.

3.3 A better algorithm [BKW]

**Theorem 4.** Fix \( \eta \in (0, 1/2) \). Then there exists an algorithm (we’ll call it the BKW algorithm) which learns PAR with \( 2^{O(\frac{\log n}{\log \log n})} \) time and examples.

[N.B. One V. Lyubachevsky modified this algorithm to use \( n^{1.01} \) examples, but with runtime \( 2^{\Theta(n/\log \log n)} \).]

High-level intuition: the real algorithm takes two parameters, \( a \) and \( b \), and learns parity using

\[
poly\left(\left(\frac{1}{1 - 2\eta}\right)^{2a}, 2^b\right)
\]

time and examples. Then, choosing \( b = 2n/\log n \) and \( a = 1/2 \log n \), we recover the claimed performance of the algorithm.

Higher-level intuition: we’re just using our same old Gaussian elimination trick, but over a larger field.

Even more intuition: if we can express any given vector \( z \in \{0, 1\}^n \) as a sum of few examples, then we can learn a single bit of the parity efficiently. Then we just repeat this process for all bits. If we have many examples, then most vectors will be expressible as a sum of few.

Some notation: let \( c = (c_1, \ldots, c_n) \in \{0, 1\} \) denote the target parity, with \( c_i = 1 \rightarrow i \) is in the target parity. So \( f(x) = c^T x \mod 2 \). Finally let \( S = (x^1, y^1), \ldots, (x^m, y^m) \) be the set of \( m \) examples from \( EX^n(f) \).

Suppose, somewhat fantastically, that we actually got a basis vector \( e_1 = (1, 0, 0, \cdots) \) in our draw. Then \( y^i \) indicates \( c_i \) with probability \( 1 - \eta \). If, even more fantastically, we were to draw this basis vector multiple times, then the Chernoff bound tells us that

\[
O\left(\frac{\log n}{(1 - 2\eta)^2}\right)
\]
duplicates of the vector would be sufficient to ensure correctness with failure probability $1/n^2$. Then we could repeat this same procedure for $e_2, \ldots, e_n$, after which we would have all of $c$ correct with failure probability $1/n^2$. Sadly, we need $m \approx 2^n$ examples in order to find enough basis vectors.

Now suppose instead that there existed $x^i, x^j$ in the data such that $x^i + x^j = e$. In this case $y^i + y^j = c$, now with probability $1 - 2\eta + 2\eta^2$ (which is certainly greater than $1/2$, so good enough for our purposes). The odds of finding two examples like this are improved—we have $m \approx 2^{n/2}$.

The BKW algorithm, given $S$, expresses $e_1$ as a sum of $2^n$ examples. The sum of their labels equals $c_1$ with advantage $(1/(1 - 2\eta))^{2^n}$. We can repeat for higher confidence, and repeat even more for the remaining $c_i$.

Let’s actually show how to do this.

Lemma 5. After $(x^1, y^1), \ldots, (x^l, y^l)$ are drawn independently from $EX^\eta(f)$, $y^1 + \cdots + y^l = c^T(x^1 + \cdots + x^l)$ with probability $1/2 + 1/2(1 - \eta)^l$.

Proof. We will use induction on $l$.

Base case: $l = 1$; we are correct with probability $1 - \eta$.

Inductive case: suppose the probability guarantee holds for $l - 1$. Then for the $l$th term, our success probability is

$$(1 - \eta)(1/2 + 1/2(1 - 2\eta)^{l-1}) + \eta(1/2 - 1/2(1 - 2\eta)^{l-1})$$

The first term corresponds to the case in which both examples are noiseless, and the second in which both examples are noisy. This sum is equal to

$$1/2 + 1/2(1 - 2\eta)^l$$

which completes our inductive step.

So we want $e_1$ as a sum of $o(n)$ examples.

Choose $a$ and $b$ so that $n = ab$ We will view the $n$ bits of our example as comprising $a$ blocks, each containing $b$ bits.

Now define $V_i = \{x \in \{0, 1\}^n \text{with 0s in all positions of the last } i \text{ blocks}\}$. An $i$-sample of size $s$ is a set of $s$ vectors, each of which is independent and uniformly distributed over $V_i$. If $S = x^1, \ldots, x^m$ are $m$ examples drawn from the noisy oracle, then $S$ is an $i$-sample for $i = 0$. Then the algorithm will successively use $(i-1)$-samples to construct $i$-samples.

Lemma 6. Let $A_i$ be an $i$-sample of size $s$. Then there is an $O(sn)$-time algorithm which, given $A_i$, outputs $A_{i+1}$, an $(i+1)$-sample, with
\begin{itemize}
  \item size $\geq s - 2^b$
  \item each example a sum of exactly two examples from $A_i$
\end{itemize}

\textbf{Proof.} Let $A_i = x^1, \ldots, x^s$. Each $x^i$ has all 0s in blocks $a - i + 1, \ldots, a$

Now partition $A_i$ into $2^b$ subsets, corresponding to partitions of the $i$th block.

Fix some “special vector” in each partition. Add that vector to every other in its
block. Finally take these new vectors, and add them all to $A_{i+1}$. This algorithm has
the desired properties:

1. Certainly $A_{i+1}$ has the right number of points (since we’ve lost at most $2^b$ points).

2. It is also an $(i + 1)$-sample. The lost block is definitely zeroed out (since we
effectively XORed it with itself), but the remaining blocks are still uniformly
random, as fixed shifts (like addition) don’t affect the distribution.