On Efficient Sketching Algorithms

Praneeth Kacham
Computer Science Department
Carnegie Mellon University
pkacham@cs.cmu.edu

Abstract

Sketching has become a very effective tool to efficiently extract useful information from very large inputs which are presented to an algorithm in various forms such as a turnstile stream or when the data is partitioned arbitrarily among multiple servers. In this proposal, we describe the main paradigms of computation with large datasets and how we obtain new efficient algorithms for a large variety of problems using sketch-based techniques.

1 Introduction

In this era of very large datasets, extracting useful insights efficiently has become challenging. Existing algorithms, many of which assume that data can be arbitrarily accessed and that run in super-linear time have become untenable with growing input sizes. In addition to growing input sizes, another key challenge is the way input to a problem is defined. For example, the input maybe arbitrarily partitioned across multiple servers and we may want to solve a problem on the underlying input with algorithms that are communication efficient. As another example, the input to a problem maybe defined by a stream of items and we may want to solve a problem under the constraint that there is not enough space to store all the items. We study algorithms for various linear algebra problems when the dataset is defined in one of the following three ways:

1. **Classic:** In this setting, we are given access to the full input and want to obtain algorithms that compute solutions as fast as possible. We assume that the input is an $n \times d$ matrix A. The rows of the matrix A are usually called the *data points* and d, the number of columns in matrix A, is called the *dimension* of the dataset. The aim is to obtain algorithms that run in near-linear time in $\mathsf{nnz}(A)$, which denotes the number of non-zero entries in the matrix A.

2. **Streaming:** In this setting, the input to a problem is implicitly defined by a stream of *updates*. Again, we define the input to be an $n \times d$ matrix A. In the row arrival model of streaming, each row of the matrix A is revealed one after another and we assume that the number of rows n is too large to store the full matrix A. We want algorithms that (i) use sublinear space in the number of rows n, (ii) process the rows as they are received, and (iii) compute a solution to a problem on the underlying matrix A at the end of processing the stream.

   In the more general turnstile streaming setting, we assume that the underlying matrix A is implicitly defined by a stream of additive updates to its entries. An update of the form $((i, j), \Delta)$ replaces the (i, j)-th entry $A_{i,j}$ with $A_{i,j} + \Delta$. Here Δ is allowed to be negative as well. We assume that both the number of rows n in the matrix and the number of updates is large and therefore desire algorithms that use sublinear space in n, process each update $((i, j), \Delta)$ as they are received, and compute a solution on the matrix A at the end of the stream.

3. **Distributed:** Here we assume that there are s servers that each store a part of the input. Specifically, the j-th server holds an $n \times d$ matrix $A(j)$ and input to a problem is defined as $A = A(1) + \cdots + A(s)$. We want algorithms in this setting, that require small amount of overall communication, compared to the size of the underlying matrix A, and compute a solution to a problem on input A.

   In the more restrictive row-partition model of distributed computation, we assume that there are s servers with the j-th server holding an $n_j \times d$ matrix $A(j)$ and input to the problem is defined by the matrix A obtained by concatenating the matrices $A(1), \ldots, A(s)$.

Each of the above settings presents different challenges with regard to the types of "efficiency" we care about. We define five complexity measures with one or more of them being relevant to each of the above settings.

- **Time Complexity:** In the classic setting, time complexity simply refers to the total amount of time required by an algorithm to compute a solution to a problem on the given dataset.

  In the streaming setting, we are concerned with the amount of time required by an algorithm to process an *update* to the underlying matrix. We call this the *update time* of a streaming algorithm. Note that having a small update time is crucial to be able to process a stream of updates at a high throughput. Another complexity measure related to time in the streaming setting is the amount of time required for a streaming algorithm to output a solution at the end of processing the stream.

- **Space Complexity:** In the streaming setting, we assume that we do not have enough space to store the underlying matrix A and hence a streaming algorithm operates in a memory-constrained setting. The space complexity of a streaming algorithm is the amount of space (in bits) it needs to process the stream of updates to the underlying matrix and output a solution at the end of the stream. We want algorithms that have as low a space complexity as possible.

- **Randomness Complexity:** Algorithms in all the above settings are often randomized and use a large number of random bits as part of the algorithm which raises the question how these bits are obtained. Thus it is important for an algorithm to use as few random bits as possible while meeting other efficiency measures.

  More importantly though, in the streaming setting, an important question is how the large number of random bits needed by an algorithm are stored since the algorithms operate in a memory-constrained setting. The usual techniques here include replacing full randomness with k-wise independent hash functions for some small value of k or use Pseudorandom Generators (PRGs), that fool small space algorithms, such as Nisan's PRG [Nis90].

- **Query Complexity:** In many settings, specific ways of interacting with the underlying matrix can be much faster and simpler than materializing the entire matrix. Some ways of interacting with a matrix are (i) querying specific entries, or (ii) querying for the result of multiplying the matrix with a specific vector or more generally (iii) applying a function, chosen from a restricted class such as the set of all linear functions, to the matrix and obtaining the result.

  In this setting, an algorithm needs to minimize the amount of queries it makes since it is directly related to the amount of time required to run such an algorithm. Thus given a class of queries that one is allowed, the query complexity of an algorithm is the number of such queries it performs to obtain a solution to the given problem. Again, we want algorithms that perform as few queries as possible.

- **Communication Complexity:** In the distributed setting discussed above, the input to a problem can be arbitrarily partitioned among s servers. Assuming there is a coordinator that can communicate with

all the servers, the communication complexity of a protocol is the number of bits exchanged between the coordinator and the s servers to solve the problem.

In the following sections, we first describe a broad classification of techniques that have been used to obtain fast algorithms to problems in each of the above settings. Then, we discuss some of the algorithms we obtain in each of the "Classic", "Streaming" and "Distributed" settings and how each algorithm is efficient with respect to one or more of the above complexity measures.

2 Classification of Techniques

In the randomized linear algebra literature, there have majorly been two different techniques to obtain efficient algorithms in the above described settings : (i) Linear sketching and (ii) Coresets. We will briefly describe these two methods at a high level. In this proposal, we refer to these techniques jointly as *sketching*.

2.1 Linear Sketching

Linear sketching broadly refers to the technique of applying an *oblivious* randomized linear transformation to decrease the dimensionality of the data or decrease the number of data points or both. The smaller dataset obtained after applying suitable transformations to the original dataset is usually termed as a "linear sketch" (or just a *sketch*). At a high level, a sketching-based algorithm has the following structure: First the algorithm computes a sketch of the entire dataset using an appropriate transformation so that the sketch captures *enough* structure of the dataset that is necessary to solve a problem and then using the sketch, the algorithm computes an (approximate) solution to the given problem on the original dataset.

To obtain fast sketch-based algorithms, it is necessary, as a first step that we can compute a sketch of the dataset quickly. After obtaining the sketch, we need an algorithm that runs quickly on the sketch to obtain a solution for the original problem. A proxy for how fast we can run an algorithm on the sketch is the so-called *size* of the sketch i.e., the output dimension of the randomized transformation. Thus we need to carefully balance both the time-to-sketch and size-of-the-sketch to obtain fast algorithms using this paradigm.

Sarlós [Sar06] first observed that the Fast Johnson Lindenstrauss Transformation of Ailon and Chazelle [AC09] can be used to compute a sketch of an $n \times d$ matrix A in $O(nd \log n)$ time and showed that this sketch can be used to approximately solve problems such as the Frobenius norm Low Rank Approximation of A and the ℓ_2 linear regression problem with A as the coefficient matrix. Later, Clarkson and Woodruff [CW17] gave a construction of a sketch that can be computed in $\mathsf{nnz}(A)$ time and obtained input-sparsity time algorithms for these problems. Thereafter, numerous works have obtained fast algorithms for a variety of problems using the sketch-and-solve paradigm.

Apart from solving problems in the classic setting, linear sketching is a major technique to obtain efficient algorithms in the streaming and distributed settings as well. We note that the linear sketch can be *efficiently*[1] updated when a coordinate of the underlying matrix A gets updated. This observation has lead to turnstile streaming algorithms for a variety of problems such as moment estimation [And17, KNPW11], heavy hitters [CCFC02], sampling coordinates from a variety of distributions [JST11, JW21] and many more.

Linear sketches are useful even in the distributed setting to obtain communication efficient algorithms. Recall that each server in this setting holds a matrix $A(1), \ldots, A(s)$ respectively and if the server j sends a linear sketch of the matrix $A(j)$ to the coordinator, then the coordinator can obtain a sketch for the matrix $A = A(1) + \cdots + A(j)$ by simply *adding* up the sketches. When the *sketches* are small, this leads to communication-efficient algorithms in the coordinator model. This technique was used to solve the low rank approximation problem in the coordinator model [KVW14].

[1] In time independent of the size of the dataset.

2.2 Coresets

Coresets usually denote a weighted subset of the original data points such that solving an appropriately modified version of the original problem on this weighted subset leads to an approximate solution for the original problem. Again, to obtain fast algorithms using coresets, it is important that we are (i) able to compute the coreset quickly and (ii) the size of the coreset is small so that the problem can be solved quickly on the coreset.

In general, coresets are constructed using importance sampling of the datapoints where the *importance* of a point depends on the problem being solved and on how the datapoint interacts with rest of the points in the dataset. Sensitivity [FL11] has emerged as a way of assigning importance to each point in the dataset and sensitivities have been used to compute small coresets for a number of problems such as clustering [VX12], regression [DMM06], etc.

We remark that while the sensitivities have been used to obtain coreset constructions for many problems, it is not always the case that they are the *best* measure of importance. For example, in the case of ℓ_p subspace embeddings, Cohen and Peng [CP15] show that importance sampling of rows using ℓ_p Lewis weights lead to coresets of smaller size as compared to coresets computed using the ℓ_p sensitivities.

Coresets have been used to obtain efficient algorithms for many problems in all of the classic, streaming and distributed settings. In the classic setting, coresets are often used to reduce the size of the dataset to make searching for brute force solutions faster for problems such as clustering (see [CASS21] and references therein).

In the row-arrival model of streaming, coresets for many different problems can be obtained using the "merge-and-reduce" framework [BS80]. Using this framework, any offline coreset construction algorithm can be converted into a streaming coreset construction in the row arrival model with only a space blow up of logarithmic factors in the length of the stream under certain conditions.

In the row-partition model of distributed computation, each server can independently compute a coreset for their datapoints and send it to the coordinator. In many constructions of coresets, simply the union of all the coresets is a coreset for the entire dataset.

3 The Classic Setting

In the classic setting, we describe our results for five different problems: (i) construction of oblivious subspace embeddings (OSE), which have been the source of many fast algorithms, (ii) dimensionality reduction for approximating sum-of-distances, which can be used to approximate the sum of distances of the given n datapoints to *any* k-dimensional object, (iii) a fast approximation algorithm for ridge regression, (iv) an input-sparsity time algorithm for reduced-rank regression with operator norm error and (v) a lower bound on the query complexity of low rank approximation algorithms.

3.1 Fast Oblivious Subspace Embeddings

A random matrix $\boldsymbol{S}$ with n columns is called an (α, δ) Oblivious Subspace Embedding (OSE) for d-dimensional subspaces of $\mathbb{R}^n$ if

$$\Pr_{\boldsymbol{S}}[\text{for all } x \in W, \|x\|_2 \leq \|\boldsymbol{S}x\|_2 \leq \alpha\|x\|_2] \geq 1 - \delta$$

for all k-dimensional subspaces $W \subseteq \mathbb{R}^n$. When δ is a small constant, we refer to $\boldsymbol{S}$ as an α-OSE for simplicity. We call α to be the distortion of the subspace embedding $\boldsymbol{S}$. Instantiating W to be the column space of a given

$n \times d$ matrix A, we obtain that with probability $\geq 1 - \delta$ over the random matrix $\boldsymbol{S}$, for all $x \in \mathbb{R}^d$,

$$\|Ax\|_2 \leq \|\boldsymbol{S}Ax\|_2 \leq \alpha\|Ax\|_2.$$

The concept of subspace embeddings was introduced by Sarlós [Sar06] who showed that it can be used to obtain fast algorithms for linear regression and low rank approximation, when $n \gg d$.

We can show, using a net argument, that if the entries of $\boldsymbol{S}$ are independent Gaussian random variables and if $\boldsymbol{S}$ has $m = O(d)$ rows, then $\boldsymbol{S}$ is an α-OSE for a constant α. But many applications of OSEs to linear algebra require that we are able to compute the matrix $\boldsymbol{S} \cdot A$. If $\boldsymbol{S}$ is a dense Gaussian matrix, then unfortunately, it requires $\min(nd^{\omega-1}, \mathsf{nnz}(A) \cdot d)$ time to compute the matrix $\boldsymbol{S} \cdot A$, where ω is the matrix multiplication constant, which is quite slow.

To remedy this problem, Sarlós used a structured Johsnon-Lindenstrauss Transform of Ailon and Chazelle [AC09] and showed that there is a construction of an α-OSE $\boldsymbol{S}$ with $m = O(d \log d)$ rows for a constant α and that given any $n \times d$ matrix, we can compute $\boldsymbol{S} \cdot A$ in $O(nd \log n)$ time, which can be much faster than multiplying A with an arbitrary dense matrix with m rows.

Later, Tropp [Tro11] simplified this construction and gave tighter bounds on the number of rows m required to obtain an OSE. But a major disadvantage of this line of work is that we cannot make use of the sparsity of matrix A. In many real-world datasets, each row of the matrix A, which corresponds to a datapoint, is quite sparse and the matrix A has much fewer than $n \cdot d$ nonzero entries. It is often the case that n and d themselves are so large that even algorithms that require $\Theta(n \cdot d)$ time are too slow and hence algorithms using sketches from this line of work can be impractical in those cases.

Clarkson and Woodruff [CW17] gave the first construction of an OSE $\boldsymbol{S}$ which can make use of the sparsity of A to compute the matrix $\boldsymbol{S} \cdot A$ faster than in $O(nd \log n)$ time. Their construction of an α-OSE $\boldsymbol{S}$ for a constant α has $m = O(d^2)$ rows and has an additional property that *each* column of $\boldsymbol{S}$ has exactly one non-zero coordinate. Thus the matrix $\boldsymbol{S} \cdot A$ can be computed in $O(\mathsf{nnz}(A))$ time with the caveat that the OSE has much larger number of rows than the dense Gaussian construction described above which has only $m = O(d)$ rows.

Nelson and Nguyên [NN13] later generalized this construction and showed that for any parameter $\gamma > 1$, there is an α-OSE $\boldsymbol{S}$ for a constant α, with $m = O(d^{1+\gamma} \log(d))$ rows and that $\boldsymbol{S} \cdot A$ can be computed in $O(\gamma^{-1}\mathsf{nnz}(A))$ time. A combination of these constructions can be used to obtain an α-OSE $\boldsymbol{S}$ with $m = O(d \cdot \mathrm{polylog}(d))$ rows, for a constant α, such that for any $\gamma > 0$, the matrix $\boldsymbol{S} \cdot A$ can be computed in time

$$O(\gamma^{-1}\mathsf{nnz}(A) + d^{2+\gamma}\,\mathrm{polylog}(d)).$$

This was the status quo before our work [CCKW22]. There was no construction of a subspace embedding that can be applied in $O(\gamma^{-1}\mathsf{nnz}(A) + d^{2+\gamma}\,\mathrm{polylog}(d))$ time and with $m = o(d \log d)$ rows. In our work, we gave the *first* construction of an OSE with $m = O(d \cdot \mathrm{poly}(\log\log d))$ rows and a distortion $\alpha = \exp(\mathrm{poly}(\log\log d))$ such that the matrix $\boldsymbol{S} \cdot A$ can be computed in

$$O(\gamma^{-1}\mathsf{nnz}(A) + d^{2+\gamma}\,\mathrm{polylog}(d))$$

time. This led to the first algorithms for many problems, such as basis finding, that run in time $O(\mathsf{nnz}(A) + d^{\omega}(\mathrm{poly}(\log\log d)))$ making steps towards completely removing the $\mathrm{poly}(\log d)$ multiplicative factor to the d^{ω} term plaguing many results before our work. We will briefly summarize our main result in the following theorem.

Theorem 3.1. *Given a parameter $\gamma > 0$, there is an $\exp(\mathrm{poly}(\log\log d))$-OSE $\boldsymbol{S}$ with $m = O(d \cdot \mathrm{poly}(\log\log d))$ rows such that given an $n \times d$ matrix A, we can compute the matrix $\boldsymbol{S} \cdot A$ in*

$$O(\gamma^{-1}\mathsf{nnz}(A) + d^{2+\gamma}\,\mathrm{poly}(\log d))$$

time.

Application	Time Complexity
$(1+\varepsilon)$ Subspace Embeddings	$\gamma^{-1}\mathsf{nnz}(A) + d^{\omega}\,\mathrm{poly}(\log\log d) + d^{2+o(1)}(\varepsilon^{-3} + \varepsilon^{-2}n^{\gamma+o(1)})$
$(1+\varepsilon)$-approximate Linear Regression	$\gamma^{-1}\mathsf{nnz}(A) + d^{\omega}\,\mathrm{poly}(\log\log d) + \varepsilon^{-2}n^{\gamma+o(1)}d^{2+o(1)}$
Independent Row Selection (rank k)	$O(\mathsf{nnz}(A) + k^{\omega}\,\mathrm{poly}(\log\log k))$

Table 1: Applications of the Oblivious Subspace Embedding Construction

We use this subspace embedding construction to obtain algorithms for various problems. We list our results in Table 1. Our crucial observation which led to this result is that when all the *unit* vectors in a subspace have large ℓ_1 norms, then a very sparse sign matrix satisfies the OSE property. We then use a deterministic embedding construction of Indyk [Ind07] to show that any d dimensional subspace of $\mathbb{R}^{O(d\,\mathrm{polylog}(d))}$ can be linearly mapped to such a flat subspace without blowing up the dimension by a lot.

This result has been recently improved in [CDDR23] which obtained the first construction of a sparse subspace embedding with $O(d)$ rows.

3.2 Dimensionality Reduction for approximating Sum-of-Distances

Consider the following simple problem: Given n points $x_1, \ldots, x_n \in \mathbb{R}^d$ and a shape $S \subseteq \mathbb{R}^d$, we want to approximate

$$\sum_i d(x_i, S)$$

where $d(x, S)$ is defined as $\inf_{y\in S} \|x - y\|_2$. When S is a set of k points, the above quantity is just the k-median cost with S as the centers and when S is a k-dimensional subspace, then the quantity is the "robust" subspace approximation cost. For this problem, Sohler and Woodruff [SW18] gave a dimensionality reduction by producing a subspace P of $\mathrm{poly}(k/\varepsilon)$ dimensions such that for any $S \subseteq \mathbb{R}^d$ with $\dim(\mathrm{span}(S)) \le k$, then

$$\sum_{i\in[n]} \sqrt{d(x_i, P)^2 + d(\mathbb{P}_P x_i, S)^2} = (1 \pm \varepsilon) \sum_{i\in[n]} d(x_i, S)$$

where $\mathbb{P}_P$ denotes the projection matrix on to subspace P. The above relation shows that one only needs the projections of x_is onto this special subspace P and the distance from x_i to P to be able to approximate the sum-of-distances to an arbitrary k-dimensional shape. Thus, we only need to store $n \cdot \mathrm{poly}(k/\varepsilon)$ parameters instead of the $n \cdot d$ parameters required to store the exact values of x_is, to be able to approximate the sum-of-distances. Unfortunately, the construction in [SW18] takes time exponential in k/ε. We [FKW21] give an improved algorithm that computes such a subspace P in time $O(\mathsf{nnz}(A)/\varepsilon^2 + (n+d)\mathrm{poly}(k/\varepsilon))$ and the reduced dimension points can be used to approximate the sum of distances of the original n points to any k dimensional shape. Our algorithm is iterative and in each iteration solves a subspace approximation problem using linear sketching techniques to reduce the *size* of the problem.

The dimensionality reduction procedure also lets us compute small coresets for many "sum-of-distances" problems. We summarize our main result in the following theorem.

Theorem 3.2. *Given an $n \times d$ matrix A, a rank parameter k and an accuracy parameter ε, there is an iterative algorithm that runs in time $O(\mathsf{nnz}(A)/\varepsilon^2 + (n+d)\cdot\mathrm{poly}(k/\varepsilon))$ and outputs a subspace P of dimension at most $\mathrm{poly}(k/\varepsilon)$ such that with probability $\ge 9/10$, for any shape S with $\dim(\mathrm{span}(S)) \le k$,*

$$\sum_{i\in[n]} \sqrt{d(A_{i,*}, P)^2 + d(\mathbb{P}_P \cdot A_{i,*}, S)^2} = (1 \pm \varepsilon) \sum_{i\in[n]} d(A_{i,*}, S),$$

where $A_{i,*} \in \mathbb{R}^d$ denotes the i-th row of the matrix A and $\mathbb{P}_P$ is the orthogonal projection matrix onto the subspace P.

3.3 Ridge Regression

Given a matrix $A \in \mathbb{R}^{n\times d}$ and a vector $b \in \mathbb{R}^d$, the ridge regression problem is defined as:

$$\min_x \|Ax - b\|_2^2 + \lambda\|x\|_2^2$$

where $\lambda > 0$ is a *regularization* parameter. When $n \geq d$, then we say we are in the *over-determined* case and when $n < d$, we say we are in the *under-determined* case. We study fast algorithms for computing a vector $\tilde{x} \in \mathbb{R}^d$ that satisfies

$$\|A\tilde{x} - b\|_2^2 + \lambda\|\tilde{x}\|_2^2 \leq (1+\varepsilon)\min_x (\|Ax - b\|_2^2 + \lambda\|x\|_2^2).$$

In the ridge regression objective above, as we increase the value of λ, the *importance* of the design matrix decreases, as in, when λ is large enough, setting $x = 0$ without considering A and b would be a good solution. Thus the "effective dimension" of the problem decreases with increasing values of λ. To capture this phenomenon, we define statistical dimension sd_λ as

$$\mathsf{sd}_\lambda := \sum_{i=1}^{\min(n,d)} \frac{\sigma_i^2}{\sigma_i^2 + \lambda}$$

where σ_i is the i-th singular value of the matrix A. In general, we would like algorithms for solving ridge regression to have small running times when sd_λ is small since we already know that $x = 0$ is a *good* solution.

3.3.1 Fast Algorithms in the Under-determined Case

We [KW22] study algorithms for Ridge Regression in the underdetermined case. Earlier algorithms [CYD18] used Oblivious Subspace Embeddings with distortion $1+\varepsilon$ to obtain a solution with the above guarantee. We show that a sketch with only the weaker ε-Approximate Matrix Multiplication (AMM) guarantee and an Oblivious Subspace Embedding guarantee with a constant distortion is sufficient to solve the ridge regression problem to a $1+\varepsilon$ factor.

Definition 3.3 (AMM)**.** A sketch $\boldsymbol{S}$ with n columns has the (ε, δ)-AMM property if given any two matrices A and B each with n rows,

$$\Pr_{\boldsymbol{S}}[\|(\boldsymbol{S}A)^\mathsf{T}(\boldsymbol{S}B) - A^\mathsf{T}B\|_\mathsf{F} \leq \varepsilon\|A\|_\mathsf{F}\|B\|_\mathsf{F}] \geq 1-\delta.$$

Our observation that only constant distortion OSEs which additionally satisfy AMM guarantees is sufficient to solve ridge regression leads to faster algorithms in some parameter regimes compared to [CYD18]. In this work, we also give a tight lower bound on sizes of the sketches that have AMM guarantee.

Theorem 3.4 (AMM lowerbound)**.** *Given an accuracy parameter ε and an integer n, any randomized sketching matrix $\boldsymbol{S}$ with n columns that satisfies*

$$\Pr_{\boldsymbol{S}}[\|(\boldsymbol{S}A)^\mathsf{T}\boldsymbol{S}B - A^\mathsf{T}B\|_\mathsf{F} \leq \varepsilon\|A\|_\mathsf{F}\|B\|_\mathsf{F}] \geq 9/10,$$

for all matrices A, B with n rows, must have $m = \Omega(\min(n, 1/\varepsilon^2))$ rows.

We note that the above lower bound is tight up to constant factors since the CountSketch matrix with $m = O(1/\varepsilon^2)$ rows satisfies the ε-AMM guarantee.

3.3.2 Optimal Deterministic Coresets

In [KW20], we study the problem of constructing coresets for the ridge regression problem. Specifically, given an instance of the ridge regression problem (A, b, λ), we want to compute a set $S \subseteq [n]$ and weights w_i for $i \in S$ such that the solution to the problem

$$\min_x \sum_{i \in S} w_i (\langle A_{i,*}, x \rangle - b)^2 + \lambda \|x\|_2^2$$

is a $1+\varepsilon$ approximation to the ridge regression problem. We show that a subset S with size $|S| = O(\mathsf{sd}_\lambda/\varepsilon)$ that satisfies the desired coreset property can be computed using a deterministic algorithm of [CNW15]. We additionally show that this bound cannot be improved in terms of statistical dimension by exhibiting an instance for every ε and $\lambda \leq O(1/\varepsilon)$ which requires that any such coreset have $\Omega(\mathsf{sd}_\lambda/\varepsilon)$ rows.

Using our deterministic coreset, we also give communication efficient algorithms for ridge regression in the row-partition model of distributed computation.

3.4 Reduced-Rank Regression with Operator Norm Error

In the same vein as above sketch-based algorithms, we obtain a fast algorithm for approximately solving

$$\min_{\text{rank-}k\ X} \|AX - B\|_2.$$

Note that the usual multi-response regression problem measures the error in terms of the Frobenius norm and has no restriction on the rank of the coefficient matrix. In the version of the problem we study, we put a rank restriction to increase interpretability of the coefficient matrix and also use operator norm of the residual matrix to measure the error which in some cases can capture more structure than Frobenius norm.

These new restrictions combined with a lack of "Pythagorean Theorem" for operator norm makes the problem harder to solve. We do not even know of a closed form solution to this problem. We use an existential criterion of [SR12] for there to exist a solution to the problem with $\|AX - B\|_2 \leq \beta$ for a given β and then show that a rank-k approximation for a specific matrix in spectral norm gives a solution to the reduced-rank regression problem.

To obtain a fast algorithm, we also show that the Block-Krylov iteration algorithm of [MM15] works even when the matrix-vector products have a certain amount of error. We summarize our result in the following theorem:

Theorem 3.5. *Given an $n \times d$ matrix A, an $n \times d'$ matrix B, an accuracy parameter ε and a rank parameter k, there is an algorithm that runs in time*

$$O\left(\left(\frac{\mathsf{nnz}(A) \cdot k}{\varepsilon^{3/2}} + \frac{\mathsf{nnz}(B) \cdot k}{\varepsilon} + \frac{d^2 k}{\varepsilon^{3/2}}\right) \cdot \mathrm{polylog}(\kappa(B), n, d, 1/\varepsilon) + d^\omega\right)$$

where $\kappa(B) = \sigma_1(B)/\sigma_{k+1}(B)$ is the rank-k condition number of the matrix B.

3.5 Query Complexity of Low Rank Approximation

As described in the introduction, very often it is much more efficient to interact with the underlying matrix in specific ways than it is to assume arbitrary access to the entries of the matrix. For examples, suppose we are given access to an $n \times n$ matrix A. Given a vector x and integer $q \geq 1$, we can compute $A^q x$ using q adaptive matrix-vector products whereas computing the entire matrix A^q is inefficient. Thus given the matrix A, we have "efficient" matrix-vector product query access to A^q. Matrix-vector products model a specific set of

linear functions on matrices. We can indeed allow for more general linear measurements of matrices and study the query complexity in those models.

Candés and Plan [CP11] study the number of *noisy* linear measurements required to extract an underlying $n \times n$ low rank matrix A. Note that a rank k matrix $n \times n$ matrix A can be described using only $O(nk)$ parameters. They give algorithms with query complexity that increases with the noise level in the measurements. At a certain large enough noise level, they show that their algorithm requires $\Omega(n^2)$ linear measurements which amounts to essentially reading all the entries of the matrix A and hence could be inefficient to implement. The algorithms they study are non-adaptive, as in, they pre-specify all the linear measurements up front and reconstruct the matrix using the received responses. Hence a natural question is if adaptivity helps us reconstruct the matrix with fewer number of linear measurements.

Indeed, we can show that using the power method, we can reconstruct the matrix using a total of $\tilde{O}(n)$ linear measurements and $O(\log n)$ rounds of adaptivity. But are $\Omega(\log n)$ rounds of measurements required to avoid essentially reading the whole matrix? Unfortunately, this turns out to be (almost) true. In [KW23], we showed that any algorithm that runs for fewer than $o(\log n / \log \log n)$ adaptive rounds, must use a total of $\Omega(n^2)$ linear measurements and therefore essentially has to read all the entries of the matrix.

Our lower bound is proved using Bayes risk lower bounds. We define a distribution over $n \times n$ matrices such that any *deterministic* algorithm which uses $n^{2-\beta}$ linear measurements in each round does not learn enough information in $o(\log n / \log \log n)$ rounds to output a good rank-k approximation. By Yao's lemma, it then follows that there is no randomized algorithm that outputs a low rank approximation for every input. Our input distribution is simply $\boldsymbol{G} + (\alpha/\sqrt{n}) \sum_{i=1}^{k} \boldsymbol{u}_i \boldsymbol{v}_i^{\mathsf{T}}$ where $\boldsymbol{G}$ is an $n \times n$ matrix with independent Gaussian entries, $\boldsymbol{u}_i, \boldsymbol{v}_i$ for $i = 1, \ldots, k$ are n-dimensional vectors with independent Gaussian entries as well, and α is a large enough constant.

Using this result we also show that general linear measurements are not more powerful than matrix-vector products for problems such as low rank approximation, eigenvalue approximation etc.

3.5.1 Next Steps

An important open problem I'd like to study over the next few months is the lower bound on query complexity to obtain a spectral norm low rank approximation of the matrix A when we are able to compute arbitrary matrix-vector products. Concretely, let A be an $n \times d$ matrix and that we can query an arbitrary $v \in \mathbb{R}^d$ or $u \in \mathbb{R}^n$ and receive Av or $u^{\mathsf{T}}A$ respectively. Given a rank parameter k, that question I'd like to study is a lower bound on the number of adaptive queries necessary to output a rank-k matrix B such that

$$\|A - B\|_2 \leq (1 + \varepsilon) \min_{\text{rank-}k\, X} \|A - X\|_2.$$

We can show that $\min_{\text{rank-}k\, X} \|A - X\|_2 = \sigma_{k+1}(A)$, the $(k+1)$-th singular value of A. The Block Krylov iteration algorithm of Musco and Musco [MM15] and the LazySVD algorithm of Allen-Zhu and Li [AZL16] both require $O(k \operatorname{polylog} n/\sqrt{\varepsilon})$ adaptive matrix-vector products and the main question is if $\Omega(k/\sqrt{\varepsilon})$ matrix-vector products are required. Simchowitz et al., [SEAR18] show that for a related problem, $\Omega(k/\sqrt{\varepsilon})$ matrix-vector products are *necessary* but their proof does not directly carry over to the spectral norm Low Rank Approximation. Recently, Bakshi and Narayanan [BN23] showed that $\Omega(1/\sqrt{\varepsilon})$ matrix-vector products are necessary to output a rank-1 approximation of matrix A but do not have a lower bound for arbitrary k. Hence we ask:

"Are $\Omega(k/\sqrt{\varepsilon})$ matrix-vector products necessary to output a $1+\varepsilon$ approximate rank-k approximation to the matrix A in Spectral Norm?"

4 The Streaming Setting

4.1 Turnstile Streaming

As we mentioned in the introduction, in the turnstile streaming model, the underlying matrix A receives updates of the form $((i, j), \Delta)$ and on receiving such an update, we update the (i, j)-th entry as $A_{i,j} \leftarrow A_{i,j} + \Delta$. The goal of a streaming algorithm is to process the stream of updates in as small space as possible and at the end of the stream output a solution to a problem on the underlying matrix.

There has been a lot of work studying optimal space bounds for various problems when the underlying object is an n-dimensional vector x receiving updates of the form (i, Δ). For the problem of approximating $F_k(x) = \sum_{i=1}^{n} |x_i|^k$, up to constant factors, an $\Omega(n^{1-2/k})$ bits lower bound was shown by [BYJKS04] for $k > 2$ and a matching upper bound (up to polylogarithmic factors) was given by [IW05]. Much later, Andoni [And17] also gave a simple linear sketch based algorithm to approximate $F_k(x)$ using $\tilde{O}(n^{1-2/k})$ bits of space.

When $k \leq 2$, a series of works ending with [KNW10] give a turnstile streaming algorithm that uses $O(\varepsilon^{-2} \log n)$ bits to approximate $F_k(x)$ up to $1 \pm \varepsilon$ factors. They also show that $\Omega(\varepsilon^{-2} \log n)$ bits are necessary thus completely resolving the space complexity of estimating $F_k(x)$ for $k \leq 2$.

While the space complexity of estimating $F_k(x)$ in a stream has been resolved, unfortunately the *update time* of these space efficient algorithms is quite large. Andoni's algorithm for estimating $F_k(x)$ for $k > 2$ uses a pseudorandom generator of Nisan and Zuckerman [NZ96] to achieve a space complexity of $O(n^{1-2/k} \operatorname{poly}(\log n))$ bits and therefore has an update time of $\operatorname{poly}(n)$ which makes the algorithm impractical. For $k \leq 2$, the algorithm of [KNW10] has an update time of $\operatorname{poly}(1/\varepsilon)$ in the WordRAM model. A later work [KNPW11] gave a new algorithm that still uses the optimal $O(\varepsilon^{-2} \log n)$ bits but has an improved update time of $O(\log^2 1/\varepsilon \log\log 1/\varepsilon)$ in the WordRAM model with a word size $O(\log n)$.

As we noted before, algorithms that have a small update time are necessary to be able to handle high-throughput streams. Thus the main question is to obtain space-optimal algorithms that also have a small update time.

To explain why the update time of these algorithms is large, we will first give the simple recipe a large number of streaming algorithms follow. First the streaming algorithms define a randomized sketching matrix $\boldsymbol{S}$ with d columns. Each of the columns of $\boldsymbol{S}$ is drawn *independently* from an appropriate distribution. The streaming algorithm essentially maintains the value of $y = \boldsymbol{S}x$ when the vector x is being updated in the stream as follows: when the vector x gets an update (i, Δ), the streaming algorithm retrieves $\boldsymbol{S}_{*,i}$, the i-th column of $\boldsymbol{S}$ and updates $y \leftarrow y + \boldsymbol{S}_{*,i} \cdot \Delta$. Note that this will ensure that at the end of processing the stream $y = \boldsymbol{S}x$ where x is the final value of the underlying vector. Then the algorithm uses the vector y to output a solution to the problem it is trying to solve.

In the above recipe, the streaming algorithm needs to be able to retrieve any column of the matrix $\boldsymbol{S}$. As the columns are all independently sampled, the algorithms thus require $\Omega(n)$ bits of space. There have majorly been two ways to address this problem: (i) Instead of sampling the columns of $\boldsymbol{S}$ independently, use a t-wise independent hash function, for an appropriate value of t, to define the matrix $\boldsymbol{S}$ or (ii) Use a pseudorandom generator (PRG) that fools small-space algorithms to define the matrix $\boldsymbol{S}$ and only store the "seed" for the pseudorandom generator so that the any column of $\boldsymbol{S}$ can be generated on demand. [KNW10, KNPW11] use the first route whereas [And17] uses the second route, which was first suggested by Indyk [Ind06]. When the amount of "independence", t, required is large, the hash functions drawn from t-wise independent hash families are slow to evaluate. Similarly, when we are trying to "fool" algorithms using Pseudorandom generators, either the hash function have to be evaluated on large inputs or many hash functions have to be evaluated sequentially.

In our work [KPTW23], we generalize the construction of Nisan's PRG [Nis90] and give a new pseudorandom generator which we call HashPRG that has a space-vs-time trade-off, using which we can obtain fast update

times by using more space. Our construction additionally has a symmetry property that lets us derandomize the guarantees Minton and Price [MP14] give for a fully random CountSketch data structure. Using HashPRG and opening up the analysis of the algorithms of Andoni [And17], we obtain a constant factor F_k approximation algorithm for $k > 2$ that uses $\tilde{O}(n^{1-2/k})$ bits of space and has an $O(1)$ update time in the Word RAM model.

Similarly, for $k < 2$ and $\varepsilon < 1/n^c$, we obtain an algorithm based on [KNPW11] to approximate $F_k(x)$ up to a $1 \pm \varepsilon$ factor using the optimal $O(\varepsilon^{-2} \log n)$ bits of space and an update time of $O(\log n)$ in the Word RAM model. We also obtain algorithms with fast update times for other problems such as for estimating $\|x\|_\infty$ in this paper. We summarize some of our results in the following theorem:

Theorem 4.1. *Suppose that a vector $x \in \mathbb{R}^n$ initialized to 0 receives the updates $(i_1, \Delta_1), \ldots, (i_t, \Delta_t)$ and suppose that the number of updates $t \leq \text{poly}(n)$ and $|\Delta_j| \leq \text{poly}(n)$ for all j. On a WordRAM machine with a word size of $O(\log n)$, the following results hold:*

1. *For $k > 2$, there is a turnstile streaming algorithm to approximate $F_k(x)$ up to a constant using $\tilde{O}(n^{1-2/k})$ bits. The algorithm has an update time of $O(1)$.*
2. *For $k < 2$ and $\varepsilon < 1/d^c$ for a small enough constant c, there is a turnstile streaming algorithm that uses the optimal $O(\varepsilon^{-2} \log n)$ bits of space. The algorithm has an update time of $O(\log n)$.*
3. *There is a turnstile streaming algorithm that uses the optimal $O(\varepsilon^{-2} \log 1/\varepsilon \log d)$ bits of space and outputs an additive $\varepsilon\|x\|_2$ approximation to $\|x\|_\infty = \max_i |x|_i$. The algorithm has an $O(\log 1/\varepsilon)$ update time.*
4. *There is a turnstile data structure that uses $O(tr \log n + \log^2 n)$ bits of space and a deterministic algorithm that given any $i \in [n]$ at the end of the stream, uses the state of the data structure, to obtain an estimate $\hat{x}_i$ such that for all i,*

$$\Pr[|x_i - \hat{x}_i| > \alpha\|x\|_2/\sqrt{t}] \leq 2\exp(-\alpha^2 r) + 1/\text{poly}(n).$$

The algorithm has an update time of $O(r \log n)$.

4.2 Row Arrival Model

Recall that in the row-arrival model of streaming, we obtain rows of the matrix A one after the other. Using a small amount of space, we want to solve some problem on the underlying matrix A. In [EKM+23b], we give fast streaming algorithms to compute coresets for the ℓ_∞ subspace approximation problem defined as

$$\min_{\text{dim-}k \text{ subspaces } V} \max_i d(A_{i,*}, V),$$

which essentially asks us to find the k-dimensional subspace V that minimizes the maximum distance from the points in the matrix A. Our streaming coreset construction selects a subset $S \subseteq [n]$ of size $|S| = \text{poly}(k, \log n)$ such that for all k-dimensional subspaces V,

$$\frac{\max_{i \in [n]} d(A_{i,*}, V)}{\text{poly}(k, \log n)} \leq \max_{i \in S} d(A_{i,*}, V) \leq \max_{i \in [n]} d(A_{i,*}, V).$$

Our coreset construction is *online*, as in whenever a new row arrives, the algorithm chooses to keep the row or discard it based only on the current value of the coreset. Thus, the algorithm requires only $d \cdot \text{poly}(k, \log n)$ bits of space to store the rows in the coreset. The above guarantee then implies that an approximate solution to the ℓ_∞ subspace approximation problem on the rows $(A_{i,*})_{i \in S}$ is also an approximate solution to the ℓ_∞ subspace approximation problem on the entire matrix A.

We show applications of our coreset construction for many problem such as ℓ_p subspace approximation, width estimation, volume estimation, etc. Our work is the first to construct such coresets for general values of k, whereas the previous works studied streaming algorithms only for the specific values such as $k = 0$ and $k = 1$ [AS15].

4.3 Next Steps: Streaming PCA

In the row-arrival model, a problem I am planning to study is the space complexity of approximating the top singular vector of the matrix A. Suppose that we receive the rows $a_1, \ldots, a_n \in \mathbb{R}^d$ of the matrix A and we want to approximate the top right singular vector of A. We can simply maintain the $d \times d$ matrix $\sum_i a_i a_i^\intercal$ in the stream using $\tilde{O}(d^2)$ bits of space and then the top eigenvector of the PSD matrix $\sum_i a_i a_i^\intercal$ is then the top right singular vector of the matrix A.

Thus, the main question is can we approximate the top eigenvector using $o(d^2)$ bits of space. In particular, what can we do using $\tilde{O}(d)$ bits of space i.e., the space enough to store only $\tilde{O}(1)$ rows of the matrix A. Price [Pri22] showed that when the spectral gap $\sigma_1(A)/\sigma_2(A) \leq O(1)$, then any streaming algorithm must use $\tilde{\Omega}(d^2)$ bits of space to approximate the top singular vector whereas when $\sigma_1(A)/\sigma_2(A) \geq C\sqrt{\log n \log d}$, then there is a streaming algorithm that uses $\tilde{O}(d)$ bits of space. We ask:

"Are there reasonable assumptions on the matrix A or the arrival order that break these lower bounds?"

One assumption that seems to be necessary to obtain low space streaming algorithms is "low coherence" i.e., no particular row is important. If a particular row is very important to approximate the top eigenvector, then a small-space streaming algorithm may *miss* that row. Over the next few months, we would like to answer the above question by obtain algorithms that work under reasonable assumptions on the input matrix.

5 The Distributed Setting

Consider the arbitrary partition model. In this setting, there are s servers all connected to a coordinator. The j-th server holds a matrix $A(j) \in \mathbb{R}^{n \times d}$ and the coordinator wants to solve a problem on the matrix $A = A(1) + \cdots + A(s)$. In each round of communication, each of the servers can send a message to the coordinator and based on all the messages received the coordinator can send a possibly distinct message to each of the servers. A protocol can use multiple such rounds of communication to solve a problem. The amount of communication of a protocol is the total number of bits of communication sent/received by the coordinator.

We [EKM+23a] study the problem of moment estimation and more generally arbitrary function sum approximation problem. In this problem, the j-th server holds a non-negative vector $x(j) \in \mathbb{R}^n$ and the coordinator wants to approximate $\sum_i f(x_i)$ where x_i is the i-th coordinate of the vector $x = x(1) + \cdots + x(s)$ and f is an arbitrary nonnegative function. If $f(y) = y^k$, then the coordinator simply wants to approximate the F_k moment of vector x. We give a two round protocol for this problem when the function f is super-additive and some other properties. To capture the communication complexity of the problem, we define a parameter $c_f[s]$ which is the smallest value that satisfies

$$f(y_1 + \cdots + y_s) \leq \frac{c_f[s]}{s}(\sqrt{f(y_1)} + \cdots + \sqrt{f(y_s)})^2 \text{ for all } y_1, \ldots, y_s \geq 0.$$

Our two round protocol uses a total of $O(c_f[s] \cdot \text{polylog}(n)/\varepsilon^2)$ bits of communication (excluding the dependence on other function specific parameters independent of the number of servers s) and at the end of the protocol, the coordinator computes a $1+\varepsilon$ approximation to $\sum_{i=1}^n f(x_i)$. Specifically, for estimating $\sum_i x_i^k$

for $k > 2$, we can show that $c_f[s] = s^{k-1}$ and therefore our algorithm uses a total of $O(s^{k-1}\,\text{polylog}(n)/\varepsilon^2)$ bits of communication. This matches the $\Omega(s^{k-1}/\varepsilon^2)$ lower bound of [WZ12]. We additionally show a lower bound of $\Omega(s^{k-1}/\varepsilon^k)$ bits on the amount of communication that any one round protocol must use therefore showing that our protocol achieves the best communication bounds using minimum possible rounds.

For general functions f satisfying certain properties, we also show an $\Omega(c_f[s]/\varepsilon^2)$ lower bound on the amount of communication thus showing that our protocol achieves near-optimal communication bounds.

Our protocol crucially uses the "max-stability" property of exponential random variables and we show that using similar techniques, we can sample from "additively-defined" distributions using a small amount of communication.

References

[AC09] Nir Ailon and Bernard Chazelle. The fast johnson–lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on computing*, 39(1):302–322, 2009. 2.1, 3.1

[And17] Alexandr Andoni. High frequency moments via max-stability. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017. 2.1, 4.1

[AS15] Pankaj K Agarwal and R Sharathkumar. Streaming algorithms for extent problems in high dimensions. *Algorithmica*, 72(1):83–98, 2015. 4.2

[AZL16] Zeyuan Allen-Zhu and Yuanzhi Li. Lazysvd: Even faster svd decomposition yet without agonizing pain. *Advances in neural information processing systems*, 29, 2016. 3.5.1

[BN23] Ainesh Bakshi and Shyam Narayanan. Krylov methods are (nearly) optimal for low-rank approximation. *arXiv preprint arXiv:2304.03191*, 2023. 3.5.1

[BS80] Jon Louis Bentley and James B Saxe. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980. 2.2

[BYJKS04] Ziv Bar-Yossef, Thathachar S Jayram, Ravi Kumar, and D Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004. 4.1

[CASS21] Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. A new coreset framework for clustering. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 169–182, 2021. 2.2

[CCFC02] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*. Springer, 2002. 2.1

[CCKW22] Nadiia Chepurko, Kenneth L Clarkson, Praneeth Kacham, and David P Woodruff. Near-optimal algorithms for linear algebra in the current matrix multiplication time. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3043–3068. SIAM, 2022. 3.1

[CDDR23] Shabarish Chenakkod, Michał Dereziński, Xiaoyu Dong, and Mark Rudelson. Optimal embedding dimension for sparse subspace embeddings. *arXiv preprint arXiv:2311.10680*, 2023. 3.1

[CNW15] Michael B Cohen, Jelani Nelson, and David P Woodruff. Optimal approximate matrix product in terms of stable rank. *arXiv preprint arXiv:1507.02268*, 2015. 3.3.2

[CP11] Emmanuel J Candés and Yaniv Plan. Tight oracle inequalities for low-rank matrix recovery from a minimal number of noisy random measurements. *IEEE Transactions on Information Theory*, 57(4):2342–2359, 2011. 3.5

[CP15] Michael B Cohen and Richard Peng. Lp row sampling by lewis weights. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 183–192, 2015. 2.2

[CW17] Kenneth L Clarkson and David P Woodruff. Low-rank approximation and regression in input sparsity time. *Journal of the ACM (JACM)*, 63(6), 2017. 2.1, 3.1

[CYD18] Agniva Chowdhury, Jiasen Yang, and Petros Drineas. An iterative, sketching-based framework for ridge regression. In *International conference on machine learning*, pages 989–998. PMLR, 2018. 3.3.1, 3.3.1

[DMM06] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Subspace sampling and relative-error matrix approximation: column-row-based methods. In *Algorithms—ESA 2006*, volume 4168 of *Lecture Notes in Comput. Sci.*, pages 304–314. Springer, Berlin, 2006. 2.2

[EKM+23a] Hossein Esfandiari, Praneeth Kacham, Vahab Mirrokni, David Woodruff, and Peilin Zhong. Optimal communication bounds for classic functions in the coordinator model and beyond. Under review at STOC, 2023. 5

[EKM+23b] Hossein Esfandiari, Praneeth Kacham, Vahab Mirrokni, David Woodruff, and Peilin Zhong. Space-efficient algorithms for high-dimensional geometric streaming for almost low rank data. Under review at ICLR, 2023. 4.2

[FKW21] Zhili Feng, Praneeth Kacham, and David Woodruff. Dimensionality reduction for the sum-of-distances metric. In *International conference on machine learning*, pages 3220–3229. PMLR, 2021. 3.2

[FL11] Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 569–578, 2011. 2.2

[Ind06] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)*, 53(3):307–323, 2006. 4.1

[Ind07] Piotr Indyk. Uncertainty principles, extractors, and explicit embeddings of ℓ_2 into ℓ_1. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 615–620, 2007. 3.1

[IW05] Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 202–208, 2005. 4.1

[JST11] Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 49–58, 2011. 2.1

[JW21] Rajesh Jayaram and David Woodruff. Perfect l_p sampling in a data stream. *SIAM Journal on Computing*, 50(2), 2021. 2.1

[KNPW11] Daniel M Kane, Jelani Nelson, Ely Porat, and David P Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, 2011. 2.1, 4.1

[KNW10] Daniel M Kane, Jelani Nelson, and David P Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1161–1178. SIAM, 2010. 4.1

[KPTW23] Praneeth Kacham, Rasmus Pagh, Mikkel Thorup, and David P Woodruff. Pseudorandom hashing for space-bounded computation with applications in streaming. *arXiv preprint arXiv:2304.06853*, 2023. FOCS. 4.1

[KVW14] Ravi Kannan, Santosh Vempala, and David Woodruff. Principal component analysis and higher correlations for distributed data. In *Conference on Learning Theory*, pages 1040–1057. PMLR, 2014. 2.1

[KW20] Praneeth Kacham and David Woodruff. Optimal deterministic coresets for ridge regression. In *International Conference on Artificial Intelligence and Statistics*, pages 4141–4150. PMLR, 2020. 3.3.2

[KW22] Praneeth Kacham and David Woodruff. Sketching algorithms and lower bounds for ridge regression. In *International Conference on Machine Learning*, pages 10539–10556. PMLR, 2022. 3.3.1

[KW23] Praneeth Kacham and David Woodruff. Lower bounds on adaptive sensing for matrix recovery. NeurIPS, 2023. 3.5

[MM15] Cameron Musco and Christopher Musco. Randomized block krylov methods for stronger and faster approximate singular value decomposition. *Advances in neural information processing systems*, 28, 2015. 3.4, 3.5.1

[MP14] Gregory T Minton and Eric Price. Improved concentration bounds for count-sketch. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 669–686. SIAM, 2014. 4.1

[Nis90] Noam Nisan. Pseudorandom generators for space-bounded computations. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 204–212, 1990. 1, 4.1

[NN13] Jelani Nelson and Huy L Nguyên. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *2013 ieee 54th annual symposium on foundations of computer science*, pages 117–126. IEEE, 2013. 3.1

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996. 4.1

[Pri22] Eric Price. Spectral guarantees for adversarial streaming pca. Unpublished, 2022. 4.3

[Sar06] Tamás Sarlós. Improved approximation algorithms for large matrices via random projections. In *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*. IEEE, 2006. 2.1, 3.1

[SEAR18] Max Simchowitz, Ahmed El Alaoui, and Benjamin Recht. Tight query complexity lower bounds for pca via finite sample deformed wigner law. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1249–1259, 2018. 3.5.1

[SR12] Kin Cheong Sou and Anders Rantzer. On generalized matrix approximation problem in the spectral norm. *Linear algebra and its applications*, 436(7):2331–2341, 2012. 3.4

[SW18] Christian Sohler and David P Woodruff. Strong coresets for k-median and subspace approximation: Goodbye dimension. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 802–813. IEEE, 2018. 3.2

[Tro11] Joel A Tropp. Improved analysis of the subsampled randomized hadamard transform. *Advances in Adaptive Data Analysis*, 3(01n02):115–126, 2011. 3.1

[VX12] Kasturi Varadarajan and Xin Xiao. On the sensitivity of shape fitting problems. *arXiv preprint arXiv:1209.4893*, 2012. 2.2

[WZ12] David P Woodruff and Qin Zhang. Tight bounds for distributed functional monitoring. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 941–960, 2012. 5