CS 15-859OO: Randomness in Computation Fall 2022

Lecture 7 Part 2 — Sketching Theorem + Intro to Streaming Model

Prof. David Woodruff *Scribe: Puhua Cheng*

1 Proof of Sketching Theorem (continued)

We are investigating $\mathbb{E}\left[Z'_{i,j} \mid E_{i,j}\right]$ where $E_{i,j}$ is the event that $|Z_{i,j}| \leq d^3$, and $Z'_{i,j} = \min\{|Z_{i,j}|, d^3\}$.

clarification We know that R has iid Cauchy rows, so when we multiply to obtain RA, we know the columns of RA are independent. But we do not know if two entries in the same row are independent.

We can compute

$$\begin{aligned}\mathbb{E}\left[Z'_{i,j} \mid E_{i,j}\right] &= \mathbb{E}\left[Z'_{i,j} \mid E_{i,j}, E\right]\mathbf{Pr}\left[E \mid E_{i,j}\right] + \mathbb{E}\left[Z'_{i,j} \mid E_{i,j}, \overline{E}\right]\mathbf{Pr}\left[\overline{E} \mid E_{i,j}\right] \\ &\geq \mathbb{E}\left[Z'_{i,j} \mid E_{i,j}, E\right]\mathbf{Pr}\left[E \mid E_{i,j}\right] \\ &= \mathbb{E}\left[Z'_{i,j} \mid E\right]\mathbf{Pr}\left[E \mid E_{i,j}\right]\end{aligned}$$

Note that by Bayes rule,

$$\mathbf{Pr}\left[E \mid E_{i,j}\right] = \frac{\mathbf{Pr}\left[E_{i,j} \mid E\right]\mathbf{Pr}\left[E\right]}{\mathbf{Pr}\left[E_{i,j}\right]} \geq 1 - \frac{\log d}{d}$$

so we also get that $\mathbb{E}\left[Z'_{i,j} \mid E\right]$ is also in $O(\log d)$.

Note that

$$|RA_{*i}| = |A_{*i}|_1 \frac{\sum_i |Z_{i,j}|}{d \log d}$$

We are interested in the expected value of the LHS. This translates to the expected values of $|Z_{i,j}|$, which we have computed above to be in $O(\log d)$ with small error probability. By Markov's inequality, with constant probability we have $\sum_i |RA_{*i}|_1 = O(\log d)\sum_i |A_{*i}|$.

To summarize where we are, with constant probability the equation holds. Recall that $A_{*,1}$

We showed that a well-conditioned basis exists, but that will give us a poly(d) bound. To obtain a better bound of $d \log d$, we need a nicer basis, called the Auerbach basis. This basis always exists, and satisfies the following properties:

- $\forall x\ |x|_\infty \leq |Ax|_1$
- $\sum_i |A_{*i}|_1 = d$

One can verify that this is indeed a well-conditioned basis.

One thing about the Auerbach basis is that we do not know how to compute them quickly, but this is fine for us as we do not need to compute this — we are only using it for the proof.

With this basis, $\sum |RA_{*i}|_1 = O(\log d)\sum_i |A_{*i}|_1 = O(d \log d)$, so

$$|RAx|_1 \leq \sum_i |RA_{*i}x_i| \leq |x|_\infty \sum_i |RA_{*i}|_i \leq |Ax|_1 O(d \log d)$$

We know from our γ-net that there is some y that is close to Ax with distance at most $\gamma = 1/(d^3 \log d)$.

Recall that for nets in the first lecture, we took repeated differences. But here, when we take R, the norm can grow. So we take a slightly different approach as follows.

$$\begin{aligned}
|RAx|_1 &\geq |Ry|_1 - |R(Ax - y)|_1 \\
&\geq |y|_1 - O(d \log d)|Ax - y|_1 \\
&\geq |y|_1 - O(d \log d)\gamma \\
&\geq |y|_1 - O(1/d^2) \\
&\geq |Ax|_1/2
\end{aligned}$$

with the last inequality due to the fact that $|y|_1 \geq |Ax|_1 - 1/(d^3 \log d) \geq |Ax|_1/2$ which in turn is because $|Ax|_1 \geq |x|_1/d = 1/d$

This concludes our proof of the sketching theorem, so our sketching solution to the ℓ_1 regression problem works.

2 Fast ℓ_1

The most expensive operation is computing RA. That is slow as R is dense. We can speed this up by picking a sparse sketching Matrix. We can do this by using some $R = SC$ instead, where S is a countsketch matrix, and C is a square matrix with Cauchy values on the diagonal. This turns out to work, and now it is sparse! This sketching matrix gives a $\text{poly}(d)$ distortion, (not $d \log d$), but at least it is fast. The overall time will be $\text{nnz}(A) + \text{poly}(d/\varepsilon)$.

3 Fun Fact about Cauchy RVs

Suppose you have iid copies $R_1, \ldots, R_n$ of some RV with mean 0 and variance σ^2. What is the distribution of $\sum R_i/n$? CLT tells us that this looks like a Gaussian $N(0, \sigma^2/n)$. Because of this, people in real life often use a normal distribution to describe things.

Intuitively, the variance is decreasing and the average is approaching its expectation. But note that the two conditions for CLT is important! Cauchy RVs do not have well-defined mean and variance, and the theorem fails for Cauchy RVs — average of iid standard Cauchys is still standard Cauchy.

4 Turnstile Streaming Model

There is some underlying n-dim vector x initialized to all zeros. We write this as o^n. Now we have a long stream of updates $x_i = x_i + \Delta_j$ for $\Delta_j \in \{-M, \ldots, M\}$. Every item in the stream is of the form (i, Δ), telling us which index to change and how much to change. Each change is an integer from $-M$ to M, and we shall assume $M \leq \text{poly}(n)$.

Throughout the stream, x is promised to be in $[-M, M]^n$. All this is to avoid talking about bit-complexity (we assume x can be represented in $\text{poly}(\log n)$ bits).

Our goal is to output an approximation to $f(x)$ with high probability over coin tosses, and use as little space as possible.

4.1 Testing if $x = 0^n$

Consider the simple problem of testing if x is all zeros. How can we test with accuracy over 0.9?

We do not want to store x — that will take poly(n) memory. Our goal is to use $O(\log n)$ bits of space.

Of course, our solution is to sketch. We saw that for any fixed vector x, if S is a CountSketch with $O(\varepsilon^{-2})$ rows, then $|Sx|_2^2 = (1 \pm \varepsilon)|x|_2^2$ with probability at least 0.9. Now we have constant number of rows, each row we need $O(\log(n))$ bits to store Sx, which is desireable. Now when we want to update some index, we just update the bucket that that index is mapped to. More formally, to update at index i with change Δ, we update $Sx = Sx + S_{i1}\Delta$.

We can store the hash and sign function defining S using $O(\log n)$ bits.

This is a randomized algorithm. Is there a deterministic algorithm with no error and $o(n \log n)$ bits of space? It turns out there is no such deterministic algorithm! This follows from some form of compression argument.

> **Theorem.** Any deterministic algorithm requires $\Omega(n \log n)$ bits of space.

Proof. Suppose the first half of the stream corresponds to updates to a vector a in $\{0, \ldots, \text{poly}(n)\}^n$. Let $S(a)$ be the state of the algo after reading the first half of the stream. If $|S(a)| \in o(n \log n)$, there exists $a \neq a'$ with $S(a) = S(a')$. Now the algorithm must output the same answer for $a + b$ and $a' + b$ for b representing the second half of the stream. When $b = -a$ or $-a'$, the algorithm must err, so we cannot achieve 0 error. ■

4.2 Recovering a k-sparse vector

In real life, many vectors are sparse. For concreteness, we are promised that x at the end of the stream has at most k non-zero entries. Here we are considering the case where k is small. Indeed, this is the case in real life as well. For example, if we are doing file backup, only a few files changed, so there are only a few number of nonzero entries.

Can you recover the indices and values of the k non-zero entries with high probability?

Note that just to describe the entries, we need around $k \log n$ bits. So is there a solution with $k\text{poly}(\log n)$ space? And can we do it deterministically?

It turns out that this is one of the few problems that we can do deterministically. How might we do this?

Let A be an $s \times n$ matrix such that any $2k$ columns are linearly independent. In particular, $s \geq 2k$. We shall maintain Ax in the stream. As before, A is a linear map, so we can maintain it.

> **Claim.** From Ax you can recover the subset S of k non-zero entries and their values.

Proof. Suppose there were some x and y each with at most k non-zero entries and $Ax = Ay$. Now $A(x - y) = 0$, and $x - y$ has at most $2k$ non-zero entries, so $A(x - y)$ is a linear combinations of at most $2k$ columns of A. Therefore we must have $x = y$. ■

So the algorithm is deterministic given A. Note that we are only concerned with space for now, not with time, so we shall not discuss how to extract the actual x from Ax.

How do we find such a A? We can take the Vandermonde matrix A with $2k$ rows and n columns, where $A_{i,j} = j^{i-1}$, where indices start from 1. We can verify that any $2k \times 2k$ sub-matrix of A is linearly independent by computing the determinant — there is an explicit formula for that.

What is the problem with this matrix? The problem is that the entries in A are huge. So instead of storing Ax, we shall store $Ax \pmod p$ with prime $p \in \text{poly}(n)$.