Approximate Multiplication of Sparse Matrices with Limited Space

Yuanyu Wan, Lijun Zhang[*]
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
{wanyy, zhanglj}@lamda.nju.edu.cn

Abstract

Approximate matrix multiplication with limited space has received ever-increasing attention due to the emergence of large-scale applications. Recently, based on a popular matrix sketching algorithm—frequent directions, previous work has introduced co-occuring directions (COD) to reduce the approximation error for this problem. Although it enjoys the space complexity of $O((m_x + m_y)\ell)$ for two input matrices $X \in \mathbb{R}^{m_x \times n}$ and $Y \in \mathbb{R}^{m_y \times n}$ where ℓ is the sketch size, its time complexity is $O\left(n(m_x + m_y + \ell)\ell\right)$, which is still very high for large input matrices. In this paper, we propose to reduce the time complexity by exploiting the sparsity of the input matrices. The key idea is to employ an approximate singular value decomposition (SVD) method which can utilize the sparsity, to reduce the number of QR decompositions required by COD. In this way, we develop sparse co-occuring directions, which reduces the time complexity to $\widetilde{O}\left((\mathrm{nnz}(X) + \mathrm{nnz}(Y))\ell + n\ell^2\right)$ in expectation while keeps the same space complexity as $O((m_x + m_y)\ell)$, where $\mathrm{nnz}(X)$ denotes the number of non-zero entries in X and the $\widetilde{O}$ notation hides constant factors as well as polylogarithmic factors. Theoretical analysis reveals that the approximation error of our algorithm is almost the same as that of COD. Furthermore, we empirically verify the efficiency and effectiveness of our algorithm.

Introduction

Matrix multiplication refers to computing the product XY^T of two matrices $X \in \mathbb{R}^{m_x \times n}$ and $Y \in \mathbb{R}^{m_y \times n}$, which is a fundamental task in many machine learning applications such as regression (Naseem, Togneri, and Bennamoun 2010; Cohen, Nelson, and Woodruff 2015), online learning (Hazan, Agarwal, and Kale 2007; Chu et al. 2011; Zhang et al. 2016, 2017), information retrieval (Eriksson-Bique et al. 2011) and canonical correlation analysis (Hotelling 1936; Chen, Liu, and Carbonell 2015). Recently, the scales of data and models in these applications have increased dramatically, which results in very large data matrices. As a result, it requires unacceptable time and space to directly compute XY^T in the main memory. To reduce both time and space complexities, approximate matrix multiplication (AMM) with limited space, which can efficiently compute a good approximation of the matrix product, has been a substitute and received ever-increasing attention (Ye, Luo, and Zhang 2016; Wan and Zhang 2018; Wan, Wei, and Zhang 2018; Kuzborskij, Cella, and Cesa-Bianchi 2019).

Given two large matrices $X \in \mathbb{R}^{m_x \times n}$ and $Y \in \mathbb{R}^{m_y \times n}$, the goal of AMM with limited space is to find two small sketches $B_X \in \mathbb{R}^{m_x \times \ell}$ and $B_Y \in \mathbb{R}^{m_y \times \ell}$ such that $B_X B_Y^T$ approximates XY^T well, where $\ell \ll \min(m_x, m_y, n)$ is the sketch size. Traditionally, randomized techniques such as column selection (Drineas, Kannan, and Mahoney 2006) and random projection (Sarlos 2006; Magen and Zouzias 2011; Cohen, Nelson, and Woodruff 2015) have been utilized to develop lightweight algorithms with the $O(n(m_x + m_y)\ell)$ time complexity and $O((m_x + m_y)\ell)$ space complexity for AMM, and yielded theoretical guarantees for the approximation error. Specifically, early studies (Drineas, Kannan, and Mahoney 2006; Sarlos 2006) focused on the Frobenius error, and achieved the following bound

$$\|XY^T - B_X B_Y^T\|_F \leq \epsilon \|X\|_F \|Y\|_F \tag{1}$$

with $\ell = \widetilde{O}(1/\epsilon^2)$. Later, two improvements (Magen and Zouzias 2011; Cohen, Nelson, and Woodruff 2015) established the following error bound measured by the spectral norm

$$\|XY^T - B_X B_Y^T\| \leq \epsilon \|X\| \|Y\| \tag{2}$$

with $\ell = \widetilde{O}\left((\mathrm{sr}(X) + \mathrm{sr}(Y))/\epsilon^2\right)$ where $\mathrm{sr}(X) = \frac{\|X\|_F^2}{\|X\|^2}$ is the stable rank of X.

However, their sketch size ℓ has a quadratic dependence on $1/\epsilon$, which means that a large sketch size is required to ensure a small approximation error. To improve the dependence on $1/\epsilon$, recent studies (Ye, Luo, and Zhang 2016; Mroueh, Marcheret, and Goel 2017) have extended a deterministic matrix sketching technique called frequent directions (FD) (Liberty 2013; Ghashami and Phillips 2014; Ghashami et al. 2016) to AMM. Specifically, Ye, Luo, and Zhang (2016) concatenated X, Y vertically as $[X; Y]$, and applied FD to $[X; Y] \in \mathbb{R}^{(m_x + m_y) \times n}$ to generate B_X and B_Y such that

$$\|XY^T - B_X B_Y^T\| \leq \left(\|X\|_F^2 + \|Y\|_F^2\right)/\ell \tag{3}$$

which also requires the $O(n(m_x + m_y)\ell)$ time complexity and $O((m_x + m_y)\ell)$ space complexity. Furthermore,

[*]Lijun Zhang is the corresponding author.

Mroueh, Marcheret, and Goel (2017) proposed a new algorithm named as co-occuring directions (COD) with the $O((m_x + m_y)\ell)$ space complexity to generate B_X and B_Y such that

$$\|XY^T - B_X B_Y^T\| \leq 2\|X\|_F \|Y\|_F/\ell. \quad (4)$$

Although the error bound of COD is similar to that in (3), Mroueh, Marcheret, and Goel (2017) showed that COD usually has a better empirical performance. Compared with previous randomized methods, COD only requires $\ell = 2\sqrt{\mathrm{sr}(X)\,\mathrm{sr}(Y)}/\epsilon$ to achieve the error bound in (2), which improves the dependence on $1/\epsilon$ to be linear. However, the time complexity of COD is $O\left(n(m_x + m_y + \ell)\ell\right)$, which is still very high for large matrices.

In this paper, we exploit the sparsity of the input matrices to reduce the time complexity of COD. In many real applications, the sparsity is a common property for large matrices. For example, in information retrieval, the word-by-document matrix could contain less than 5% non-zero entries (Dhillon and Modha 2001). In recommender systems, the user-item rating matrix could contain less than 7% non-zero entries (Zhang, Yao, and Xu 2017). The computational bottleneck of COD is to compute the QR decomposition $O(n/\ell)$ times. We note that a similar bottleneck also exists in FD, which needs to compute SVD $O(n/\ell)$ times. To make FD more efficient for sparse matrices, Ghashami, Liberty, and Phillips (2016) utilized a randomized SVD algorithm named as simultaneous iteration (SI) (Musco and Musco 2015) to reduce the number of exact SVD. Inspired by this work, we propose to accelerate COD for sparse matrices by utilizing SI. Our main contributions are summarized as follows.

- We first develop verified simultaneous iteration (VSI) by introducing a verification process, which can efficiently perform a good decomposition for the product of two small sparse matrices with a sufficiently large probability.
- Then, we develop sparse co-occuring directions (SCOD) by employing VSI to reduce the number of QR decompositions required by COD. In this way, the time complexity is reduced to $\widetilde{O}\left((\mathrm{nnz}(X) + \mathrm{nnz}(Y))\ell + n\ell^2\right)$ in expectation.
- Moreover, we prove that the space complexity of our algorithm is still $O((m_x + m_y)\ell)$, and it enjoys almost the same error bound as that of COD.

Preliminaries

In this section, we review necessary preliminaries about co-occuring directions and simultaneous iteration.

Co-occuring Directions

Co-occuring directions (Mroueh, Marcheret, and Goel 2017) is an extension of frequent directions (Liberty 2013) for AMM. For brevity, the most critical procedures of COD are extracted and summarized in Algorithm 1, which is named as dense shrinkage (DS), where $\sigma_x(A)$ is the x-th largest singular value of any matrix A, and $\max(A, 0)$ is performed on each entry of A. Given $X \in \mathbb{R}^{m_x \times n}$ and $Y \in \mathbb{R}^{m_y \times n}$, COD first initializes $B_X = 0_{m_x \times \ell}$ and $B_Y = 0_{m_y \times \ell}$. Then, it processes the i-th column of X and Y as follows

```
Insert $X_i$ into a zero valued column of $B_X$
Insert $Y_i$ into a zero valued column of $B_Y$
**if** $B_X, B_Y$ have no zero valued column **then**
    $B_X, B_Y = \mathrm{DS}(B_X, B_Y)$
**end if**
```

for $i = 1, \cdots, n$. Each time B_X, B_Y have no zero valued column, performing $B_X, B_Y = \mathrm{DS}(B_X, B_Y)$ can ensure that the last $1 + \ell/2$ columns of B_X and B_Y are zero valued. Therefore, it is easy to verify that the space complexity of COD is only $O((m_x + m_y)\ell)$. However, it needs to compute the QR decomposition of B_X, B_Y and SVD of $R_X R_Y^T$ almost $O(n/\ell)$ times, which implies that its time complexity is

$$O\left(\frac{n}{\ell}(m_x\ell^2 + m_y\ell^2 + \ell^3)\right) = O(n(m_x + m_y + \ell)\ell).$$

We will reduce the time complexity by utilizing the sparsity of the input matrices. Our key idea is to employ simultaneous iteration to reduce the number of QR decompositions and SVD.

Algorithm 1 Dense Shrinkage (DS)

```
1: **Input:** $B_X \in \mathbb{R}^{m_x \times \ell'}, B_Y \in \mathbb{R}^{m_y \times \ell'}$
2: $Q_X, R_X = \mathrm{QR}(B_X)$
3: $Q_Y, R_Y = \mathrm{QR}(B_Y)$
4: $U, \Sigma, V = \mathrm{SVD}(R_X R_Y^\top)$
5: $\gamma = \sigma_{\ell'/2}(\Sigma)$
6: $\widetilde{\Sigma} = \max(\Sigma - \gamma I_{\ell'}, 0)$
7: $B_X = Q_X U \sqrt{\widetilde{\Sigma}}$
8: $B_Y = Q_Y V \sqrt{\widetilde{\Sigma}}$
9: **return** $B_X, B_Y$
```

Simultaneous Iteration

Simultaneous iteration (Musco and Musco 2015) is a randomized method for approximate SVD, which provides a very efficient way to handle sparse matrices. Specifically, given a matrix $A \in \mathbb{R}^{m_x \times m_y}$ and an error coefficient ϵ, it performs the following procedures

$$q = O\left(\log(m_x)/\epsilon\right), G \sim \mathcal{N}(0,1)^{m_y \times \ell}$$
$$K = (AA^T)^q AG \quad (5)$$

Orthonormalize the columns of K to obtain Q

to generate an orthonormal matrix $Q \in \mathbb{R}^{m_x \times \ell}$, where $G \sim \mathcal{N}(0,1)^{m_y \times \ell}$ denotes that each entry of G is independently sampled from the standard Gaussian distribution $\mathcal{N}(0,1)$. SI enjoys the the following guarantee (Musco and Musco, 2015, Theorem 10).

Theorem 1 *With probability* 99/100*, applying (5) to any matrix* $A \in \mathbb{R}^{m_x \times m_y}$ *has*

$$\|A - QQ^T A\| \leq (1 + \epsilon)\sigma_{\ell+1}(A).$$

Algorithm 2 Simultaneous Iteration (SI)

```
1: Input: $S_X \in \mathbb{R}^{m_x \times d}, S_Y \in \mathbb{R}^{m_y \times d}, \ell, 0 < \epsilon < 1$
2: $q = O(\log(m_x)/\epsilon)$, $G \sim \mathcal{N}(0,1)^{m_y \times \ell}$
3: $K = S_X(S_Y^T G)$
4: while $q > 0$ do
5:     $K = S_X(S_Y^T(S_Y(S_X^T K))), q = q - 1$
6: end while
7: Orthonormalize the columns of $K$ to obtain $Q$
8: return $Q, S_Y(S_X^T Q)$
```

Note that some earlier studies (Rokhlin, Szlam, and Tygert 2009; Halko, Martinsson, and Tropp 2011; Woodruff 2014; Witten and Candès 2014) have also analyzed this algorithm and achieved similar results.

We will utilize SI to approximately decompose $A = S_X S_Y^T$, where $S_X \in \mathbb{R}^{m_x \times d}, S_Y \in \mathbb{R}^{m_y \times d}$ are two buffer matrices, which will be used to store non-zero entries and satisfy $d \leq m = \max(m_x, m_y)$, $\mathrm{nnz}(S_X) \leq m\ell$ and $\mathrm{nnz}(S_Y) \leq m\ell$. The detailed procedures are derived by substituting $A = S_X S_Y^T$ into (5), and are summarized in Algorithm 2. Specifically, lines 3 to 6 in SI is designed to compute

$$K = (S_X S_Y^T S_Y S_X^T)^q S_X S_Y^T G$$

in $O((\mathrm{nnz}(S_X)+\mathrm{nnz}(S_Y))\ell \log(m_x))$ time, which requires $O((m_x + m_y + d)\ell)$ space. Line 7 in SI can be implemented by Gram-Schmidt orthogonalization or Householder reflections, which only requires $O(m_x\ell^2)$ time and $O(m_x\ell)$ space. Due to $d \leq m = \max(m_x, m_y)$, the space complexity of SI is only $O(m\ell)$, and the time complexity of SI is

$$O((\mathrm{nnz}(S_X) + \mathrm{nnz}(S_Y))\ell \log(m_x) + m_x\ell^2) \tag{6}$$

which is efficient for sparse S_X and S_Y. By comparison, decomposing $S_X S_Y^T$ with DS requires $O((m_x+m_y)d^2+d^3)$ time and $O(md)$ space, which is unacceptable for large d even if S_X and S_Y are very sparse.

Main Results

In this section, we first incorporate simultaneous iteration with a verification process, which is necessary for controlling the failure probability of our algorithm. Then, we describe our sparse co-occuring directions for AMM with limited space and its theoretical results. Finally, we provide a detailed space and runtime analysis of our algorithm. The proofs for theoretical results can be found in the full version (Wan and Zhang 2020).

Verified Simultaneous Iteration

From previous discussions, in the simple case $n \leq m = \max(m_x, m_y)$, $\mathrm{nnz}(X) \leq m\ell$ and $\mathrm{nnz}(Y) \leq m\ell$, we can generate B_X and B_Y by performing

$$B_X, B_Y = \mathrm{SI}(X, Y, \ell, 1/10).$$

According to Theorem 1, with probability $99/100$

$$\|XY^T - B_X B_Y^T\| \leq \left(1 + \frac{1}{10}\right)\sigma_{\ell+1}(XY^T).$$

Algorithm 3 Verified Simultaneous Iteration (VSI)

```
1: Input: $S_X \in \mathbb{R}^{m_x \times d}, S_Y \in \mathbb{R}^{m_y \times d}, \ell, 0 < \delta < 1$
2: Initialization: persistent $j = 0$ ($j$ retains its value between invocations)
3: $j = j + 1, p = \lceil \log(2j^2\sqrt{m_x e}/\delta) \rceil$
4: $\Delta = \frac{11}{10\ell}\sum_{i=1}^{\mathrm{cols}(S_X)} \|S_{X,i}\|_2 \|S_{Y,i}\|_2$
5: while True do
6:     $C_X, C_Y = \mathrm{SI}(S_X, S_Y, \ell, 1/10)$
7:     $C = (S_X S_Y^T - C_X C_Y^T)/\Delta$ ($C$ is not computed)
8:     $\mathbf{x} \sim \mathcal{N}(0,1)^{m_x \times 1}$
9:     if $\|(CC^T)^p \mathbf{x}\|_2 \leq \|\mathbf{x}\|_2$ then
10:        return $C_X, C_Y$
11:    end if
12: end while
```

Although X and Y generally have more non-zero entries and columns, we could divide X and Y into several smaller matrices that satisfy the conditions of the above simple case, and repeatedly perform SI. However, in this way, the failure probability will increase linearly, where failure means that there exists a run of SI, after which the error between its input and output is unbounded. To reduce the failure probability, we need to verify whether the error between the input and output is bounded by a small value after each run of SI.

Ghashami, Liberty, and Phillips (2016) have proposed an algorithm to verify the spectral norm of a symmetric matrix. Inspired by their algorithm, we propose verified simultaneous iteration (VSI) as described in Algorithm 3, where δ is the failure probability. Let S_X and S_Y be its two input matrices. In line 3 of VSI, we use j to record the number of invocations of VSI and set $p = \lceil \log(2j^2\sqrt{m_x e}/\delta) \rceil$, where e is Euler's number. In line 4 of VSI, we set $\Delta = \frac{11}{10\ell}\sum_{i=1}^{\mathrm{cols}(S_X)} \|S_{X,i}\|_2 \|S_{Y,i}\|_2$, where $\mathrm{cols}(S_X)$ denotes the column number of S_X. From lines 5 to 12 of VSI, we first utilize SI to generate C_X, C_Y, and then verify whether

$$\|(CC^T)^p \mathbf{x}\|_2 \leq \|\mathbf{x}\|_2 \tag{7}$$

holds, where $C = (S_X S_Y^T - C_X C_Y^T)/\Delta$ and $\mathbf{x}$ is drawn from $\mathcal{N}(0,1)^{m_x \times 1}$. If so, we will return C_X, C_Y. Otherwise, we will rerun SI and repeat the verification process until it holds.

Note that the condition (7) is used to verify whether $\|C\| > 2$, and if C satisfies this condition, with high probability, $\|C\| \leq 2$. Specifically, we establish the following guarantee.

Lemma 1 *Assume that C_X, C_Y are returned by the j-th run of VSI, with probability at least $1 - \frac{\delta}{2j^2}$,*

$$\left\|S_X S_Y^T - C_X C_Y^T\right\| \leq 2\Delta$$

where $\Delta = \frac{11}{10\ell}\sum_{i=1}^{\mathrm{cols}(S_X)} \|S_{X,i}\|_2 \|S_{Y,i}\|_2$.

Lemma 1 implies that the failure probability of bounding $\left\|S_X S_Y^T - C_X C_Y^T\right\|$ decreases as the number of invocations of VSI increases, instead of keeping $1/100$ for the naive SI, which is essential for our analysis.

Algorithm 4 Sparse Co-occuring Directions (SCOD)

```
1:  Input: $X \in \mathbb{R}^{m_x \times n}, Y \in \mathbb{R}^{m_y \times n}, \ell, 0 < \delta < 1$
2:  Initialization: $B_X = 0_{m_x \times l}, B_X = 0_{m_y \times l}$, $S_X = 0_{m_x \times 0}, S_Y = 0_{m_y \times 0}$
3:  for $i = 1, ..., n$ do
4:      $S_X = [S_X, X_i], S_Y = [S_Y, Y_i]$
5:      if $\text{nnz}(S_X) \geq \ell m$ or $\text{nnz}(S_Y) \geq \ell m$ or $\text{cols}(S_X) = m$ then
6:          $C_X, C_Y = \text{VSI}(S_X, S_Y, \ell, \delta)$
7:          $D_X = [B_X, C_X], D_Y = [B_Y, C_Y]$
8:          $B_X, B_Y = \text{DS}(D_X, D_Y)$
9:          $S_X = 0_{m_x \times 0}, S_Y = 0_{m_y \times 0}$
10:     end if
11: end for
12: return $B_X, B_Y$
```

Sparse Co-occuring Directions

To work with limited space and exploit the sparsity, we propose an efficient variant of COD for sparse matrices as follows.

Let $X \in \mathbb{R}^{m_x \times n}$ and $Y \in \mathbb{R}^{m_y \times n}$ be the two input matrices. In the beginning, we initialize $B_X = 0_{m_x \times \ell}$ and $B_Y = 0_{m_y \times \ell}$, where $\ell \ll \min(m_x, m_y, n)$. Moreover, we initialize two empty buffer matrices as $S_X = 0_{m_x \times 0}$ and $S_Y = 0_{m_y \times 0}$, which will be used to store the non-zero entries of X and Y. To avoid excessive space cost, the buffer matrices are deemed full when S_X or S_Y contains $m\ell$ non-zero entries or m columns. For $i = 1, \cdots, n$, after receiving X_i and Y_i, we store the non-zero entries of them in the buffer matrices as

$$S_X = [S_X, X_i], S_Y = [S_Y, Y_i]$$

where $[\cdot, \cdot]$ concatenates two matrices horizontally. Each time the buffer matrices become full, we first utilize VSI in Algorithm 3 to approximately decompose $S_X S_Y^T$ as

$$C_X, C_Y = \text{VSI}(S_X, S_Y, \ell, \delta)$$

where $C_X \in \mathbb{R}^{m_x \times \ell}, C_Y \in \mathbb{R}^{m_y \times \ell}$ and δ is the failure probability. Then, to merge C_X, C_Y into B_X, B_Y, we utilize DS in Algorithm 1 as follows

$$D_X = [B_X, C_X], D_Y = [B_Y, C_Y]$$
$$B_X, B_Y = \text{DS}(D_X, D_Y)$$

where $B_X \in \mathbb{R}^{m_x \times \ell}, B_Y \in \mathbb{R}^{m_y \times \ell}$ are large enough to store the non-zero valued columns returned by DS. Finally, we reset the buffer matrices as

$$S_X = 0_{m_x \times 0}, S_Y = 0_{m_y \times 0}$$

and continue to process the remaining columns in the same way. The detailed procedures of our algorithm are summarized in Algorithm 4 and it is named as sparse co-occuring directions (SCOD).

Following Mroueh, Marcheret, and Goel (2017), we first bound the approximation error of our SCOD as below.

Theorem 2 *Given $X \in \mathbb{R}^{m_x \times n}, Y \in \mathbb{R}^{m_y \times n}, \ell \leq \min(m_x, m_y, n)$ and $\delta \in (0, 1)$, Algorithm 4 outputs $B_X \in \mathbb{R}^{m_y \times \ell}, B_Y \in \mathbb{R}^{m_y \times \ell}$ such that*

$$\|XY^T - B_X B_Y^T\| \leq \frac{16\|X\|_F\|Y\|_F}{5\ell}$$

with probability at least $1 - \delta$.

Compared with the error bound (4) of COD, the error bound of our SCOD only magnifies it by a small constant factor of 1.6. Furthermore, the following theorem shows that the output of SCOD can be used to compute a rank-k approximation for XY^T.

Theorem 3 *Let $B_X \in \mathbb{R}^{m_y \times \ell}, B_Y \in \mathbb{R}^{m_y \times \ell}$ be the output of Algorithm 4. Let $k \leq \ell$ and $\bar{U}, \bar{V}$ be the matrices whose columns are the top-k left and right singular vectors of $B_X B_Y^T$. Let $\pi_{\bar{U}(X)} = \bar{U}\bar{U}^T X$ and $\pi_{\bar{V}}(Y) = \bar{V}\bar{V}^T Y$. For $\epsilon > 0$ and $\ell \geq \frac{64\sqrt{\text{sr}(X)\,\text{sr}(Y)}}{5\epsilon} \frac{\|X\|\|Y\|}{\sigma_{k+1}(XY^T)}$, we have*

$$\|XY^T - \pi_{\bar{U}}(X)\pi_{\bar{V}}(Y)^T\| \leq \sigma_{k+1}(XY^T)(1 + \epsilon) \quad (8)$$

with probability at least $1 - \delta$.

Mroueh, Marcheret, and Goel (2017) have proved that the output of COD enjoys (8) with

$$\ell \geq \frac{8\sqrt{\text{sr}(X)\,\text{sr}(Y)}}{\epsilon} \frac{\|X\|\|Y\|}{\sigma_{k+1}(XY^T)}. \quad (9)$$

By contrast, the lower bound of ℓ for our SCOD only magnifies the right term in (9) by a small constant factor of 1.6.

Space and Runtime Analysis

The total space complexity of our SCOD is only $O(m\ell)$, as explained below.

- $B_X, B_Y, S_X, S_Y, C_X, C_Y, D_X, D_Y$ maintained in Algorithm 4 only require $O(m\ell)$ space.
- Because of the *if* conditions in SCOD, VSI invoked by Algorithm 4 only requires $O(m\ell)$ space.
- Because of $\text{cols}(D_X) = \text{cols}(D_Y) = 2\ell$, DS invoked by Algorithm 4 only requires $O(m\ell)$ space.

To analyze the expected runtime of SCOD, we first note that it is dominated by the cumulative runtime of VSI and DS that are invoked after each time the *if* statement in Algorithm 4 is triggered. Without loss of generality, we assume that the *if* statement is triggered s times in total. It is not hard to verify

$$s \leq \frac{\text{nnz}(X) + \text{nnz}(Y)}{m\ell} + \frac{n}{m}. \quad (10)$$

Because of $\text{cols}(D_X) = \text{cols}(D_Y) = 2\ell$, each run of D-S requires $O(m\ell^2)$ time. Therefore, the total time spent by invoking DS s times is

$$O\left(sm\ell^2\right) = O\left((\text{nnz}(X) + \text{nnz}(Y))\ell + n\ell^2\right). \quad (11)$$

Then, we further bound the expected cumulative runtime of VSI. It is not hard to verify that the runtime of VSI is dominated by the time spent by lines 6 and 9 in Algorithm 3.

According to Theorem 1, after each execution of line 6 in Algorithm 3, with probability $99/100$, we have

$$\begin{aligned}\|S_XS_Y^T - C_XC_Y^T\| &\leq (1+\frac{1}{10})\sigma_{\ell+1}\left(S_XS_Y^T\right)\\ &\leq \frac{11}{10\ell}\|S_XS_Y^T\|_*\\ &\leq \frac{11}{10\ell}\sum_{i=1}^{\mathrm{cols}(S_X)}\|S_{X,i}S_{Y,i}^T\|_*\\ &\leq \frac{11}{10\ell}\sum_{i=1}^{\mathrm{cols}(S_X)}\|S_{X,i}\|_2\|S_{Y,i}\|_2\\ &=\Delta\end{aligned}$$

which implies $\left\|(S_XS_Y^T - C_XC_Y^T)/\Delta\right\| \leq 1$.

Combining with $C = (S_XS_Y^T - C_XC_Y^T)/\Delta$, we have

$$\|(CC^T)^p\mathbf{x}\|_2 \leq \|(CC^T)\|^p\|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_2$$

with probability $99/100$. Therefore, for Algorithm 3, the probability that C_X and C_Y generated by executing its line 6 satisfy the *if* condition in its line 9 is $99/100 > 1/2$. Hence, in each run of VSI, the expected number of executing lines 6 and 9 is at most 2.

Because of the *if* conditions in SCOD and the time complexity of SI in (6), each time executing line 6 in VSI requires

$$O((\mathrm{nnz}(S_X)+\mathrm{nnz}(S_Y))\ell\log(m_x)+m_x\ell^2)$$

time. Let S_X^t, S_Y^t denote the values of S_X, S_Y in the t-th execution of line 6 in Algorithm 4, where $t = 1,\cdots,s$. Because of $\sum_{t=1}^s \mathrm{nnz}(S_X^t) = \mathrm{nnz}(X)$ and $\sum_{t=1}^s \mathrm{nnz}(S_Y^t) = \mathrm{nnz}(Y)$, in expectation, the total time spent by executing line 6 in VSI is

$$O((\mathrm{nnz}(X)+\mathrm{nnz}(Y))\ell\log(m_x)+n\ell^2). \tag{12}$$

In the j-th run of VSI, each time executing its line 9 needs to compute $\|(CC^T)^p\mathbf{x}\|_2$, which requires $O(m\ell p)$ time, because $C = (S_XS_Y^T - C_XC_Y^T)/\Delta$ and S_X, S_Y, C_X, C_Y have less than $O(m\ell)$ non-zero entries. Note that $p = \left\lceil\log(2j^2\sqrt{m_xe}/\delta)\right\rceil$. So, in expectation, the total time spent by executing line 9 in VSI is

$$\sum_{j=1}^{s} O\left(m\ell\log(m_xj/\delta)\right) \leq O\left(sm\ell\log(m_xs/\delta)\right).$$

Finally, combining the above inequality, (10), (11) and (12), the expected runtime of SCOD is

$$O\left(N\ell\log(m_x)+n\ell^2+(N+n\ell)\log(n/\delta)\right)$$

where $N = \mathrm{nnz}(X)+\mathrm{nnz}(Y)$.

Discussions

First, we note that there exist numerical softwares (e.g., Matlab), which provide highly optimized subroutine to efficiently compute the exact multiplication XY^T, if X and Y are sparse. However, they suffer a high space complexity of $\mathrm{nnz}(X)+\mathrm{nnz}(Y)+\mathrm{nnz}(XY^T)$ to operate X, Y and XY^T in the main memory, which is not applicable for large matrices when the memory space is limited. By contrast, the space complexity of our SCOD is only $O(m\ell)$.

Second, when this paper is under review, we find that a concurrent work (Luo et al. 2020) independently proposed almost the same algorithm as our work. The only difference between the algorithm of Luo et al. (2020) and our SCOD is the way to control the failure probability caused by repeatedly invoking SI. In our paper, this problem is addressed by adding a verification process to SI. By contrast, Luo et al. (2020) addressed it by increasing the number of iteration in SI according to the number of invocations of SI. Furthermore, with a more careful analysis, Luo et al. (2020) established the error bound of

$$\begin{aligned}&\|XY^T - B_XB_Y^T\|\\ \leq &O\left(\frac{\|X\|_F\|Y\|_F - \sum_{i=1}^k\sigma_i(XY^T)}{\ell-k}\right)\end{aligned}$$

for $k<\ell$, which is tighter than our error bound in Theorem 2 when XY^T is low-rank. Since their algorithm is almost the same as ours, their error bound actually also provides a better guarantee for our algorithm.

Experiments

In this section, we perform numerical experiments to verify the efficiency and effectiveness of our SCOD.

Datasets

We conduct experiments on two synthetic datasets and two real datasets: NIPS conference papers[1] (Perrone et al. 2017) and MovieLens 10M[2]. Each dataset consists of two sparse matrices $X\in\mathbb{R}^{m_x\times n}$ and $Y\in\mathbb{R}^{m_y\times n}$. The synthetic datasets are randomly generated with sprand, which is a built-in function of Matlab. Specifically, we first generate a low-rank dataset by setting

$$\begin{aligned}X &= \mathrm{sprand}(1e3, 1e4, 0.01, \mathbf{r})\\ Y &= \mathrm{sprand}(2e3, 1e4, 0.01, \mathbf{r})\end{aligned}$$

where $\mathbf{r} = [400, 399, \cdots, 1] \in \mathbb{R}^{400}$, which ensures that $X\in\mathbb{R}^{1e3\times1e4}$ and $Y^{2e3\times1e4}$ only contain 1% non-zero entries and their non-zero singular values are equal to $\mathbf{r}$. With the same $\mathbf{r}$, a noisy low-rank dataset is generated by adding a sparse noise to the above low-rank matrices as

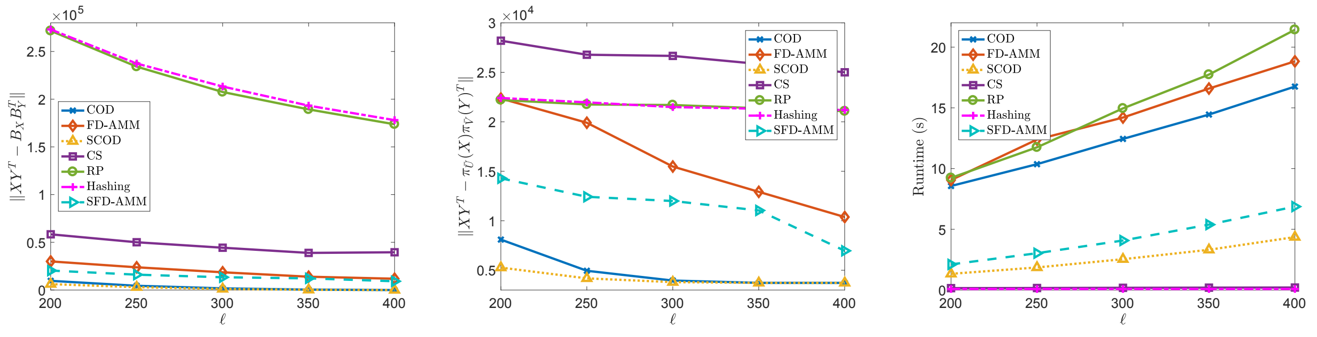
$$\begin{aligned}X &= \mathrm{sprand}(1e3, 1e4, 0.01, \mathbf{r}) + \mathrm{sprand}(1e3, 1e4, 0.01)\\ Y &= \mathrm{sprand}(2e3, 1e4, 0.01, \mathbf{r}) + \mathrm{sprand}(2e3, 1e4, 0.01)\end{aligned}$$

where X and Y contain less than 2% non-zero entries.

Moreover, NIPS conference papers dataset is originally a 11463×5811 word-by-document matrix M, which contains the distribution of words in 5811 papers published between the years 1987 and 2015. In our experiment, let X^T be the

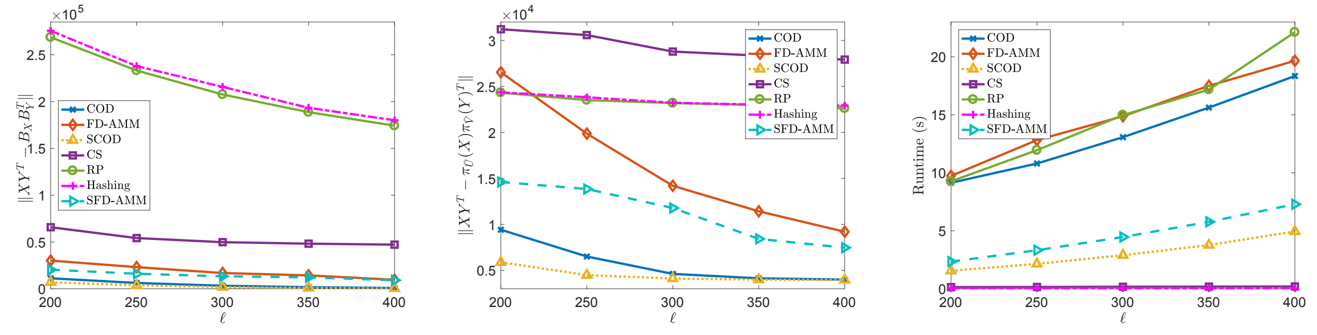
[1]https://archive.ics.uci.edu/ml/datasets/NIPS+Conference+Papers+1987-2015

[2]https://grouplens.org/datasets/movielens/10m/



(a) Approximation Error (b) Projection Error (c) Runtime

Figure 1: Experimental results among different sketch size on the low-rank dataset.



(a) Approximation Error (b) Projection Error (c) Runtime

Figure 2: Experimental results among different sketch size on the noisy low-rank dataset.

first 2905 columns of M, and let Y^T be the others. Therefore, the product XY^T reflects the similarities between two sets of papers. Similarly, MovieLens 10M dataset is originally a 69878×10677 user-item rating matrix M. We also let X^T be the first 5338 columns of M and Y^T be the others.

Baselines and Setting

We first show that our SCOD can match the accuracy of COD, and significantly reduce the runtime of COD for sparse matrices. Moreover, we compare SCOD against other baselines for AMM with limited space including FD-AMM (Ye, Luo, and Zhang 2016) and the following randomized algorithms:

- Sparse frequent directions based AMM (SFD-AMM): replacing FD used in FD-AMM with sparse frequent directions (SFD) proposed by Ghashami, Liberty, and Phillips (2016), which is suggested by an anonymous reviewer;
- Column selection (CS) (Drineas, Kannan, and Mahoney 2006): independently sampling ℓ columns from X and Y with the probability $p_i = \frac{\|X_i\|_2\|Y_i\|_2}{S}$ for each column pair X_i, Y_i, and scaling each selected column by $\frac{1}{\sqrt{\ell p_i}}$, where $S = \sum_{i=1}^n \|X_i\|_2\|Y_i\|_2$, which is implemented with ℓ independent reservoir samplers for processing the data in a streaming way;
- Random projection (RP) (Sarlos 2006): $B_X = X\Pi$, $B_Y = Y\Pi$, where $\Pi \in \mathbb{R}^{n\times\ell}$ and each entry Π_{ij} is uniformly sampled from $\{-1/\sqrt{\ell}, 1/\sqrt{\ell}\}$;[3]
- Hashing (Clarkson and Woodruff 2013): for each $i = 1, \cdots, n$, updating B_X, B_Y as $B_{X,h(i)} = B_{X,h(i)} + s(i)X_i$, $B_{Y,h(i)} = B_{Y,h(i)} + s(i)Y_i$, where $h(i)$ and $s(i)$ are uniformly sampled from $\{1, \cdots, \ell\}$ and $\{1, -1\}$, respectively.

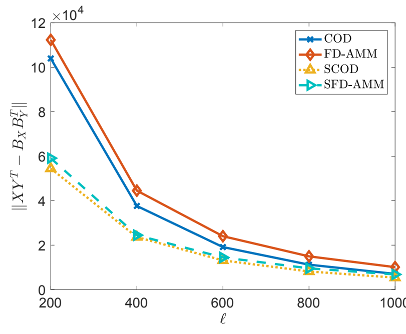
In the previous sections, to control the failure probability of SCOD, we have used VSI in line 6 of Algorithm 4. However, in practice, we find that directly utilizing SI is enough to ensure the accuracy of SCOD. Therefore, in the experiments, we implement SCOD by replacing the original line 6 of Algorithm 4 with

$$C_X, C_Y = \mathrm{SI}(S_X, S_Y, \ell, 1/10).$$

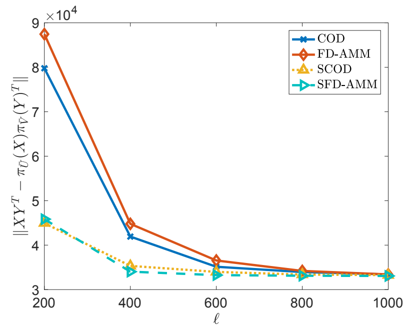
Note that a similar strategy has also been adopted by Ghashami, Liberty, and Phillips (2016) in the implementation of SFD.

In all experiments, each algorithm will receive two matrices $X \in \mathbb{R}^{m_x\times n}$ and $Y \in \mathbb{R}^{m_y\times n}$, and then output two matrices $B_X \in \mathbb{R}^{m_x\times\ell}$ and $B_Y \in \mathbb{R}^{m_y\times\ell}$. We adopt the approximation error $\|XY^T - B_X B_Y^T\|$ and the projection error $\|XY^T - \pi_{\bar{U}}(X)\pi_{\bar{V}}(Y)^T\|$ to measure the accuracy of

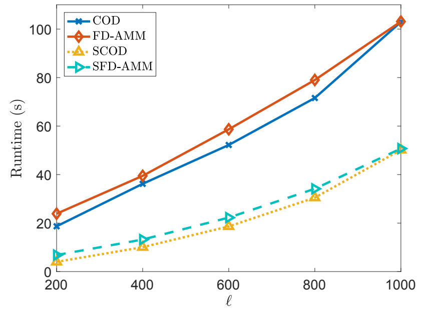
[3]We have also tried Gaussian random projection, the results of which are very close to RP. So, those results are omitted.



(a) Approximation Error

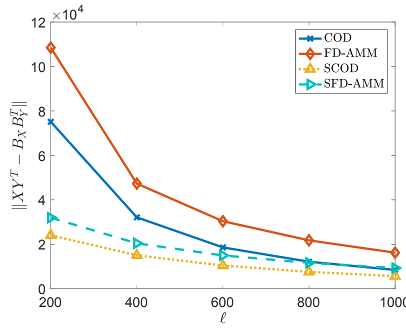


(b) Projection Error

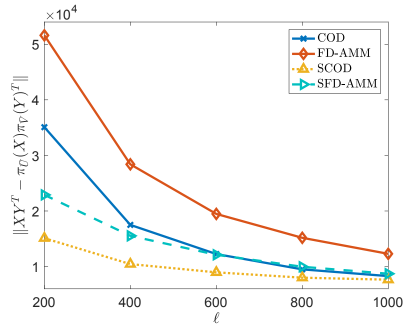


(c) Runtime

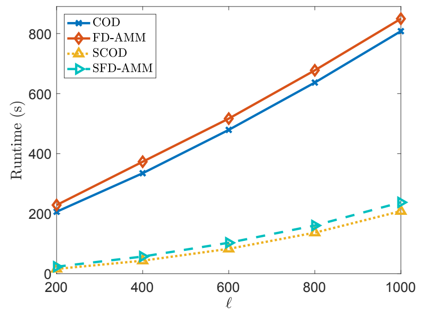
Figure 3: Experimental results among different sketch size on NIPS conference papers dataset.



(a) Approximation Error



(b) Projection Error



(c) Runtime

Figure 4: Experimental results among different sketch size on MovieLens 10M dataset.

each algorithm, where $\bar{U} \in \mathbb{R}^{m_x \times k}, \bar{V} \in \mathbb{R}^{m_y \times k}$ and we set $k = 200$. Furthermore, we report the runtime of each algorithm to verify the efficiency of our SCOD. Because of the randomness of SCOD, SFD-AMM, CS, RP and Hashing, we report the average results over 50 runs.

Results

Fig. 1 and 2 show the results of different algorithms among different ℓ on the synthetic datasets. First, from the comparison of runtime, we find that our SCOD is significantly faster than COD, FD-AMM and RP among different ℓ. Moreover, with the increase of ℓ, the runtime of our SCOD increases more slowly than that of COD, FD-AMM and RP, which verifies the time complexity of our SCOD. Although CS and Hashing are faster than our SCOD, their accuracy is far worse than that of our SCOD. Second, in terms of approximation error and projection error, our SCOD matches or improves the performance of COD among different ℓ. The improvement may be due to the fact that our SCOD performs fewer shrinkage than COD, which is the source of error. We note that a similar result happened in the comparison between FD and SFD by Ghashami, Liberty, and Phillips (2016). Third, our SCOD outperforms other baselines including FD-AMM, SFD-AMM, CS and RP.

Fig. 3 and 4 show the results of SCOD, COD, FD-AMM and SFD-AMM among different ℓ on the real datasets. The results of CS, RP and Hashing are omitted, because they are much worse than SCOD and other baselines. Compared with COD and FD-AMM, we again find that our SCOD is faster and achieves a better performance among different ℓ. Moreover, our SCOD is slightly faster than SFD-AMM on both real datasets, and outperforms SFD-AMM in Fig. 3(a), 4(a) and 4(b). In Fig. 3(b), the performance of SCOD is very close to that of SFD-AMM.

Conclusions

In this paper, we propose SCOD to reduce the time complexity of COD for approximate multiplication of sparse matrices with the $O\left((m_x + m_y + \ell)\ell\right)$ space complexity. In expectation, the time complexity of our SCOD is $\widetilde{O}\left((\mathrm{nnz}(X) + \mathrm{nnz}(Y))\ell + n\ell^2\right)$, which is much tighter than $O\left(n(m_x + m_y + \ell)\ell\right)$ of COD for sparse matrices. Furthermore, the theoretical guarantee of our algorithm is almost the same as that of COD up to a constant factor. Experiments on both synthetic and real datasets demonstrate the advantage of our SCOD for handling sparse matrices.

Acknowledgments

This work was partially supported by the National Key R&D Program of China (2017YFB1002201), NSFC-NRF Joint Research Project (61861146001), and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

Chen, X.; Liu, H.; and Carbonell, J. G. 2015. Structured Sparse Canonical Correlation Analysis. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, 199–207.

Chu, W.; Li, L.; Reyzin, L.; and Schapire, R. E. 2011. Contextual Bandits with Linear Payoff Functions. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, 208–214.

Clarkson, K. L.; and Woodruff, D. P. 2013. Low Rank Approximation and Regression in Input Sparsity Time. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, 81–90.

Cohen, M. B.; Nelson, J.; and Woodruff, D. P. 2015. Optimal Approximate Matrix Product in Terms of Stable Rank. *ArXiv e-prints* arXiv:1507.02268.

Dhillon, I. S.; and Modha, D. S. 2001. Concept Decompositions for Large Sparse Text Data Using Clustering. *Machine learning* 42: 143–175.

Drineas, P.; Kannan, R.; and Mahoney, M. W. 2006. Fast Monte Carlo Algorithms for Matrices I: Approximating Matrix Multiplication. *SIAM Journal on Computing* 36(1): 132–157.

Eriksson-Bique, S.; Solbrig, M.; Stefanelli, M.; Warkentin, S.; Abbey, R.; and Ipsen, I. C. F. 2011. Importance Sampling for a Monte Carlo Matrix Multiplication Algorithm, with Application to Information Retrieval. *SIAM Journal on Scientific Computing* 33(4): 1689–1706.

Ghashami, M.; Liberty, E.; and Phillips, J. M. 2016. Efficient Frequent Directions Algorithm for Sparse Matrices. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 845–854.

Ghashami, M.; Liberty, E.; Phillips, J. M.; and Woodruff, D. P. 2016. Frequent Directions: Simple and Deterministic Matrix Sketching. *SIAM Journal on Computing* 45(5): 1762–1792.

Ghashami, M.; and Phillips, J. M. 2014. Relative Errors for Deterministic Low-rank Matrix Approximations. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, 707–717.

Halko, N.; Martinsson, P.-G.; and Tropp, J. A. 2011. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Review* 53(2): 217–288.

Hazan, E.; Agarwal, A.; and Kale, S. 2007. Logarithmic Regret Algorithms for Online Convex Optimization. *Machine Learning* 69(2): 169–192.

Hotelling, H. 1936. Relations Between Two Sets of Variate. *Biometrika* 28: 321–377.

Kuzborskij, I.; Cella, L.; and Cesa-Bianchi, N. 2019. Efficient Linear Bandits through Matrix Sketching. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, 177–185.

Liberty, E. 2013. Simple and Deterministic Matrix Sketching. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 581–588.

Luo, L.; Chen, C.; Xie, G.; and Ye, H. 2020. Revisiting Co-Occurring Directions: Sharper Analysis and Efficient Algorithm for Sparse Matrices. *ArXiv e-prints* arXiv:2009.02553.

Magen, A.; and Zouzias, A. 2011. Low Rank Matrix-valued Chernoff Bounds and Approximate Matrix Multiplication. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, 1422–1436.

Mroueh, Y.; Marcheret, E.; and Goel, V. 2017. Co-Occuring Directions Sketching for Approximate Matrix Multiply. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 567–575.

Musco, C.; and Musco, C. 2015. Randomized Block Krylov Methods for Stronger and Faster Approximate Singular Value Decomposition. In *Advances in Neural Information Processing Systems 28*, 1396–1404.

Naseem, I.; Togneri, R.; and Bennamoun, M. 2010. Linear Regression for Face Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32(11): 2106–2112.

Perrone, V.; Jenkins, P. A.; Spanó, D.; and Teh, Y. W. 2017. Poisson Random Fields for Dynamic Feature Models. *Journal of Machine Learning Research* 18: 1–45.

Rokhlin, V.; Szlam, A.; and Tygert, M. 2009. A Randomized Algorithm for Principal Component Analysis. *SIAM Journal on Matrix Analysis and Applications* 31(3): 1100–1124.

Sarlos, T. 2006. Improved Approximation Algorithms for Large Matrices via Random Projections. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, 1–10.

Wan, Y.; Wei, N.; and Zhang, L. 2018. Efficient Adaptive Online Learning via Frequent Directions. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2748–2754.

Wan, Y.; and Zhang, L. 2018. Accelerating Adaptive Online Learning by Matrix Approximation. In *Proceedings of the 22nd Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 405–417.

Wan, Y.; and Zhang, L. 2020. Approximate Multiplication of Sparse Matrices with Limited Space. *ArXiv e-prints* arXiv:2009.03527.

Witten, R.; and Candès, E. J. 2014. Randomized Algorithms for Low-rank Matrix Factorizations: Sharp Performance Bounds. *Algorithmica* 31(3): 1–18.

Woodruff, D. P. 2014. Sketching as a Tool for Numerical Linear Algebra. *Foundations and Trends in Machine Learning* 10(1–2): 1–157.

Ye, Q.; Luo, L.; and Zhang, Z. 2016. Frequent Direction Algorithms for Approximate Matrix Multiplication with Applications in CCA. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, 2301–2307.

Zhang, L.; Yang, T.; Jin, R.; Xiao, Y.; and Zhou, Z.-H. 2016. Online Stochastic Linear Optimization under One-bit Feedback. In *Proceedings of the 33rd International Conference on Machine Learning*, 392–401.

Zhang, L.; Yang, T.; Yi, J.; Jin, R.; and Zhou, Z.-H. 2017. Improved Dynamic Regret for Non-degenerate Functions. In *Advances in Neural Information Processing Systems 30*, 732–741.

Zhang, S.; Yao, L.; and Xu, X. 2017. AutoSVD++: An Efficient Hybrid Collaborative Filtering Model via Contractive Auto-encoders. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 957–960.