Sketching Transformed Matrices with Applications to Natural Language Processing

Yingyu Liang
University of Wisconsin-Madison

Zhao Song
IAS

Mengdi Wang
Princeton University

Lin F. Yang
UCLA

Xin Yang
University of Washington

Abstract

Suppose we are given a large matrix $A = (a_{i,j})$ that cannot be stored in memory but is in a disk or is presented in a data stream. However, we need to compute a matrix decomposition of the entry-wisely transformed matrix, $f(A) := (f(a_{i,j}))$ for some function f. Is it possible to do it in a space efficient way? Many machine learning applications indeed need to deal with such large transformed matrices, for example word embedding method in NLP needs to work with the pointwise mutual information (PMI) matrix, while the entrywise transformation makes it difficult to apply known linear algebraic tools. Existing approaches for this problem either need to store the whole matrix and perform the entry-wise transformation afterwards, which is space consuming or infeasible, or need to redesign the learning method, which is application specific and requires substantial remodeling.

In this paper, we first propose a space-efficient sketching algorithm for computing the product of a given small matrix with the transformed matrix. It works for a general family of transformations with provable small error bounds and thus can be used as a primitive in downstream learning tasks. We then apply this primitive to a concrete application: low-rank approximation. We show that our approach obtains small error and is efficient in both space and time. We complement our theoretical results with experiments on synthetic and real data.

Proceedings of the 23[rd]International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108.

1 Introduction

Matrix datasets are ubiquitous in machine learning. However, many matrix datasets are usually too large to fit in the computer memory in large scale applications, e.g., image clustering [PPP06], natural language processing [MSA+11], network analysis [MS04, GL16], and recommendation systems [KBV09]. Many techniques have been proposed to perform the learning tasks on these data in an efficient way; see, e.g., [Mah11, Woo14, ZWSP08, GNHS11] and the references therein. However, challenges arise when the learning task is performed on an entrywise transformation of the matrix, which prevents applying many linear algebraic techniques. Furthermore, due to large sizes, these matrices are often constructed by entrywise updates, i.e., the entries of the matrix are constructed from a stream of updates where each update adds some value on some entry. More specifically, there is a very large underlying matrix A (that cannot be stored in memory easily) whose entries are constructed by a data stream where each item in the stream is of the form (i, j, Δ) with $\Delta \in \{\pm 1\}$ representing the update $A_{i,j} \leftarrow A_{i,j} + \Delta$. The downstream learning task (e.g., low rank approximation), however, needs to take input as matrix M where $M_{i,j} = f(A_{i,j})$ for some transformation function f (e.g., $f(x) = \log(|x| + 1)$).

A concrete example is word embedding in natural language processing (NLP). Word embedding methods aim to embed each word to a vector space. It becomes a basic building block in many modern NLP systems. Many of these systems achieve the state of the art performance on various tasks via word embedding [PSM14, MSC+13, WSC+16]. A basic routine in word embedding is to explicitly or implicitly perform

low rank approximation of an entry-wise transformed matrix [LG14, LZM15]. For instance, the transformation is to apply a log likelihood function on each entry. The matrix itself is the so-called co-occurrence count matrix, which can be constructed by scanning the text corpus, e.g., the entire Wikipedia database. This matrix is usually of size millions by millions.

Similar examples include regressions on huge accumulated datasets in economics [DVF13, Var14], where different transformations on covariates are often used to reduce biases. Other examples include visual feature extraction [BPL10], kernel methods [RR08], and M-estimators [Zha97]. These large scale applications make it impractical or hard to implement existing methods, which keep the matrix in memory. Some other approaches exploit the problem structure to get around the huge space requirement. For instance, some of them propose sequential models of the data, and design online algorithms for computing the embeddings (e.g.,[MSC$^+$13, BGJM16]). These methods, however, are more task-specific and cannot be applied to other tasks involving more general entrywise matrix transformations.

In this paper, we show that learning based on transformed large matrices is possible even when storing such a matrix is not feasible. Our main contributions are:

- For a general class of transformation function f, we provide an efficient one-pass *matrix-product sketch* for computing the product of a given small matrix B with the transformed matrix $f(A)$ with provable error bounds. This algorithm uses space at most the size of the output. The method assumes no statistical model about the updates and can handle a general family of transformations. In particular, these transformations include logarithmic functions and small degree polynomials. This method can also be used as building blocks for downstream tasks: any algorithm requires access to the transformed matrix via a matrix product can apply our algorithm to obtain space saving.
- We demonstrate the application of our algorithm in a concrete task: low rank approximation. To the best of our knowledge, our algorithm is the first one that is able to compute low rank approximation of large matrices under entrywise transformations. We plug in our matrix product sketch into known algorithms as black boxes. We provide theoretical analysis on the tradeoff between the space and the accuracy of these algorithms. We show that our algorithms are space efficient and almost match the accuracy of using the full matrix. These theoretical guarantees are complemented by experiments for low rank approximation on synthetic and real data. The empirical results show that our algorithm can reduce the space usage by orders of magnitude while the error is almost the same as the optimum. We show that our algorithms beat the baseline of using uniform sampling on columns of the transformed matrix by a large margin. We also provide results on linear regression in the appendix.

Road Map. We provide definitions and basic concepts in Section 3. In Section 4, we introduce our basic routine called the *matrix product* sketch. We use our sketching algorithms to compute the low rank approximation of a transformed matrix in Section 5, and the application on linear regression is in Appendix E. In Section 6, we use numeric experiments to justify our approach. The appendix provides a list of related works, the complete proofs, details of the experiments, and also additional theoretical and empirical results.

2 Related Work

There exists a large body of work on fast algorithms for large scale matrices. Some are based on randomized matrix algorithms and use techniques like sampling and sketching; see [Mah11, Woo14] and the reference therein. Some others are based on optimization algorithms like Alternating Least Square and Stochastic Gradient Descent and their variants; see [ZWSP08, GNHS11] for some examples. However, most existing approaches do not apply to the settings considered in this paper. The closest work is [WZ16], which considers low rank approximation of the element-wise transformation of the sum of several matrices located in different machines. This distributed setting is different from our setting and naively applying their algorithm will lead to a large space cost. Furthermore, our sketching method can be applied to learning tasks beyond low rank approximation.

Our work is built on techniques from numerical linear algebra and streaming data analysis in the recent decade. There are numerous research works along this line. Here we list a few but far from exhaustive.

Low-rank approximation or matrix factorization of a matrix is an important task in numerical linear algebra. In this problem, we are given a $n \times d$ matrix A and a parameter k, the goal is to find a rank-k matrix $\widehat{A}$ so as to minimize the residual error $\|\widehat{A} - A\|_F^2$, where the Frobenius norm is defined as $\|A\|_F = (\sum_{i=1}^n \sum_{j=1}^d A_{i,j}^2)^{\frac{1}{2}}$. Note that an optimal $\widehat{A}$ provides a good estimation to the leading eigenspace of the matrix A. Classical way of

speeding up low-rank approximation via sketching requires showing two properties for sketching matrix: subspace embedding [Sar06, LWW20, WW19] and approximate matrix product [NN13, KN14]. Low-rank approximation algorithm via combining those two properties has been presented in several papers [CW13, MM13, SWZ19b]. The classical sketching idea is easy to be made a streaming algorithm, since we usually use linear sketching matrix, which we don't need to explicitly write down during the stream. However none of these methods are applicable to our setting, which is much harder than the classical streaming low-rank approximation problem. This is mainly because the transformation f that acts on an the matrix A completely destroyes the linear algebraic property of matrix A; see Appendix D for some discussions. The storage of A can also be indefeasibly large to be stored and apply the above mentioned methods.

Streaming algorithms have gained great progress since its first systematic study by [AMS99]. Classic streaming problems ask how to estimate a function over a vector, which is under streaming updates. For instance, [AMS99] approximates $\|v\|_p$ while observing a sequence of updates to the coordinates of v. The usual assumption is that $v \in \mathbb{R}^n$ and n is so large that v cannot be stored in memory easily. Since [AMS99], a line of research works (e.g. [Ind00, IW05, BYKS02, BKSV14, KNW10]) gradually improve the algorithm and obtain nearly optimal upper and lower bounds. Very recently, [BO10b, BO10a, BVWY17] attempts to handle a more general set of functions. [BVWY17] gives a nearly optimal characterization of this problem. [BBC+17] studies a more general setting, i.e., functions that do not have a summation structure $f : \mathbb{R}^n \to \mathbb{R}$. They give optimal characterization for streaming all symmetric norms. Given theses advances, none of them solves our problem directly since a streaming estimation only gives a value of vector, that is unrelated to the matrix formulation of the input.

3 Preliminaries

Notation. $[n]$ denotes the set $\{1, 2, \cdots, n\}$. For a vector $x \in \mathbb{R}^n$, $|x| \in \mathbb{R}^n$ denotes a vector whose i-th entry is $|x_i|$. For a matrix $A \in \mathbb{R}^{n\times n}$, let $\|A\|$ denote its spectral norm, $\sigma_i(A)$ to denote its i-th largest singular value, and $[A]_k$ denote its best rank-k approximation. Also let $\det(A)$ denote its determinant when A is square. For a function f, $M = f(A)$ means entrywise transformation $M_{ij} = f(A_{ij})$. We also denote A_{i*} as the i-th row of matrix A and A_{*j} as its j-th column.

Problem Definition. The problem of interests is defined as follows. Suppose we have a underlying large matrix $A = (A_{i,j}) \in \mathbb{R}^{n\times n}$ initialized as a zero matrix.[1] Now, we have observed a sequence of updates of the form $\langle (i_1, j_1, \Delta_1), (i_2, j_2, \Delta_2), \ldots, (i_m, j_m, \Delta_m)\rangle$ for some $m = \text{poly}(n)$, $i_t, j_t \in [n]$ and $\Delta_t \in \{-1, 1\}$. At the t-th update, we are updating the underlying matrix by $a_{i_t,j_t} \leftarrow a_{i_t,j_t} + \Delta_t$. We assume that m is bounded by $\text{poly}(n)$. Note that the assumptions of integer updates is without loss of generality. For instance, if the updates is not an integer, we can round them to a specified precision $\epsilon > 0$ and then scale them to integers. The polynomially bounded length is also a usual and reasonable assumption. At the end of the stream, one would like to perform some learning task (such as low-rank approximation) on the matrix $M = f(A)$ for some fixed function $f : \mathbb{R} \to \mathbb{R}$ and would like to do so using as small space as possible, in particular, avoid storing the large matrix A. Some examples of the transformation functions are

$$f(x) = \log(|x| + 1), \text{ or } f(x) = |x|^\alpha, \ \forall \alpha \geq 0. \quad (1)$$

Functions of this form are important in machine learning. For example, $f(x) = \log(|x| + 1)$ corresponds to the log likelihood function and $f(x) = |x|^\alpha$ corresponds to a general family of statistic models or feature expansion.

In this paper we would like to design a space efficient method for approximating $Z = f(A)B$ for a given matrix B, where $f(A) \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times k}$ for some integer n and k with $k \ll n$. We would like to design algorithms that uses space $\widetilde{O}(nk)$ instead of $\widetilde{O}(n^2)$. This can then be used as a plug-in primitive and turn learning algorithms into space efficient ones if they only access $f(A)$ by matrix product with small B. More formally,

Problem 3.1 (approximate transformed matrix and matrix product)**.** *Given a fixed matrix B and function $f : \mathbb{R} \to \mathbb{R}$, design an algorithm that makes a single pass over an update stream of a matrix A, output an approximated value of $f(A)B$ with high probability. We require the algorithm to use as small space as possible (without counting the space of B).*

We call our method the *sketch for f-matrix product*. We then demonstrate its effectiveness in the applications of linear regression and low rank approximation on $M = f(A)$. Linear regression is to minimize $\|Mx - b\|_2^2$, and low rank approximation is defined as follows.

Problem 3.2 (low-rank approximation)**.** *Given integers $k \leq n$, an $n \times n$ matrix M, two parameters*

[1] Our method also applies to non-square A; we consider square matrices for simplicity.

$\epsilon, \delta > 0$, the goal is to output an orthonormal $n \times k$ matrix L such that

$$\|LL^\top M - M\|_F^2 \leq (1+\epsilon)\|M - [M]_k\|_F^2 + \delta.$$

where $[M]_k = \arg\min_{\text{rank}-k\ M'} \|M - M'\|_F^2$.

4 Sketch for f-Matrix Product

Our goal in this section is to compute the matrix product $f(A)B$ where B is given and A is under updating or can only be read entry by entry. We observe that each entry of $Z = f(A)B$ can be written as a vector product: $Z_{i,j} = \langle f(A)_{i*}, B_{*j} \rangle$. Thus, we will first design a primitive to compute each $Z_{i,j}$ using small space. Running a primitive in parallel for each entry $Z_{i,j}$ results in our full algorithm for computing the matrix product. In the following sections, we will first introduce the vector sketch problem and present our vector product primitives for different functions f. Lastly, we will combine them to form a unified algorithm for matrix product.

4.1 Sketch for f-Vector Product

Recall that for given vectors $x, y \in \mathbb{R}^n$, the inner product is defined as $\langle x, y \rangle = \sum_{i=1}^n x_i y_i$. In our setting, we are also given a function $f : \mathbb{R} \to \mathbb{R}$ and a vector $x \in \mathbb{R}^n$ where the storage of x is free, but not directly given y. The *f-vector* product is defined as $\langle x, f(y) \rangle$, where f is applied to y coordinate-wisely. The updates to y is a stream, i.e., we observe a sequence of integer pairs (z_t, Δ_t) for $t = 1, 2, \ldots, m$, where each $z_t \in [n]$ and $\Delta_t \in \{-1, 1\}$. Thus, we initialize y as a $y^{(0)} \leftarrow 0$, a zero-vector, and at time t, the update to y is described by $y^{(t)} \leftarrow y^{(t-1)} + \Delta_{z_t} \cdot e_{z_t}$ where e_{z_t} is the standard unit vector with only the z_t-th coordinate non-zero. Our goal is to approximate $\langle x, f(y) \rangle$ without storing y, where x is given to the algorithm without storage cost. Formally, we define the following problem.

Problem 4.1 (approximate transformed vector and vector inner product)**.** *Given a fixed vector x and function $f : \mathbb{R} \to \mathbb{R}$, design an algorithm that makes a single pass over an update stream of a vector y, output an approximated value of $\langle f(y), x \rangle$ with high probability. We require the algorithm to use as small space as possible (excluding the space of x).*

We note that a naive algorithm would be storing the vector y as a whole. However such an algorithm is not feasible when n is large or the demand of computing such inner products is too high (e.g., in our matrix applications for computing $Z = f(A)B \in \mathbb{R}^{n \times k}$, each entry of Z is an inner product. If each inner product requires space n, then final space can be $O(n^2 k)$ which is prohibitively high.). In Section 4.2 below, we design an algorithm that accomplish this task for function $f(y) = \log(|y| + 1)$, which only uses $\widetilde{O}(1)$ bits of memory. In Section B.3, we present a general framework that works for a general family of functions f with nearly optimal space complexity.

4.2 Sketch $\log(|\cdot| + 1)$-Vector Product

Algorithm 1

data structure LogSum ▷ Theorem 4.2
procedure Initialize(x)
  $\gamma \leftarrow \epsilon^{-2} \operatorname{poly}(\log n/\delta)$
  $t \leftarrow \Theta(\log n)$, $p_j \leftarrow 2^{-j} \cdot \gamma, \forall j \in [t]$
  for $j = 1 \to t$ **do**
    Sample a $\log n$-wise independent hash function $h_j : [n] \to \{0, 1\}$ such that $\forall i \in [n] : \Pr[h_j(i) = 1] = \min(p_j, 1)$.
    Sample a K-set structure KSet_j with error parameter $\Theta(\delta/t)$ and memory budget $\epsilon^{-2} \operatorname{poly}(\log n/\delta)$
  end for
end procedure
procedure Update(a) ▷ $a \in [n]$
  for $j = 1 \to t$ **do**
    if $h_j(a) = 1$ and $x_a \neq 0$ **then**
      KSet_j.update(a)
    end if
  end for
end procedure
procedure Query()
  Pick the largest j such that KSet_j does not return "Fail"
  Let v be the output of KSet_j, denote $S_j = \text{supp}(v)$
  return $2^j \sum_{i \in S_j} x_i \log(|v_i| + 1)$
end procedure
end data structure

Recall that, when $f(\cdot) = \log(|\cdot| + 1)$, we are designing an algorithm for computing the inner product $\langle \log(|y| + 1), x \rangle$, where $x, y \in \mathbb{R}^n$ are two vectors, x is given to the algorithm for free and y is under updating. Our full algorithm is Algorithm 1, which is composed of 3 sub-procedures: procedure Initialize is called on initialization with given vector x, procedure Update is called when we go over the update stream of the vector y, and procedure Query is called at the end to report the answer. The detailed analysis of Algorithm 1, can be found in Appendix B. We here sketch the high level ideas for how it works. For ease of representation, we consider x has no zero coordinates, since otherwise we can simply ignore these coordinates and change our universe $[n]$ to $\text{supp}(x)$ accordingly. Our algorithm is originated from [BO10b] but it is much

simplified in this paper. From a high level, our algorithm can be viewed as an ℓ_0-sampler, namely, sample uniformly at random from the support of an updating vector y. Note that the support of y is changing over time. Thus it is non-trivial to maintain a uniform sample while using only small space. We also note that it is necessary to sample coordinates from the support of y, since otherwise we can always construct worst-case examples for algorithms that sample coordinates uniformly from $[n]$.

We design our algorithm thus by maintaining independently $\Theta(\log n)$ many sub-vectors of the vector y. Each sub-vector is generated by sampling a set of coordinates uniformly from $[n]$ with geometrically decreasing probabilities. For instance, in our algorithm, we first generate $\Theta(\log n)$ many hash functions, each defines a set $S_j \subset [n]$. For each $i \in [n]$, we demand that $i \in S_j$ with probability 2^{-j}. Thus if the size of the support of y is of order $\Theta(2^j)$, then we are expected to sample $\Theta(1)$ samples of y using the set S_j. We now describe how to maintain these sampled coordinates in memory. For convinience we assume $\gamma = 1$ in line 3 in Algorithm 1.

For the case of insertion-only stream (once a coordinate of y becomes larger than 0, it stays so), maintaining the sub-vector y_{S_j} is a trivial task since the number of coordinates of y_{S_j} is expected to be $O(1)$. However, for $j' \leq j$, the sub-vectors $y_{S_{j'}}$s contain too many coordinates. We handle this quite straightforwardly: if any of them exceeds our memory budget, we just ignore them. For the case of general stream, in which coordinates can be 0 even they were non-zero at some time-point. We will be using the K-set data structure presented in [Gan07]. This data structure supports insertion and deletion of data points and can maintain the samples only if the number of final samples is under the memory budget. The formal guarantee of the K-set data structure presented in Theorem B.1.

Suppose now we have collected sufficiently many samples from the support of the vector y. Suppose the set of samples is collected using set S_j. We can have an empirical estimator for the inner product as $2^j \sum_{i \in S_j} x_i \log(|y_i| + 1)$. Notice that this estimator is unbiased. Also since the variance of the estimator is bounded by

$$\begin{aligned} &\sum_i 2^j x_i^2 \log^2(|y_i| + 1) \\ = &O(1) \cdot \|x\|_\infty^2 \cdot \sum_i \log^2(|y_i| + 1) \cdot \log^2 m, \end{aligned}$$

where m is the length of the stream and is usually assumed to be of oder $\mathrm{poly}(n)$, thus we only need $\mathrm{poly}\log n$ samples to obtain an accurate estimation.

We summarize the main guarantee in the following theorem, while the formal proof can be found in Section B.

Theorem 4.2 (approximate inner product of transformed vector and vector)**.** *Suppose vector $x \in \mathbb{R}^n$ is given without memory cost. There exists a streaming algorithm (data structure* LogSum *in Algorithm 1) that makes a single pass over the stream updates to a vector $y \in \mathbb{R}^n$ and outputs $Z \in \mathbb{R}$, such that, with probability at least $1 - \delta$,*

$$|Z - \langle x, \log(|y| + 1)\rangle| \leq \epsilon \cdot \|x\|_\infty \cdot \sum_{i=1}^n \log(|y_i| + 1).$$

The algorithm uses space $O(\epsilon^{-2} \mathrm{poly}(\log(n/\delta)))$ (excluding the space of x) has a $\mathrm{poly}(\log n, 1/\epsilon)$ query time.

Remark 4.3. *We also note that our algorithm naturally works for $f(y) := \log^c(|y| + 1)$ for any constant c. To modify our algorithm, we only need to keep slightly larger space and change the final estimation to be $2^j \sum_{i \in S_j} x_i \log^c(|v_i| + 1)$. It also enjoys the same relative error guarantee in Theorem 4.2.*

4.3 From Vector Product Sketch to Matrix Product Sketch

With the f-inner product sketch tools established, we are now ready to present the result for sketching the matrix product, $Z = f(A)B$. Notice that each entry $Z_{i,j} := \langle f(A_i), B_j \rangle$ is an inner product.

Thus our algorithm for the matrix sketch is simply maintaining an f-inner product sketch for each $Z_{i,j}$. In our algorithm, we assume that matrix B is given to the algorithm for free. Thus, if $B \in \mathbb{R}^{n \times k}$ for some $k \ll n$, we only need to keep up to $\widetilde{O}(nk)$ vector product sketches, which cost in total $\widetilde{O}(nk)$ words of space. For the ease of representation, we present our guarantee for matrix product for $f(z) := \log^c(|z| + 1)$ for some c or for $f(z) = z^p$ for $0 \leq p \leq 2$, and for matrix $B \in \{-1, 0, 1\}^{n \times k}$. Our results can be generalized to a more general set of functions and matrix B using the results presented in Section B.3. The proof of the following theorem is a straightforward application of Theorem 4.2 and B.2.

Theorem 4.4 (approximate each coordinate of the transformed matrix)**.** *Given a matrix $B \in \{-1, 0, 1\}^{n \times k}$, and a function $f(x) := \log^c(|x| + 1)$ for some c or $f(x) := |x|^p$ for some $0 \leq p \leq 2$, then there exists a one-pass streaming algorithm that makes a single pass over the stream updates to an underlying matrix $A \in \mathbb{R}^n$ and outputs a matrix $\widehat{Z}$, such that, with probability at least $1 - \delta$, for all i, j,*

$$|\widehat{Z}_{i,j} - Z_{i,j}| \leq \epsilon \sum_{j'=1}^n f(|A_{i,j'}|).$$

The algorithm uses space $\epsilon^{-2}nk\,\mathrm{poly}(\log(n/\delta))$ *and has an* $nk\,\mathrm{poly}(\log n, 1/\epsilon)$ *query time.*

Remark 4.5. *We note that our sketch in the last theorem can be easily used to approximate the* 2*-norm of each row of the matrix* $f(A)$*. In this case, we simply choose* $B \in \mathbb{R}^{n\times 1}$ *as the all-*1 *vector and change* $f(\cdot)$ *to be* $f^2(\cdot)$*. For* $f(x) = \mathrm{poly}\log(|x|+1)$ *or* $f(x) = |x|^p$ *with* $0 \le p \le 1$*, it can be easily verify that our output is a* $(1\pm\epsilon)$ *approximation to* $f^2(A)\cdot 1$*, hence the approximation of* 2*-norm squared of each row of* $f(A)$*.*

5 Application to Low Rank Approximation

This section considers the concrete application of rank-k approximation for M where $M_{i,j} = \log(|A_{i,j}|+1)$, i.e., finding k orthonormal vectors L such that $\|M - LL^\top M\|_F$ is minimized. Our algorithm for rank-k approximation is presented in Algorithm 2. Low rank approximation for other functions f follows the same algorithm and similar analysis.

There exists a large body of work for low rank approximation (see, e.g., [HMT11, DMIMW12, Woo14, CW13, MM13, NN13, CW15, RSW16, SWZ17, CGK+17, SWZ18, BW18, KPRW19, SWZ19a, SWZ19b, SWZ19c, Son19, BBB+19, DJS+19, BCW19, IVWW19, BWZ19] and references therein) but most of them are designed for the case without transformation and thus cannot be directly applied. As mentioned in previous sections, if an algorithm only accesses the transformed matrix via a matrix product, plugging in our sketching method leads to a suitable algorithm. We design an algorithm that applies generalized leverage score sampling approach [DMIMW12, BLS+16] for low-rank approximation. Leverage score sampling is a non-oblivious sketching technique that is widely used in numerical linear algebra and has been successfully applied to speed up different problems such as linear regression [CW13, PSW17, AKK+17, SWZ19b, DSWY19], row sampling [SS11, LMP13], spectral approximation [CLM+15], low rank approximation [BW14, SWZ17, SWZ19b], cutting plane methods [Vai89, LSW15, JLSW20], linear programming [BLSS20], computing John Ellipsoid [CCLY19]. From the perspective of graph problems, leverage score is closely related to random spanning tree [Sch18, KS18], graph sparsification and Laplacian system solver [ST04, SS11, BSS12]. Readers may refer to Appendix C.1 for more detailed discussion on leverage score sampling.

On a high level, we would like to sample matrix $M \in \mathbb{R}^{n\times n}$ according to its leverage scores. It turns out it is sufficient to use the leverage scores of SM where S is a sketching matrix. We apply Algorithm 1 to do so and obtain the sampled set P (Step 1). We then apply the technique of adaptive sampling to refine the sampling and obtain Y (Step 2) so that we have better control over the rank, and finally compute the solution using Y by taking projection and computing singular vectors (Step 3). Detailed description and analysis of Algorithm 2 can be found in Appendix C. Overall we have the following guarantee.

Theorem 5.1 (low-rank approximation)**.** *For any parameter* $\epsilon \in (0,1)$ *and integer* $k \ge 1$*, there is an algorithm (procedure* LowRankApprox *in Algorithm 2) that runs in* $\widetilde{O}(n)\cdot k^3 \cdot \mathrm{poly}(1/\epsilon)$ *time, takes* $\widetilde{O}(n)\cdot k^3/\epsilon^2$ *spaces, and outputs a matrix* $L \in \mathbb{R}^{n\times k}$ *such that*

$$\|LL^\top M - M\|_F^2 \le 10\cdot\|M - [M]_k\|_F^2 + O\left(\frac{\epsilon^2}{k^3\log^5 k}\right)\cdot\|M\|_{1,2}^2,$$

holds with probability at least $9/10$*, where* $\|M\|_{1,2} = (\sum_j \|M_{*,j}\|_1^2)^{1/2}$*.*

For a large n and fixed ϵ, our algorithm uses much less space than storing the full matrix. Note that our algorithm still needs to make several passes over the stream of updates. Whether there exists a one-pass algorithm is still an open problem, and is left for future work.

6 Experiments

To demonstrate the advantage of our proposed method, we complement the theoretical analysis with empirical study on synthetic and real data. We consider the low rank approximation task with $f(x) = \log(|x|+1)$. We adjust the constant factors in the amount of space used by our method and compare the errors of the obtained solutions. In the appendix, we describe more experimental details. We also provide additional experiments in the appendix to show that the method also works for $f(x) = \sqrt{|x|}$.

We furthre demonstrate the robustness of the parameter selections in the algorithm.

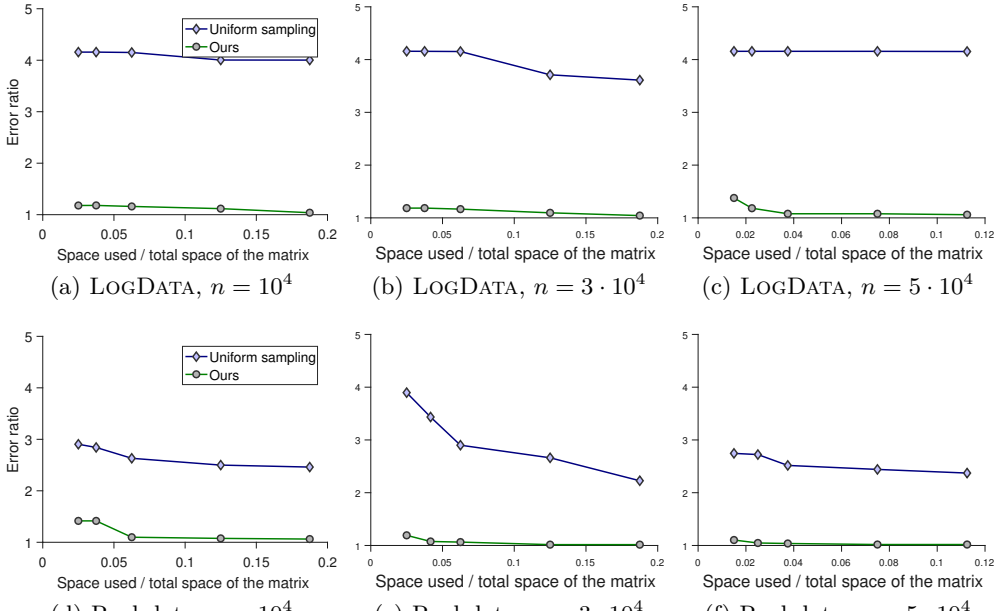
Setup. Given a data stream in the form of (i_t, j_t, δ_t), we use the algorithm in Section 5 to compute the top $k = 10$ singular vectors L, and then compare the error of this solution to the error of the optimal solution (i.e., the true top k singular vectors). Let A denote the accumulated matrix, $M = f(A)$ denote the transformed one, and U denote the top k singular vectors of M. Then the evaluation criterion is

$$\text{error-ratio}(L) = \|M - LL^\top M\|_F / \|M - UU^\top M\|_F.$$

Clearly, the error ratio is at least 1, and a value closer to 1 means a better solution.

Algorithm 2 Low rank approximation of $M = \log(|A| + 1)$

procedure LowRankApprox(A, k, ϵ) ▷ Theorem 5.1
  $s \leftarrow O(k \log k)$
  $d_1 \leftarrow O(k \log^2 k)$
  $d_2 \leftarrow O(k/\epsilon)$
  $\eta \leftarrow O(\epsilon\sqrt{d_1} + \epsilon^2 d_1)$
  ▷ Step 1 : Sampling according to generalized leverage scores of M
  Let S be the CountSketch (SparesJL) matrix of size $s \times n$ ▷ Appendix A.1
  Let S_+ and S_- be its positive and negative parts of S.
  $R \leftarrow [S_+; S_-]$
  $\widetilde{E} \leftarrow \text{LogSum}(RM)$ ▷ $\|\widetilde{E}_i\|_2^2 = (1 \pm \epsilon)\|(RM)_i\|_2, \forall i$
  Sample a set P of d_1 columns of M according to the leverage score of $\widetilde{E}$. ▷ Definition C.2
  ▷ Step 2 : Adaptive sampling
  $[Q_p, \cdot] \leftarrow \text{QRFactorization}(P)$ ▷ Q_p is the basis vectors for P
  $\widetilde{\Gamma} \leftarrow \text{LogSum}(Q_p^\top M)$ ▷ $\|\widetilde{\Gamma}_i\|_2^2 = (1 \pm \epsilon)\|(Q_p^\top M)_i\|_2^2, \forall i$
  $\widetilde{z} \leftarrow \text{LogSum}(M)$ ▷ $\widetilde{z}_i = (1 \pm \epsilon)\|M_i\|_2^2, \forall i$
  $\widetilde{s}_i \leftarrow \widetilde{z}_i - \|\widetilde{\Gamma}_i\|_2^2$
  Sample a set $\widetilde{Y}$ of d_2 columns from M according to $p_i = \max(\widetilde{s}_i, \eta\widetilde{z}_i)$
  $Y \leftarrow \widetilde{Y} \cup P$
  ▷ Step 3 : Computing approximation solutions
  $[Q_y, \cdot] \leftarrow \text{QRFactorization}(Y)$ ▷ Q_y is the basis vectors for Y
  $\widetilde{\Pi} \leftarrow \text{LogSum}(Q_y^\top M)$ ▷ $\|\widetilde{\Pi}_i\|_2^2 = (1 \pm \epsilon^2)\|(Q_y^\top M)_i\|_2^2, \forall i$
  Compute the top k singular vectors $\widetilde{W}$ of $\widetilde{\Pi}$
  $L \leftarrow Q_y\widetilde{W}$
  return L
end procedure



(a) LogData, $n = 10^4$ (b) LogData, $n = 3 \cdot 10^4$ (c) LogData, $n = 5 \cdot 10^4$

(d) Real data, $n = 10^4$ (e) Real data, $n = 3 \cdot 10^4$ (f) Real data, $n = 5 \cdot 10^4$

Figure 1: Error ratios on the synthetic data (top row) and the real data (bottom row). The x-axis is the ratio between the amount of space used by the algorithms and the total amount of space occupied by the data matrix. The y-axis is the ratio between the error of the solutions output by the algorithms and the optimal error.

Besides demonstrating the effectiveness, we also exam the tradeoff between the solution quality and the space used. Recall that there are constant parameters in the sketching methods controlling the amount of space used. We vary its value, and set the parameters in other steps of our algorithm so that the amount of space used is dominated by that of the sketch. We then plot how the error ratios change with the amount of space used. The plotted results are averages of 5 runs; the variances are too small to plot. Finally, we also report the results of a baseline method: uniformly at random sample a subset T of columns from A, and then compute the top k singular vectors of $f(T)$. The space occupied by the columns sampled is similar to the space required by our algorithm for fair comparison. We choose uniform sampling as baseline because to the best of the authors' knowledge, our algorithm is the first one to deal with low-rank approximation on transformed matrix in the stream setting, and we are not aware of any other non-trivial algorithm working in this setting.

6.1 Synthetic Data

Data Generation. The data sets LogData are generated as follows. First generate a matrix M of $n \times n$ where the entries are i.i.d. Gaussians. To break the symmetry of the columns, we scale the norm of the i-th column to $4/i$. Finally, we generate matrix A with $A_{ij} = \exp(M_{ij}) - 1$. Each entry A_{ij} is divided into equally into 5 updates $(i, j, A_{ij}/5)$, and all the updates arrive in a arbitrary order. The size n can be 10000, 30000, and 50000.

Parameter Setting. In our algorithm for low rank approximation, an FJLT matrix S is used [Ach03, AC06]. For the sketching subroutine, instead of specifying the desired ϵ, we directly set the size of the data structure (line 19 in LogSum), so as to exam the tradeoff between space and accuracy. We set $m_c = m_s = m_a$ and set their value so that the space used is at most that used by the sketch method.

Results. Figure 1 top row shows the results on the synthetic data. In general, the error ratio of our method is much better than that of the uniform sampling baseline: ours is close to 1 while that of uniform sampling is about 4. It also shows that our method can greatly reduce the amount of space needed, e.g., by orders of magnitude, but still preserve a good solution. This advantage is more significant on larger data sets. For example, when $n = 50000$, to obtain 5% error over the optimum solution, we only needs space corresponding to 5% of the size of the matrix.

6.2 Real Data

We experiment our method on the real world data from NLP applications, which are the motivating examples for our approach. Our method with $f(x) = \log(|x|+1)$ is used. The parameters are set in a similar way as for the synthetic data.

Data Collection. The data set is the entire Wikipedia corpus [Wik12] consisting of about 3 billion tokens. Details can be found in the appendix and only a brief description is provided here. The matrix to be factorized is M with $M_{ij} = p_j \log(\frac{N_{ij}N}{N_i N_j} + 1)$ where N_{ij} is the number of times words i and j co-occur in a window of size 10, N_i is the number of times word i appears, N is the total number of words in the corpus, and p_j is a weighting factor depending on N_j (putting larger weights on more frequent words). Note that N_i's and N can be computed easily, so essentially the only dynamically update part is $\log N_{ij}$. The data stream is generated by considering each window of size 10 along the sentences in the corpus and collecting the co-occurrence counts of the word pairs in that window. We consider the matrix for the most frequent n words, where $n = 10000$, 30000, and 50000.

Results. Figure 1 bottom row shows the results on the real data. The observations are similar to those on the synthetic data: the errors of our method are much smaller than the baseline, and are close to the optimum. These results again demonstrate the accuracy and space efficiency of our methods.

7 Conclusions

We considered the setting where a large matrix is updated by a data stream and the learning tasks is performed on an element-wise transformation of the matrix. We proposed a method for computing the product of its element-wise transformation with another given matrix. For a large family of transformations, our method only needs a single pass over the data and provable guarantees on the error. Our method uses much smaller space than directly storing the matrix. Our approach can be used as a building block for many learning tasks. We provided a concrete application for low-rank approximation with theoretical analysis and empirical verification, showing the effectiveness of this approach.

References

[AC06] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing (STOC)*, pages 557–563. ACM, 2006.

[Ach03] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

[AKK+17] Naman Agarwal, Sham Kakade, Rahul Kidambi, Yin Tat Lee, Praneeth Netrapalli, and Aaron Sidford. Leverage score sampling for faster accelerated regression and erm. *arXiv preprint arXiv:1711.08426*, 2017.

[ALS+18] Alexandr Andoni, Chengyu Lin, Ying Sheng, Peilin Zhong, and Ruiqi Zhong. Subspace embedding and linear regression with orlicz norm. In *ICML*. arXiv preprint arXiv:1806.06430, 2018.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1):137–147, 1999.

[And17] Alexandr Andoni. High frequency moments via max-stability. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 6364–6368. IEEE, 2017.

[BBB+19] Frank Ban, Vijay Bhattiprolu, Karl Bringmann, Pavel Kolev, Euiwoong Lee, and David P Woodruff. A ptas for ℓ_p-low rank approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 747–766. SIAM, 2019.

[BBC+17] Jaroslaw Blasiok, Vladimir Braverman, Stephen R Chestnut, Robert Krauthgamer, and Lin F Yang. Streaming symmetric norms via measure concentration. In *Proceedings of the 49th Annual Symposium on the Theory of Computing(STOC)*. ACM, `https://arxiv.org/pdf/1511.01111`, 2017.

[BCW19] Ainesh Bakshi, Nadiia Chepurko, and David P Woodruff. Robust and sample optimal algorithms for psd low-rank approximation. *arXiv preprint arXiv:1912.04177*, 2019.

[BCWY16] Vladimir Braverman, Stephen R Chestnut, David P Woodruff, and Lin F Yang. Streaming space complexity of nearly all functions of one variable on frequency vectors. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 261–276. ACM, 2016.

[BGJM16] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[BKSV14] Vladimir Braverman, Jonathan Katzman, Charles Seidell, and Gregory Vorsanger. An optimal algorithm for large frequency moments using o (nˆ(1-2/k)) bits. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 28. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[BLS+16] Maria Florina Balcan, Yingyu Liang, Le Song, David Woodruff, and Bo Xie. Communication efficient distributed kernel principal component analysis. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[BLSS20] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. `https://arxiv.org/pdf/2002.02304.pdf`, 2020.

[BO10a] Vladimir Braverman and Rafail Ostrovsky. Recursive sketching for frequency moments. *arXiv preprint arXiv:1011.2571*, 2010.

[BO10b] Vladimir Braverman and Rafail Ostrovsky. Zero-one frequency laws. In *Proceedings of the forty-second ACM symposium on Theory of computing (STOC)*, pages 281–290. ACM, 2010.

[BPL10] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

[BSS12] Joshua Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. In *SIAM Journal on Computing*, volume 41(6), pages 1704–1721. `https://arxiv.org/pdf/0808.0163`, 2012.

[BVWY17] Vladimir Braverman, Emanuele Viola, David P. Woodruff, and Lin F. Yang. Revisiting frequency moment estimation in random order streams. In *Manuscript*, 2017.

[BW14] Christos Boutsidis and David P Woodruff. Optimal cur matrix decompositions. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 353–362. ACM, `https://arxiv.org/pdf/1405.7910`, 2014.

[BW18] Ainesh Bakshi and David Woodruff. Sublinear time low-rank approximation of distance matrices. In *Advances in Neural Information Processing Systems*, pages 3782–3792, 2018.

[BWZ19] Frank Ban, David Woodruff, and Richard Zhang. Regularized weighted low rank approximation. In *Advances in Neural Information Processing Systems*, pages 4061–4071, 2019.

[BYKS02] Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 623–632. Society for Industrial and Applied Mathematics, 2002.

[CCLY19] Michael B Cohen, Ben Cousins, Yin Tat Lee, and Xin Yang. A near-optimal algorithm for approximating the John ellipsoid. In *Annual Conference on Learning Theory (COLT)*. `https://arxiv.org/pdf/1905.11580.pdf`, 2019.

[CEM+15] Michael B Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (STOC)*, pages 163–172. ACM, `https://arxiv.org/pdf/1410.6801`, 2015.

[CGK+17] Flavio Chierichetti, Sreenivas Gollapudi, Ravi Kumar, Silvio Lattanzi, Rina Panigrahy, and David P Woodruff. Algorithms for ℓ_p low rank approximation. In *ICML*. arXiv preprint arXiv:1705.06730, 2017.

[CLM+15] Michael B Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 181–190. ACM, `https://arxiv.org/pdf/1408.5099`, 2015.

[CW13] Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *Symposium on Theory of Computing Conference (STOC)*, pages 81–90. `https://arxiv.org/pdf/1207.6365`, 2013.

[CW15] Kenneth L Clarkson and David P Woodruff. Input sparsity and hardness for robust subspace approximation. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 310–329. IEEE, `https://arxiv.org/pdf/1510.06073`, 2015.

[CWW19] Kenneth L. Clarkson, Ruosong Wang, and David P Woodruff. Dimensionality reduction for tukey regression. In *ICML*. arXiv preprint arXiv:1904.05543, 2019.

[DJS+19] Huaian Diao, Rajesh Jayaram, Zhao Song, Wen Sun, and David P. Woodruff. Optimal sketching for kronecker product regression and low rank approximation. In *NeurIPS*, 2019.

[DMIMW12] Petros Drineas, Malik Magdon-Ismail, Michael Mahoney, and David Woodruff. Fast approximation of matrix coherence and statistical leverage. *The Journal of*

Machine Learning Research, 13(1):3475–3506, 2012.

[DRVW06] Amit Deshpande, Luis Rademacher, Santosh Vempala, and Grant Wang. Matrix approximation and projective clustering via volume sampling. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1117–1126. Society for Industrial and Applied Mathematics, 2006.

[DSSW18] Huaian Diao, Zhao Song, Wen Sun, and David P. Woodruff. Sketching for kronecker product regression and p-splines. In *AISTATS*, 2018.

[DSWY19] Huaian Diao, Zhao Song, David P. Woodruff, and Xin Yang. Total least squares regression in input sparsity time. In *NeurIPS*, 2019.

[DV06] Amit Deshpande and Santosh Vempala. Adaptive sampling and fast low-rank matrix approximation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 292–303. Springer, 2006.

[DVF13] Debbie J Dupuis and Maria-Pia Victoria-Feser. Robust vif regression with application to variable selection in large data sets. *The Annals of Applied Statistics*, 7(1):319–341, 2013.

[FT07] Shmuel Friedland and Anatoli Torokhti. Generalized rank-constrained matrix approximations. *SIAM Journal on Matrix Analysis and Applications*, 29(2):656–659, 2007.

[Gan07] Sumit Ganguly. Counting distinct items over update streams. *Theoretical Computer Science*, 378(3):211–222, 2007.

[GL16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[GNHS11] Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yannis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM, 2011.

[HMT11] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[Ind00] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings. 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 189–197. IEEE, 2000.

[IVWW19] Piotr Indyk, Ali Vakilian, Tal Wagner, and David Woodruff. Sample-optimal low-rank approximation of distance matrices. *arXiv preprint arXiv:1906.00339*, 2019.

[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 (STOC)*, pages 202–208. ACM, 2005.

[JLSW20] Haotian Jiang, Yin Tat Lee, Zhao Song, and Sam Chiu-wai Wong. An improved cutting plane method for convex optimization, convex-concave games and its applications. In *STOC*, 2020.

[KBV09] Y Koren, R Bell, and C Volinsky. Matrix factorization techniques for recommender systems. *Computer (IEEE) 42 (8): 30-37.*, 2009.

[KN14] Daniel M Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. In *Journal of the ACM (JACM)*, volume 61(1), page 4. `https://arxiv.org/pdf/1012.1577`, 2014.

[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 (SODA)*, pages 1161–1178. Society for Industrial and Applied Mathematics, 2010.

[KPRW19] Ravi Kumar, Rina Panigrahy, Ali Rahimi, and David Woodruff. Faster algorithms for binary matrix factorization. In *International Conference on Machine Learning*, pages 3551–3559, 2019.

[KS18] Rasmus Kyng and Zhao Song. A matrix chernoff bound for strongly rayleigh distributions and spectral sparsifiers from a few random spanning trees. In *FOCS*. https://arxiv.org/pdf/1810.08345, 2018.

[Lat05] Rafał Latała. Some estimates of norms of random matrices. *Proceedings of the American Mathematical Society*, 133(5):1273–1282, 2005.

[LG14] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, 2014.

[LHW17] Xingguo Li, Jarvis Haupt, and David Woodruff. Near optimal sketching of low-rank tensor regression. In *Advances in Neural Information Processing Systems*, pages 3466–3476, 2017.

[LMP13] Mu Li, Gary L. Miller, and Richard Peng. Iterative row sampling. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 127–136. https://arxiv.org/pdf/1211.2713, 2013.

[LSW15] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1049–1065. https://arxiv.org/pdf/1508.04874.pdf, 2015.

[LWW20] Yi Li, Ruosong Wang, and David P. Woodruff. Tight bounds for the subspace sketch problem with applications. In *SODA*, 2020.

[LZM15] Shaohua Li, Jun Zhu, and Chunyan Miao. A generative word embedding model and its low rank positive semidefinite solution. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1599–1609, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[Mah11] Michael Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning*, 3(2):123–224, 2011.

[MM13] Xiangrui Meng and Michael W Mahoney. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing (STOC)*, pages 91–100. ACM, https://arxiv.org/pdf/1210.3135, 2013.

[MS04] Yun Mao and Lawrence K Saul. Modeling distances in large-scale networks by matrix factorization. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 278–287. ACM, 2004.

[MSA+11] Jean-Baptiste Michel, Yuan Kui Shen, Aviva Presser Aiden, Adrian Veres, Matthew K Gray, Joseph P Pickett, Dale Hoiberg, Dan Clancy, Peter Norvig, Jon Orwant, et al. Quantitative analysis of culture using millions of digitized books. *science*, 331(6014):176–182, 2011.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 2013.

[MZ10] Lingsheng Meng and Bing Zheng. The optimal perturbation bounds of the moore–penrose inverse under the frobenius norm. *Linear Algebra and its Applications*, 432(4):956–963, 2010.

[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 (FOCS)*, pages 117–126. IEEE, https://arxiv.org/pdf/1211.1002, 2013.

[PPP06] V Paul Pauca, Jon Piper, and Robert J Plemmons. Nonnegative matrix factorization for spectral data analysis. *Linear algebra and its applications*, 416(1):29–47, 2006.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing*, 2014.

[PSW17] Eric Price, Zhao Song, and David P. Woodruff. Fast regression with an ℓ_∞ guarantee. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2017.

[RR08] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.

[RSW16] Ilya Razenshteyn, Zhao Song, and David P Woodruff. Weighted low rank approximations with provable guarantees. In *Proceedings of the 48th Annual Symposium on the Theory of Computing (STOC)*, 2016.

[Sar06] Tamás Sarlós. Improved approximation algorithms for large matrices via random projections. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS) , 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 143–152, 2006.

[Sch18] Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2018.

[Son19] Zhao Song. *Matrix Theory : Optimization, Concentration and Algorithms*. PhD thesis, The University of Texas at Austin, 2019.

[SS11] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing (STOC)*, pages 81–90. ACM, 2004.

[SWZ17] Zhao Song, David P Woodruff, and Peilin Zhong. Low rank approximation with entrywise ℓ_1-norm error. In *Proceedings of the 49th Annual Symposium on the Theory of Computing (STOC)*. ACM, `https://arxiv.org/pdf/1611.00898`, 2017.

[SWZ18] Zhao Song, David P Woodruff, and Peilin Zhong. Towards a zero-one law for entrywise low rank approximation. *arXiv preprint arXiv:1811.01442*, 2018.

[SWZ19a] Zhao Song, David P Woodruff, and Peilin Zhong. Average case column subset selection for entrywise ℓ_1-norm loss. In *NeurIPS*, 2019.

[SWZ19b] Zhao Song, David P Woodruff, and Peilin Zhong. Relative error tensor low rank approximation. In *SODA*. arXiv preprint arXiv:1704.08246, 2019.

[SWZ19c] Zhao Song, David P Woodruff, and Peilin Zhong. Towards a zero-one law for column subset selection. In *NeurIPS*, 2019.

[Vai89] Pravin M Vaidya. A new algorithm for minimizing convex functions over convex sets. In *30th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 338–343, 1989.

[Var14] Hal R Varian. Big data: New tricks for econometrics. *Journal of Economic Perspectives*, 28(2):3–28, 2014.

[Ver10] Roman Vershynin. Introduction to the non-asymptotic analysis of random matrices. *arXiv preprint arXiv:1011.3027*, 2010.

[Wed73] Per-Åke Wedin. Perturbation theory for pseudo-inverses. *BIT Numerical Mathematics*, 13(2):217–232, 1973.

[Wik12] Wikimedia. English Wikipedia dump. `http://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2`, 2012. Accessed Mar-2015.

[Woo14] David P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1-2):1–157, 2014.

[WSC+16] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[WW19] Ruosong Wang and David P. Woodruff. Tight bounds for ℓ_p oblivious subspace embeddings. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1825–1843, 2019.

[WZ16] David P Woodruff and Peilin Zhong. Distributed low rank approximation of implicit functions of a matrix. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 847–858. IEEE, 2016.

[Zha97] Zhengyou Zhang. Parameter estimation techniques: A tutorial with application to conic fitting. *Image and vision Computing*, 15(1):59–76, 1997.

[ZWSP08] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *International Conference on Algorithmic Applications in Management*, pages 337–348. Springer, 2008.