Here are the main points of the lecture:

1 Linear Sketches allow us to Handle Updates

1. Want to keep track of some vector $x^t \in \mathbb{R}^n$ that is changing over time. Each time some update Δ_t arrives, and then $x^{t+1} \leftarrow x^t + \Delta_t$.

2. A (linear) sketch is a short fat matrix $S : \mathbb{R}^n \to \mathbb{R}^k$ where think of $k \ll n$. So instead of maintaining $x^t \in \mathbb{R}^n$ explicitly, we maintain the sketch $Sx^t \in \mathbb{R}^k$.

3. The advantage of this linear sketch is that if we know $y^t := Sx^t$, then $y^{t+1} \leftarrow y^t + (S\Delta_t)$. And if we can compute the sketch $S\Delta_t$ of the "update" vector Δ_t quickly and in little space, we're all set. That's what we will do.

2 Maintaining the Euclidean norm of a vector

1. Want to maintain some info so that we can answer question: what is $\|x^t\|^2$? I.e., want to output some estimate Z such taht $Z \in (1 \pm \epsilon)\|x^t\|^2$

2. We maintain two hash functions: one maps coordinates of x into k bins: i.e., $h : [n] \to [k]$, it is a 2-wise independent hash function. The other maps each coordinate into random bits: $\sigma : [n] \to \{-1, 1\}$, which is 4-wise independent hash function.

3. This gives a sketch matrix $S \in \{-1, 1\}^{k \times n}$, where the i^{th} column has a single non-zero entry in the $h(i)^{th}$ row. This entry has value $\sigma(i)$.

drawimageherelater.

4. Now we claim that $(Sx)^2$ is a good estimate of $\|x\|^2$. Why? First, let's consider $\mathrm{E}[(Sx)^2]$.

$$\begin{aligned}
\mathrm{E}[(Sx)^2] &= \mathrm{E}\left[\sum_{j=1}^{k}\left(\sum_{i\in[n]}\delta(h(i)=j)\sigma(i)x_i\right)^2\right] \\
&= \mathrm{E}\left[\sum_{j=1}^{k}\sum_{i,i'\in[n]}\delta(h(i)=j)\cdot\delta(h(i')=j)\cdot\sigma(i)\ x_i\ \sigma(i')\ x_{i'}\right] \\
&= \sum_{j=1}^{k}\sum_{i,i'\in[n]}\mathrm{E}[\delta(h(i)=j)\cdot\delta(h(i')=j)]\cdot\mathrm{E}[\sigma(i)\sigma(i')]\ \ x_i\ x_{i'} \\
&= \sum_{j=1}^{k}\sum_{i,i'\in[n]}\Pr[h(i)=j)\wedge h(i')=j]\cdot\mathrm{E}[\sigma(i)\sigma(i')]\ \ x_i\ x_{i'}
\end{aligned}$$

But $\mathrm{E}[\sigma_i] = 0$, since it takes on value -1 and 1 with equal probability. And since the hash function σ is pairwise independent, the expectation will be zero *except when $i = i'$, when it is* 1! So we get

$$= \sum_{j=1}^{k} \sum_{i \in [n]} \Pr[h(i) = j)] \cdot x_i^2$$

And finally, since $\Pr[h(i) = j] = 1/k$, we get $\sum_{j=1}^{k} \|x\|^2/k = \|x\|^2$. So the expectation of our estimator $(Sx)^2$ is indeed correct!

5. The expectation being correct is fine, what about the variance? In reci we will show that the variance is also controlled, i.e.,

$$\mathrm{Var}((Sx)^2) = 2\|x\|^4/k. \tag{1}$$

This is where we will use the 4-wise independence of the hash function σ, and the pairwise independence of the function h, because the above argument only used the pairwise independence of σ!

6. Now if we know the mean is correct, and the variance is bounded as in (1), we can use Chebyshev's inequality.[1] This says that we are close to the mean with high probability. Recall it says that for a random variable Z with mean μ and variance σ^2,

$$\Pr[|Z - \mu| \geq \lambda] \leq \frac{\sigma^2}{\lambda^2}.$$

(Please don't confuse this variance symbol σ with the hash function σ.) In our case the r.v. is $Z = (Sx)^2$, with mean $\mu = \|x\|^2$, and variance at most $2\|x\|^4/k$, then

$$\Pr[|Z - \mu| \geq \varepsilon\mu] \leq \frac{2\|x\|^4/k}{\varepsilon^2\mu^2} = \frac{2\|x\|^4/k}{\varepsilon^2\|x\|^4} = \frac{2}{k\varepsilon^2}.$$

So if we want a failure probability of 1/10, say, then we can set $k = \frac{20}{\varepsilon^2}$.

7. In summary, the sketch S we defined (using the hash functions h, σ, gives us a $(1 \pm \varepsilon)$ estimate of the squared Euclidean length of a vector with probability at least 9/10, as long as $k \geq 20/\varepsilon^2$.

8. Note that instead of maintaining $x \in \mathbb{R}^n$, we just needed to store the sketch $Sx \in \mathbb{R}^k$, where $k \approx 20/\varepsilon^2$. That's a huge reduction in space!! Also, it's a linear sketch, so we can maintain this sketch as the vector x changes via updates.

3 Finding a Non-zero Coordinate of a Vector

1. Now we show a linear sketch that allows us to find a non-zero coordinate of a vector. Since this is a linear sketch, we can use it to find a non-zero coordinate of the current vector, as the vector changes via updates.

[1] Recall how to prove this too: $\Pr[|Z - \mu| \geq \lambda] = \Pr[(Z - \mu)^2 \geq \lambda^2] \leq \frac{\mathrm{E}[(Z-\mu)^2]}{\lambda^2}$, where the inequality is Markov's inequality. Now the numerator is just the definition of variance σ^2.

3.1 1-Sparse Finder

We start by designing a procedure which, with probability $1 - 1/\text{poly}(n)$, has the following guarantees:

- if $x \in \mathbb{R}^n$ is 1-sparse, i.e., has exactly one non-zero entry x_i, the procedure returns the identity i.
- otherwise, the procedure outputs FAIL.

The algorithm starts by choosing a prime $p = \text{poly}(n)$, which can be deterministically chosen to be say, between n^2 and $2n^2$. We also choose a random integer $z \in \{0, 1, 2, \ldots, p-1\}$.

We compute the three linear sketches:

1. $A = \sum_i x_i$.
2. $B = \sum_i x_i \cdot i$.
3. $C = \sum_i x_i z^i \mod p$.

If we're in a stream, we just maintain A, B, and C in the stream. After processing the stream, we then check if B/A is in the set $\{1, 2, \ldots, n\}$. If it is not, then we immediately know that B/A is not 1-sparse and we can output FAIL.

Otherwise, we check if $C = A \cdot z^{B/A} \mod p$. Notice that if x is 1-sparse with non-zero item i, then $C = x_i z^i \mod p$, while $A = x_i$ and $z^{B/A} = z^i$, and so indeed $C = A \cdot z^{B/A} \mod p$ as desired.

On the other hand, suppose x is not 1-sparse. Then the claim is that we output FAIL with probability at least $1 - 1/\text{poly}(n)$. To see this, define the polynomial $q(y) = \sum_i x_i y^i - A \cdot y^{B/A} \mod p$, where y is a formal variable. Notice that $B/A \in \{1, 2, \ldots, n\}$, as otherwise we would have already output FAIL. Moreover, $q(y)$ cannot be the all zeros polynomial, since $\sum_i x_i y^i$ has at least two non-zero terms, and both cannot cancel with the $A \cdot y^{B/A}$ monomial.

Hence, $q(y)$ is a non-zero polynomial of degree at most n modulo p, and it is well-known that such a polynomial can have at most n roots, i.e., y for which $q(y) = 0$. Since z was a random integer in $\{0, 1, 2, \ldots, p-1\}$, the probability that z is a root of q is at most $\frac{n}{p}$, which since $p \geq n^2$, say, is at most $\frac{1}{n}$. Consequently, with probability $1 - \frac{1}{n}$, the test "does $C = A \cdot z^{B/A} \mod p$?" fails and we output FAIL, as we should in this case.

Notice that it takes $O(\log n)$ bits to store A, B, and C, since each is a number specified with $O(\log n)$ bits. Let us call the above algorithm 1-Sparse-Finder.

3.2 Outputting a Non-Zero of a k-Sparse Vector

We next show how to use 1-Sparse-Finder to find a non-zero item of x when x is not 1-sparse, but rather has at most k non-zero entries. The main idea is to use hashing.

Let h be a 2-universal hash function from $[n]$ to $[10k]$, where we think of the integers $1, 2, 3, \ldots, 10k$ as hash buckets. In the j-th hash bucket, we run 1-Sparse Finder on the set of indices i of x for which $h(i) = j$. If any of the 1-Sparse Finder algorithms outputs an index i, then we output an i returned by one of the 1-Sparse Finder algorithms. Note that more than one 1-Sparse Finder algorithm may output an index i; in that case we just choose to output an index output by an arbitrary one of

these 1-Sparse Finder algorithms. If every single 1-Sparse Finder algorithm outputs FAIL, then we output FAIL.

We now analyze the above algorithm, which we call k-Sparse Finder. Notice that each 1-Sparse Finder algorithm is incorrect with probability at most $1/n$, so by a union bound the probability that any of them is incorrect is at most $(10k)/n$, and we condition on none of these algorithms being incorrect; call this event $\mathcal{F}$, which we condition on. Note that being incorrect is different from the algorithm outputting FAIL; outputting FAIL indicates that the input to the 1-Sparse Finder is not 1-sparse, assuming the algorithm is correct.

The main concern one might have is that all the non-zero items of x collide in a bucket, or more precisely, each of the $10k$ buckets is not 1-sparse. To show this cannot happen, consider an arbitrary non-zero entry i of x, and suppose $h(i) = j$. Now consider some non-zero entry $i' \neq i$. Since h is 2-universal, $\Pr[h(i') = h(i)] \leq \frac{1}{10k}$. By a union bound, the probability that there exists an $i' \neq i$ with $x_{i'} \neq 0$ for which $h(i') = h(i)$ is at most $\frac{k-1}{10k} < \frac{1}{10}$. Hence, with probability at least $9/10$, i is *isolated*, meaning no other non-zero entries of x hash to the same bucket as i. It follows that since we conditioned on $\mathcal{F}$ occurring, 1-Sparse Finder succeeds in the $h(i)$-th bucket, and thus will output i. It follows that we will output a non-zero item of x with probability at least $9/10$, given $\mathcal{F}$ (which in turn holds with probability at least $1 - (10k)/n^2$).

In summary, we have an algorithm which, if x has at most k non-zero items, uses $O(k \log n)$ bits of space and succeeds in outputting an item with probability at least $9/10$. We call this algorithm k-Sparse Finder. When k-Sparse Finder does output an item i though, we are guaranteed (except with probability at most $(10k)/n$) that $x_i \neq 0$.

3.3 Subsampling

The above gives $O(k \log n)$ bits of space, which is great if k is small, but not so good if k is large. The next main idea is to use subsampling, whereby we uniformly sample subsets of coordinates of x in a nested sequence of subsets. More precisely, we start with $S_0 = [n]$. We then form S_1 by independently including each item of S_0 with probability $1/2$, then form S_2 by independently including each item of S_1 with probability $1/2$, etc. We let x_{S_i} denote x restricted to coordinates in S_i; more precisely, x_{S_i} is n-dimensional, just like x, but $(x_{S_i})_j = x_j$ if $j \in S_i$, and $(x_{S_i})_j = 0$ otherwise.

We now analyze the subsampling process. Suppose x has k non-zero entries. What's the expected number of non-zero entries in x_{S_i}? To analyze this, for each non-zero entry j in x, let $Z_j = 1$ if $j \in S_i$, and $Z_i = 0$ otherwise. Let $Z = \sum_j Z_j$ be the total number of non-zero entries of x included in S_i. Then $\mathbf{E}[Z] = k \cdot \mathbf{E}[Z_j] = \frac{k}{2^i}$. Also, since the Z_j are independent, $\mathbf{Var}[Z] = \sum_j \mathbf{Var}[Z_j] = k \cdot \mathbf{Var}[Z_1] = k\left(\frac{1}{2^i}\right)\left(1 - \frac{1}{2^i}\right) \leq \frac{k}{2^i}$.

Now, notice if $i = \lfloor \log_2 k \rfloor - 5$, then $32 \leq \mathbf{E}[Z] < 64$ and $\mathbf{Var}[Z] < 64$. This statement requires $i \geq 0$, but observe that if $i < 0$, then this means $k = O(1)$, and our earlier k-Sparse Finder for constant k would succeed in finding a non-zero entry of x and use only $O(\log n)$ bits of space. So, assuming $i \geq 0$, we in fact have, by Chebyshev's inequality, that

$$\Pr[|Z - \mathbf{E}[Z]| \geq 32] \leq \frac{\mathbf{Var}[Z]}{32^2} \leq \frac{1}{16}.$$

Hence, with probability at least $15/16$, we have that $1 \leq Z < 96$, using that $32 \leq \mathbf{E}[Z] < 64$. It follows that if we run a k'-sparse algorithm with $k' = 96$ on x_{S_i}, we recover a non-zero item of x_{S_i} with probability at least $1 - 1/16 - 1/10 - (10k)/n > 4/5$, where the $1/16$ error probability comes

from the fact that the subsampling might not have between 1 and 96 non-zero items, the $1/10$ error probability comes from no item being isolated in the 96-Sparse Finder algorithm, and the $(10k)/n$ comes from one of the 1-Sparse Finder algorithms failing.

Hence, for the above special choice of i, we succeed with probability at least $4/5$ in outputting a non-zero item of a vector x, but we don't know i in advance, so what can we do?

The idea is to run our k'-Sparse Finder algorithm on every x_{S_i} simultaneously in parallel. Every time we see a stream update $x_j \leftarrow x_j + \Delta_t$ for some coordinate j and increment/decrement amount Δ_t, we feed this stream update into a k'-Sparse Finder algorithm for processing x_{S_ℓ} for each ℓ for which $j \in S_\ell$. Note that we are running $O(\log n)$ k'-Sparse Finder algorithms in parallel, and feeding each stream update to a subset of them.

Now there are $O(\log n)$ different k'-Sparse Finder algorithms, each running $10k'$ instances of 1-Sparse Finder. The probability that any of the 1-Sparse Finder algorithms for any of these algorithms is incorrect is therefore at most $O(\log n) \cdot (10k')/n = O(\log n)/n$, and thus we can condition on all of these being correct. Thus, if there is at least one algorithm processing an x_{S_ℓ} which succeeds in outputting a coordinate j, we output an arbitrary such coordinate. It follows that we succeed in outputting a non-zero coordinate of x with probability at least $1 - 1/16 - 1/10 - O(\log n)/n > 4/5$. Moreover, our total space is $O(\log n) \cdot O(k' \log n) = O(\log^2 n)$ bits.