Journal of Mathematical Modeling
Vol. 10, No. 2, 2022, pp. 263-278. Research Article

JMM

Accurate and fast matrix factorization for low-rank learning

Reza Godaz[†], **Reza Monsefi**[†*], **Faezeh Toutounian**[‡], **Reshad Hosseini**[§]

[†] *Department of Computer Engineering, Faculty of Engineering, Ferdowsi University of Mashhad, Mashhad, Iran*
[‡] *Department of Applied Mathematics, Faculty of Mathematical Sciences, Ferdowsi University of Mashhad, Mashhad, Iran*
[§] *School of ECE, College of Engineering, University of Tehran, Tehran, Iran*
Email(s): reza.godaz@mail.um.ac.ir, monsefi@um.ac.ir, toutouni@math.um.ac.ir, reshad.hosseini@ut.ac.ir

Abstract. In this paper, we tackle two important problems in low-rank learning, which are partial singular value decomposition and numerical rank estimation of huge matrices. By using the concepts of Krylov subspaces such as Golub-Kahan bidiagonalization (GK-bidiagonalization) as well as Ritz vectors, we propose two methods for solving these problems in a fast and accurate way. Our experiments show the advantages of the proposed methods compared to the traditional and randomized singular value decomposition methods. The proposed methods are appropriate for applications involving huge matrices where the accuracy of the desired singular values and also all of their corresponding singular vectors are essential. As a real application, we evaluate the performance of our methods on the problem of Riemannian similarity learning between two different image datasets of MNIST and USPS.

Keywords: Krylov subspace, Ritz vectors, Golub-Kahan bidiagonalization, Riemannian optimization, low-rank learning.
AMS Subject Classification 2010: 34A34, 65L05.

1 Introduction

Singular value decomposition (SVD) [13, 14] is an important matrix factorization method with a wide variety of applications in various sciences such as mathematics, artificial intelligence and physics. Common methods for solving SVD can be used when the input matrix is small or at most medium size, but they can not be used (or require excessive execution time) for problems that involve huge matrices (matrices with more than $1e8$ entries). There are some methods that use randomization and sampling to tackle this problem [15, 20, 21]. For example, fast SVD algorithms of [15] are based on randomization

[*] Corresponding author.

Received: 15 June 2021 / Revised: 31 August 2021/ Accepted: 31 August 2021
DOI: 10.22124/jmm.2021.19892.1723

and sampling, and have strong theoretical guarantees. These methods are currently used in various fields like social network data analysis [30]. In sampling and randomization methods, the large input matrix is converted into a smaller matrix considering a predefined error, and then one of the standard SVD methods can be applied. As a result, due to the use of the standard SVD, these methods are obliged to find a trade-off between determining the column sampling rate (that is related to the accuracy), and the execution time. Our experiments show that the eigenvalues and corresponding eigenvectors obtained by these algorithms are not accurate, although the predefined errors are maintained (see Subsection 5.1).

A fundamental problem in machine learning related to SVD is modeling vast amounts of data using a low-dimensional representation. Processing raw data in typical ways is time-consuming and not always feasible, especially when the data are high dimensional. Representing data in a suitable lower-dimensional form can alleviate challenges of computational complexity, high memory usage, model compression, and noisy environments [7]. The reduced dimensional form of the data can be used in a wide variety of applications such as neural language processing [12], video and image processing [35], recommender systems [7,32], multidimensional scaling [3,26], collaborative filtering and genomics data in bioinformatics [5,27]. It has been shown that in huge matrices the numerical rank of the matrix grows logarithmically with its dimensions [31]. The rank on the data matrix has different interpretations in different applications. For example in latent semantic analysis, the rank can be interpreted as the number of concepts related to a document. In recommender systems, the rank of a user-rating matrix corresponds to a small set of typical users [19].

For problems involving factorization of huge matrices, there is currently a significant gap between being fast, and having high level of accuracy. In the current work, the aim is to bridge this gap by using the well-known Krylov subspace methods and Ritz vectors [13,28]. In what follows, our aim is to pursue the following two goals: (i) to find a fixed number of singular values and corresponding singular vectors of a huge matrix in order to transform the matrix into a low-rank form accurately with moderate time and memory usage, (ii) to approximate the rank of a huge matrix in a fast and accurate way.

The remainder of this article is organized as follows. Section 2 contains a brief review of the SVD problem, and the basic concepts such as Ritz vectors and GK-bidiagonalization. Our algorithm for accurate and fast SVD is the focus of Section 3. In this section, an algorithm to determine an accurate numerical rank of a matrix is also presented. Section 4 introduces an application of the proposed method in Riemannian similarity learning (RSL). In Section 5, we evaluate the efficiency of the proposed algorithms on several numerical examples. Finally, we make some concluding remarks in Section 6.

2 Background

In this section, we first give an overview of SVD and randomized methods to solve SVD. Then, we introduce Ritz vectors and GK-bidiagonalization, that we show in the next section (our proposed algorithms) that it can be used for approximate computations of the largest singular values and the rank of a matrix.

2.1 Singular value decomposition

SVD is a well-known mathematical method that has a wide variety of real world applications in machine learning and mathematical problems. SVD of matrix A with size $m \times n$ and rank r is given by [13, 14]:

$$A = U\Sigma V^T, \qquad \sum = \begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix}, \tag{1}$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are matrices with orthogonal columns, and $D = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_r)$ is a diagonal matrix. Here, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ denote $r \leq \mathbf{min}(m,n)$ nonzero singular values of A. The computational complexity of traditional SVD methods for matrix $A \in \mathbb{R}^{m\times n}$ is $\mathscr{O}(mn\,\mathbf{min}(m,n))$. An important use of this algorithm in learning problems is related to finding a low-rank form of the data matrix or a parameter matrix, where we are looking for k singular vectors associated with the k largest singular values of the matrix.

2.1.1 Randomization-based methods

Randomized methods [15, 20] can make it possible to compute singular values and corresponding singular vectors of huge matrices in a reasonable amount of time. These methods usually use Eckart-Young theorem [29] to find the rank of the input matrix, and thereby determine the required number of sampled columns (see Section 4 in [20] and also Section 10.2 in [15]). In [15], the authors added a fixed number (called oversampling parameter) to this theoretically-driven number to get the final sampling rate.

Experiments of Section 5 demonstrate that when the input matrix is huge, setting a small value for oversampling parameter (for example 10 as suggested by authors in [15]) results in a fast execution time but inaccurate results; whereas fixing a larger value for it implies finding more accurate results, but the execution time will inevitably be longer. Based on the results given in [15] (and also our experiments), certain factors affect this oversampling parameter, such as the dimension, the singular values, and the numerical rank of the input matrix. Since both the numerical rank and the singular values of the matrix are unknown before setting the oversampling parameter (since they are costly to be computed), the resulting decomposition is not very accurate in huge matrices having a large numerical rank.

2.2 Golub-Kahan bidiagonalization for computing Ritz Vectors

We first recall the definition of a Ritz pair [28]. Then, we describe the GK-bidiagonalization process for computing the l largest Ritz values and the corresponding Ritz vectors of A^TA and AA^T [2, 13, 28].

Definition 1. *Consider a k-dimensional subspace $\mathscr{W}$ of $\mathbb{C}^n$. Given a matrix $B \in \mathbb{C}^{n\times n}, \tilde{\theta} \in \mathbb{C}$, and $\tilde{y} \in \mathscr{W}$, $(\tilde{\theta}, \tilde{y})$ is a Ritz pair of B with respect to $\mathscr{W}$ if and only if*

$$B\tilde{y} - \tilde{\theta}\tilde{y} \perp \mathscr{W},$$

or equivalently, for the canonical scalar product,

$$\forall w \in \mathscr{W}, \quad w^H(B\tilde{y} - \tilde{\theta}\tilde{y}) = 0.$$

We call $\tilde{y}$ a Ritz vector associated with the Ritz value $\tilde{\theta}$.

Let $W_k = [w_1, w_2, \dots, w_k] \in \mathbb{R}^{n\times k}$ be an orthogonal basis of subspace $\mathscr{W}$. Using Definition 1, the Ritz pairs $(\tilde{\theta}_i, \tilde{y}_i), i = 1, 2, \dots, k$, of A^TA and AA^T can be obtained by solving the following small eigenvalue problems

$$W_k^T A^T A W_k g_i = \tilde{\theta}_i g_i, \quad \tilde{y}_i = W_k g_i, \quad i = 1, 2, \dots, k, \tag{2}$$

and

$$W_k^T AA^T W_k g_i = \tilde{\theta}_i g_i, \quad \tilde{y}_i = W_k g_i, \quad i = 1, 2, \ldots, k, \tag{3}$$

respectively. As it is known [25, Pages 215-217], this procedure is optimal in a global sense and is optimal for exterior eigenvalues.

We begin by recalling some necessary properties of the GK-bidiagonalization process [13] which is one of the well known algorithms for computing the orthonormal bases for Krylov subspaces

$$\begin{aligned} \mathscr{K}_m(AA^T, q_1) &= \operatorname{span}\{q_1, (AA^T)^1 q_1, \ldots, (AA^T)^k q_1\}, \\ \mathscr{K}_m(A^T A, p_1) &= \operatorname{span}\{p_1, (A^T A)^1 p_1, \ldots, (A^T A)^k p_1\}, \end{aligned} \tag{4}$$

with $q_1 = q_1/\|q_1\|$ and $p_1 = A^T q_1/\|A^T q_1\|$, where $q_1 \in \mathbb{R}^m$ is a nonzero arbitrary vector and $\|\cdot\|$ denotes the Euclidean vector norm or associated matrix norm.

Assuming that A has full column rank and $k \ll \mathbf{min}(m,n)$, the GK-bidiagonalization process [13] which is called the procedure Bidiag 1 in [24], reduces A to its lower bidiagonal form. Procedure Bidiag 1 with starting vector q_1 can be described as follows:

$$\beta_1 q_1 = q_1, \quad \alpha_1 p_1 = A^T q_1, \tag{5}$$

$$\left.\begin{aligned} \beta_{i+1} q_{i+1} &= A p_i - \alpha_i q_i, \\ \alpha_{i+1} p_{i+1} &= A^T q_{i+1} - \beta_{i+1} p_i, \end{aligned}\right\} \quad i = 1, 2, \ldots. \tag{6}$$

The scalars $\alpha_i \geq 0$ and $\beta_i \geq 0$ are chosen such that $\|q_i\|_2 = \|p_i\|_2 = 1$. Together with the definitions

$$Q_k = [q_1, q_2, \ldots, q_k], \quad P_k = [p_1, p_2, \ldots, p_k],$$

and

$$B_{k+1,k} = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_k & \alpha_k \\ & & & \beta_{k+1} \end{bmatrix}, \tag{7}$$

the recurrence relations (5) and (6) can be rewritten as the following equations:

$$Q_{k+1}(\beta_1 e_1) = q_1, \tag{8}$$

$$AP_k = Q_{k+1} B_{k+1,k}, \tag{9}$$

$$A^T Q_{k+1} = P_k B_{k+1,k}^T + \alpha_{k+1} p_{k+1} e_{k+1}^T, \tag{10}$$

where e_1 and e_{k+1} denote the first and the $(k+1)$st columns of an identity matrix of order $k+1$, respectively. In exact arithmetic, we have

$$Q_{k+1}^T Q_{k+1} = I, \tag{11}$$

$$P_k^T P_k = I, \tag{12}$$

where I is the identity matrix. The columns of Q_k and P_k are orthonormal bases for the Krylov subspaces defined in (4).

The Golub-Kahan bidiagonalization process is outlined in Algorithm 1. The vectors $q_{k'+1}$ and $p_{k'+1}$ in lines 6 and 13 of the algorithm are re-orthogonalized to preserve orthogonality. Additional details of the Golub-Kahan bidiagonalization process can be found in [24].

Algorithm 1 GK-bidiagonalization and numerical rank estimation

Inputs:

$A \in \mathbb{R}^{m\times n}$,

k: number of iterations ($k \leq \mathbf{min}(m,n)$),

$\varepsilon \in \mathbb{R}$.

Outputs:

$k' = \mathbf{min}(k,$ the approximate numerical rank of A$)$,

$B_{k'+1,k'} \in \mathbb{R}^{(k'+1)\times(k')}$,

$P_{k'} \in \mathbb{R}^{n\times k'}$,

$Q_{k'+1} \in \mathbb{R}^{m\times(k'+1)}$,

```
$q_1 \sim \mathcal{N}(2,1)^{m\times 1}$ , $\beta_1 = \|q_1\|$ , $q_1 = q_1/\beta_1$,$Q_1 = q_1$
$p_1 = A^T q_1$ , $\alpha_1 = \|p_1\|$ , $p_1 = p_1/\alpha_1$ , $P_1 = p_1$
$k' = 0$
while $k' < k$ do
    $q_{k'+1} = Ap_{k'} - q_{k'}\alpha_{k'}$
    $q_{k'+1} = q_{k'+1} - Q_{(1:k')}(Q^T_{(1:k')}q_{k'+1})$
    $\beta_{k'+1} = \|q_{k'+1}\|$
    $q_{k'+1} = q_{k'+1}/\beta_{k'+1}$ , $Q_{k'+1} = [Q_{k'}$ , $q_{k'+1}]$
    if $\|q_{k'+1}\| < \varepsilon$ do
        break
    end if
    $p_{k'+1} = A^T q_{k'+1} - p_j\beta_{k'+1}$
    $p_{k'+1} = p_{k'+1} - P_{(1:k')}(P^T_{(1:k')}p_{k'+1})$
    $\alpha_{k'+1} = \|p_{k'+1}\|$ , $p_{k'+1} = p_{k'+1}/\alpha_{k'+1}$
    $P_{k'+1} = [P_{k'}, p_{k'+1}]$
    $k' = k' + 1$
end while
```

3 Accurate and fast SVD (F-SVD) algorithm

The largest Ritz values and the corresponding Ritz vectors of A^TA and AA^T can be used for computing the approximations of the l singular triplets $\{\sigma_i, u_i, v_i\}_{i=1}^{l}$ associated with the l largest singular values of A. The following results show that GK-bidiagonalization (Algorithm 1) can be used for obtaining the Ritz values and corresponding Ritz vectors of A^TA and AA^T.

Multiplying the second equation in (8) by A^T yields

$$A^TAP_k = P_kB^T_{k+1,k}B_{k+1,k} + \alpha_{k+1}\beta_{k+1}p_{k+1}e^T_k. \tag{13}$$

We wish to find approximate eigenvectors of A^TA from the subspace spanned by the columns of P_k. To compute the approximate eigenvectors of A^TA, by using relation (13), the small eigenvalue problem posed at (2) can be written as follows

$$B^T_{k+1,k}B_{k+1,k}g_i = \tilde{\theta}_i g_i, \quad \tilde{v}_i = P_kg_i, \quad i = 1,2,\ldots,k. \tag{14}$$

By using (1), the approximate right singular vector $\tilde{u}_i$, $i = 1, 2, \ldots, k$, can be computed as follows:

$$\tilde{u}_i = \frac{1}{\sigma_i} A \tilde{v}_i^T, \quad i = 1, 2, \ldots, k. \tag{15}$$

The resulting algorithm is summarized as Algorithm 2.

Algorithm 2 Accurate and fast SVD (F-SVD)

Inputs:

$A \in \mathbb{R}^{m\times n}$,
q is arbitrary,
k: number of the iterations in Algorithm 1,
r: number of desired larger eigen triplets $\{\sigma_i, u_i, v_i\}$.

Output:

r desired larger eigen triplets $\{\sigma_i, u_i, v_i\}$ of A.

```
1: Set $k$ as number of iterations and run Algorithm 1 to find $B_{k'+1,k'}$,$P_{k'}$,$Q_{k'+1}$.
2: Find eigen decomposition of $(B^T_{k'+1,k'} B_{k'+1,k'})$ for $V_1$ (eigenvectors) and $S_1$ (eigenvalues).
3: $V_2 = P_{k'} V_1$.
4: Select $r$ larger eigenvalues of $S_1$ and corresponding eigenvectors from $V_2$ as $\Sigma_1$ and $V_r$, respectively.
5: $\Sigma_r = \sqrt{\Sigma_1}$.
6: for $i = 1$ to $r$ do:
7:     $U_r[:, i] = 1/\sigma_i A V_r[:, i]$
8: end for
9: Return $U_r$, $\Sigma_r$ and $V_r$.
```

3.1 Complexity analysis and comparison with randomized SVD [15]

The maximal computational complexity of the first three lines of Algorithm 1 is related to computing $A^T q_1$ that is $\mathscr{O}(mn)$ flops. Computing $Ap_{k'}$ in line 5, and also $A^T q_{k'+1}$ in line 12, both takes $\mathscr{O}(mnk')$ flops (concerning the loop), and it is $\mathscr{O}(mk'^2)$ and $\mathscr{O}(nk'^2)$ flops for lines 6 and 13 respectively. Therefore the overall computational complexity involves $\mathscr{O}(mnk' + (m+n)k'^2)$ flops.

With respect to the memory requirements of Algorithm 1, in addition to the input matrix that can be sparse or dense, we consider the P and Q matrices inside the algorithm each require memory of order $\mathscr{O}((m+n)k')$. Because matrix B is a bidiagonal matrix, two vectors of length k' can be used for it. As a result the overall memory usage of Algorithm 1 is $\mathscr{O}((m+n+2)k')$ which is gain of order $\mathscr{O}((m+n)k')$.

The first part of Algorithm 2 involves computing $B_{k'+1,k'}$, $P_{k'}$, and $Q_{k'+1}$ (using Algorithm 1) which has computational complexity $\mathscr{O}(mnk' + (m+n)k'^2)$ flops. Then, the eigen decomposition of the matrix $B^T_{k'+1,k'} B_{k'+1,k'}$ in the second step is computed. The computational complexity of this computation, in general requires at most $\mathscr{O}(k'^3)$ flops. In our algorithms, because the matrix $B^T_{k'+1,k'} B_{k'+1,k'}$ is a small tridiagonal matrix, the computational complexity of this part of the algorithm is close to $\mathscr{O}(k'^2)$. We also have a matrix multiplication in step 3 that involves $\mathscr{O}(mk'^2)$ flops. In steps 6 and 7, there is a loop which requires $\mathscr{O}(mnr)$ flops. As a result, the overall complexity of this algorithm amounts to

$\mathcal{O}(mn(k'+r)+(m+n)k'^2)$ flops. Using the assumption $k', r \ll min(m,n)$ (that is correct when the numerical rank of the input matrix is small) the computational complexity of Algorithm 2 is $\mathcal{O}(mnk')$.

Suppose that $X \in \mathbb{R}^{m\times n}$ and we are looking for a low-rank form for X with the numerical rank k. Using relation (1), the computational complexity of the traditional SVD is $\mathcal{O}(mn^2), n < m$ and $\mathcal{O}(mn\log(l)+l^2(m+n))$, $l = k+p$ for the randomized SVD (R-SVD) [15]. In [15], the authors assumed that the oversampling parameter p is a small value that can be ignored. But for the matrices where the decay of the singular values is slow, the oversampling parameter is not a small value and cannot be ignored. In such situations, the second term of the computational complexity of the randomized algorithm becomes dominant. Therefore its computational complexity approaches that of the traditional method of SVD.

As we observe, the standard SVD, R-SVD, and F-SVD methods can be used for determining the k largest singular values of a matrix, but in different qualities. The standard SVD computes a complete singular decomposition and the results are accurate, but it is not reasonable when $k \ll \min(m,n)$ because of memory and time requirements. The R-SVD method, as explained before, selects a subset of columns of A and performs a complete SVD on this subset. So its time and memory requirements reduce dramatically, but the resulted singular values and corresponding singular vectors are not accurate. For this purpose, by selecting a suitable value for k, the F-SVD method is able to provide the accurate results. In addition, its execution time and memory usage are comparable to those of other methods. The numerical experiments (Section 5) demonstrate the efficiency of the F-SVD Algorithm and confirm that this algorithm quickly executes and provides the accurate results for huge matrices.

3.2 Fast numerical rank estimation

To estimate the numerical rank of an input matrix $A \in \mathbb{R}^{m\times n}$, we use the theory of the Lanczos process in applying the criterion $\|q_{k'+1}\| < \varepsilon$ to terminate the Algorithm 1 [18, 24]. In this formulation ε is a very small positive value determined by the user. This criterion prevents the algorithm from performing extra iterations. As a result, if this condition is satisfied after k' iterations and $k' < k$, then k' can be used as a preliminary estimate of the numerical rank of the matrix, but this estimate is not very accurate (see Algorithm 1). Thereafter, one can obtain an accurate numerical rank using eigen decomposition of the matrix $B^T_{k'+1,k'}B_{k'+1,k'}$ generated in Algorithm 1; see Algorithm 3 for details.

Algorithm 3 Rank determination algorithm

```
Inputs:
    $A \in \mathbb{R}^{m\times n}$,
    $\varepsilon$: a small value (default: 1e-8).
Output:
    $r$: The rank of A.

$k = \min(m,n)$
Set $k$ as the number of iterations and run Algorithm 1 to find $B_{k'+1,k'}$
Find the eigen decomposition of $(B^T_{k'+1,k'}B_{k'+1,k'})$ to determine $S$, the matrix that contains the eigenvalues.
Count the diagonal elements of $S$ that exceed $\varepsilon$ as the accurate numerical rank $r$.
```

4 An application: Riemannian similarity learning

There are a wide variety of learning problems in which the parameters of interest lie on a Riemannian manifold, such as principal components analysis (PCA) [33] and robust PCA [8, 34], independent component analysis (ICA) [23], subspace learning, low-rank matrix completion [32], dictionary learning [11] and Gaussian mixture model [16]. One of the important assumptions in some of these learning problems is that the data are localized around low-dimensional structures, that is we have Riemannian manifold of low-rank matrices. When the optimization constraint is having low-rank matrices, we can use Riemannian optimization algorithm [32]. In-depth discussions of different Riemannian optimization methods with their theoretical analysis can be found in [1, 6].

One of the instances of low-rank learning that Riemannian optimization can be used to solve is Riemannian similarity learning (RSL) that we explain here. Suppose we have pairs of data from following two domains

$$\mathscr{D}_X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots\}, \;\; \boldsymbol{x_i} \in \mathbb{R}^{d_1}, \tag{16a}$$

$$\mathscr{D}_V = \{\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots\}, \;\; \boldsymbol{v_j} \in \mathbb{R}^{d_2}, \tag{16b}$$

and the task is estimating a function $\hat{y} : \mathscr{D}_X \times \mathscr{D}_V \to \mathbb{R}$ that measures the similarity of these pair of data. The training data consists of n tuples $(\boldsymbol{x_i}, \boldsymbol{v_i}, y_i)$, $i \in \{1, 2, \ldots, n\}$, where $\boldsymbol{x_i} \in \mathscr{D}_X$, $\boldsymbol{v_i} \in \mathscr{D}_V$ and y_i is a value measuring the similarity of the data pair $(\boldsymbol{x_i}, \boldsymbol{v_i})$. In [9, 10], the following parametric function was introduced for measuring the similarity:

$$f_W : (\boldsymbol{x}, \boldsymbol{v}) \mapsto \boldsymbol{x}^T \boldsymbol{W} \boldsymbol{v}. \tag{17}$$

In [22], similar cost as that of (17) was used for similarity learning but they induces low-rank constraint on W to regularize their model.

In some problems, $y_i \in \{-1, 1\}$ depending on whether the two samples are similar or not. Therefore, similarity learning can be cast as the following problem:

$$\min_{W \in \mathscr{M}_r} f(W) = \frac{1}{n} \sum_{i=1}^{n} l(f_W(\boldsymbol{x}_i, \boldsymbol{v}_i), y_i), \tag{18}$$

where l is a loss function such as hinge or cross-entropy loss function, and $\mathscr{M}_r = \{W \in \mathbb{R}^{d_1 \times d_2} : \text{rank}(W) = r\}$. Since the low-rank constraint can be seen as a Riemannian manifold, this similarity learning problem can be called Riemannian similarity learning.

To solve (18), one of the different types of Riemannian gradient descent methods presented in [4, 17, 34] can be used. We assume that the rank is significantly smaller than the matrix dimensions, i.e. $r \ll \mathbf{min}(d_1, d_2)$. In addition, to improve the performance, we use Algorithm 2 inside a RSGD instead of standard SVD to solve (18).

4.1 Riemannian stochastic gradient descent

Consider the following minimization problem:

$$\min_{W \in \mathscr{M}_r} f(W), \tag{19}$$

where $\mathscr{M}_r$ is the Riemannian manifold of matrices with rank equal to r, and we use a noisy version of the Riemannian gradient (∇f_t) at step t in the update rule. Riemannian stochastic gradient descent (RSGD) [4] uses the following update rule:

$$W_{t+1} = \mathrm{R}_{W_t}(-\eta_t \nabla f_t), \tag{20}$$

where η_t is the step size at step t, and R_{W_t} is a retraction at W_t. A retraction is a mapping from tangent space at a point to the manifold satisfying certain properties. Indeed for any tangent vector $\xi \in T_W\mathscr{M}_r$, a retraction can be found by solving the following equation [32]

$$R_W(\xi) = \underset{X \in \mathscr{M}_r}{\operatorname{argmin}} \|W + \xi - X\|_F. \tag{21}$$

This equation has a closed form solution as

$$R_W(\xi) = \sum_{i=1}^{r} \sigma_i u_i v_i^T, \tag{22}$$

where $\{\sigma_i, u_i, v_i\}_{i=1}^r$ are the first r singular triplets of SVD in the point $W + \xi$.

Geometric meaning of SVD. Suppose $\mathscr{M}_W := \{W \in \mathbb{R}^{d_1 \times d_2} : W = U\Sigma V^T\}$, where U, Σ, V are matrices defined the same as (1). It was demonstrated in [1, Chapter 3] and [32] that $\mathscr{M}_W$ is a Riemanninan manifold, and the tangent space at point W is defined by

$$T_W\mathscr{M} := \{UMV^T + U_P V^T + UV_P^T\}, \tag{23}$$

where $U_P^T U = 0$ and $V_P^T V = 0$, and M is an arbitrary r by r matrix. The Riemannian metric is defined as $< \eta, \zeta >= \mathrm{tr}(\eta^T \zeta)$, where $\mathrm{tr}(.)$ is the matrix trace.

Riemannian gradient. Assume $\mathrm{Gr}_W^f \in \mathbb{R}^{m \times n}$ represents the Euclidean gradient of the objective function (19) w.r.t to W. The Riemannian gradient, that is computed by the projection of the Euclidean gradient onto the tangent space of $\mathscr{M}_W$, has the following form

$$\mathrm{Grad}_W f = P_U^{\mathscr{H}} \mathrm{Gr}_W^f P_V^{\mathscr{H}} + P_U^{\mathscr{V}} \mathrm{Gr}_W^f P_V^{\mathscr{H}} + P_U^{\mathscr{H}} \mathrm{Gr}_W^f P_V^{\mathscr{V}}, \tag{24}$$

where the shorthand notations $P_X^{\mathscr{H}} := XX^T$ and $P_X^{\mathscr{V}} := I - XX^T$.

RSGD algorithm for RSL. The complete process of the RSGD method that is customized for the Riemannian similarity learning problem is represented in Algorithm 4.

5 Experimental results

In this section, we investigate the effectiveness of Algorithms 1, 2, and 3 which we developed in Python 3.7.6. Our experiments were performed on the google cloud with 16 vCPUs @ 2.2 GHz (8 real cores) and 128GB RAM. To measure execution times and errors, we calculate an average value for the results of five repetitions of each algorithm. Our Python code is available via `https://github.com/rezagodaz/accurate-partial-svd`.

Algorithm 4 Fast Riemannian mini-batch gradient descent algorithm

```
Inputs:
  $X \in \mathbb{R}^{m\times d_1}$, $V \in \mathbb{R}^{n\times d_2}$,
  $K$ : number of iterations, b: mini batch size
  $\eta$: RSGD training rate
  $r$: rank of low-rank manifold $\mathscr{M}$
Outputs:
  $W \in \mathbb{R}^{d_1\times d_2}$ as the optimized parameter matrix,

$W \sim \mathscr{N}(0,1)^{d_1\times d_2}$
For each step $t < K$ do
    $Gr \in \mathbb{R}^{d_1\times d_2} = 0$
    Draw a batch of data tuple $\mathscr{B} = \{(\boldsymbol{x}_i, \boldsymbol{v}_i, y_i)\}_{i=1}^{b}$
    Compute $Gr = \frac{1}{|\mathscr{B}|} \sum_{(\boldsymbol{x}_i,\boldsymbol{v}_i,y_i)\in\mathscr{B}} (\nabla f_W(\boldsymbol{x}_i, \boldsymbol{v}_i))y_i$
    $Gr = Gr - \lambda * W$
    Compute the SVD of $Gr = U_r\Sigma_r V_r^T$ using Algorithm 2 with desired rank $r$
    $Z = U_rU_r^T GrV_rV_r^T + (I - U_rU_r^T)GrV_rV_r^T + U_rU_r^T Gr(I - V_rV_r^T)$
    Compute the SVD of $(W - \eta Z) = U_r\Sigma_r V_r$ using Algorithm 2 with desired rank $r$
    $W_{new} = U_r\Sigma_r V_r$
end For
```

5.1 Comparing accuracy and speed of SVD algorithms

We select two algorithms, traditional SVD and R-SVD [15] for our comparisons because traditional SVD is one of the most accurate methods and R-SVD is one of the fastest choices among the randomized methods. In addition the performance of R-SVD with respect to errors makes it a suitable representative for the various randomized methods. In the experiments to build a synthetic matrix $A \in \mathbb{R}^{m\times n}$, with fixed rank l, we multiplied two matrices $M \in \mathbb{R}^{m\times l}$ and $N \in \mathbb{R}^{l\times n}$. Each instance of M and N was randomly and independently using a Gaussian distribution. The reported errors are defined as the relative error, $err_{rel} = \|A^TU - V\Sigma\|_F/\|\Sigma\|_F$, and the residual error, $err_{res} = \|A - U\Sigma V^T\|_F$, where the matrices U, Σ, and V are the result of a specific SVD algorithm for decomposing A.

The time and accuracy comparison of different SVD algorithms for different random synthetic matrix are shown in Table 1. The size of synthetic matrices vary as included in the first columns of Tables 1a, 1b, and 1c. Table 1a is related to approximate the numerical rank of a huge matrix. The required CPU time of Algorithm 3 is presented in this table. We observe that the proposed Algorithm 3 is fast in determining the numerical rank of the input matrix providing a distinct time advantage over the current practical methods used by Python, which executes traditional partial SVD for huge matrices.

When the true numerical rank of a huge original matrix is small, Algorithm 1 and also Algorithm 3 perform the best. The last column of Table 1a records the number of iterations of Algorithm 3 after terminating based on the condition $\|q_{k'}\| < \varepsilon$ in iteration k', without any predefined parameter or user intervention. This value is our first approximation of the numerical rank of the input matrix.

Considering the complexity analysis of the F-SVD method (mentioned in Section 3.1) and also the experimental results summarized in Table 1b, F-SVD has a computational complexity comparable to that

Table 1: Comparison of various SVD algorithms with respect to execution time (a), estimated numerical rank of the input matrix (b), and the observed residual and relative errors (c). The results are obtained using synthetic matrices of different sizes, all of which have a numerical rank equal to 100. For the randomized SVD algorithm (R-SVD) of [15], we investigate two possible scenarios: (i) knowing the required value of oversampling parameter p (see section 2.1.1), and (ii) using the default value of p by [15]). For (b) and (c), the underlying goal is to determine the 20 dominant triplets of the input matrix using the various possible SVD algorithms and to compare them with respect to execution time and accuracy.

(a)

Input Matrix Size	Rank determination Time (sec)		Alg. 3 Number of Iterations
	SVD	Alg. 3	
$1e3 \times 1e3$	0.17	0.063	102
$1e4 \times 1e3$	0.59	0.641	102
$1e5 \times 1e3$	5.92	6.590	102
$1e4 \times 1e4$	113.52	4.073	104
$1e5 \times 1e4$	379.62	32.550	104
$1e5 \times 2e4$	1854.72	59.137	104
$1e5 \times 3e4$	4788.48	77.533	105
$1e5 \times 8e4$	NA	185.544	105

(b)

Input Matrix size	Time of Algorithms (sec)			
	SVD	F-SVD	R-SVD (default)	R-SVD (oversampled)
$1e3 \times 1e3$	0.33	0.06	0.03	0.10
$1e4 \times 1e3$	1.16	0.53	0.24	0.65
$1e5 \times 1e3$	10.27	5.29	3.15	11.02
$1e4 \times 1e4$	163.06	3.18	2.02	5.64
$1e5 \times 1e4$	856.55	28.60	23.83	52.15
$1e5 \times 2e4$	4520.32	65.84	63.44	102.18
$1e5 \times 3e4$	NA	77.73	69.83	142.45
$1e5 \times 8e4$	NA	203.18	201.97	362.04

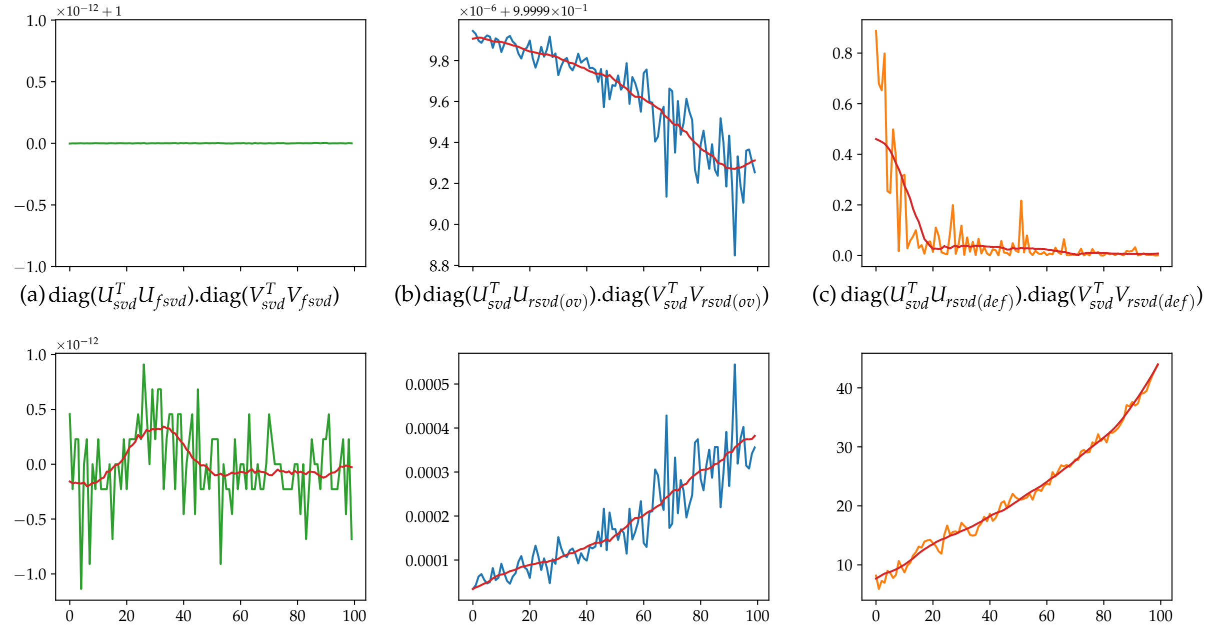
(c)

Input Matrix size	error of SVD		error of F-SVD		error of R-SVD (oversampled)		error of R-SVD (default)	
	Residual	Relative	Residual	Relative	Residual	Relative	Residual	Relative
$1e3 \times 1e3$	**6.97e-12**	1.90e-15	6.77e-11	**7.27e-17**	2656.72	2.28e-15	2664.66	1.35e-15
$1e4 \times 1e3$	**9.89e-12**	2.83e-15	8.01e-11	**7.43e-17**	2660.99	2.69e-15	2664.57	1.56e-15
$1e5 \times 1e3$	**9.97e-12**	2.96e-15	2.24e-10	**7.26e-17**	2646.96	2.36e-15	2665.86	1.68e-15
$1e4 \times 1e4$	**3.20e-11**	3.14e-15	3.92e-10	**8.04e-17**	8783.91	1.86e-15	8810.52	2.00e-15
$1e5 \times 1e4$	**4.51e-11**	4.25e-15	3.73e-10	**8.56e-17**	8779.20	2.03e-15	8616.37	1.72e-15
$1e5 \times 2e4$	**6.69e-11**	4.50e-15	3.07e-10	**7.06e-17**	12494.80	2.03e-15	12543.44	1.80e-15
$1e5 \times 3e4$	NA	NA	2.30e-9	**8.18e-17**	15332.69	2.06e-15	15399.78	1.87e-15
$1e5 \times 8e4$	NA	NA	NA	**7.30e-17**	NA	2.03e-15	NA	1.74e-15

of R-SVD when the decay of matrix singular values is slow. In addition, it can be seen that the execution time of the F-SVD methods is less than that of R-SVD (oversampled) method.

Table 1c shows that Algorithm 2 exhibits the smallest relative error among all executions. It also gives low residual error and fast computation of the singular values and the corresponding singular vectors, especially the smaller singular triplets, which represents the relative superiority of F-SVD over R-SVD (oversampled).Therefore, this algorithm is a suitable choice for use in almost all problem settings.

In Figure 1, we are comparing the quality of SVD triplet obtained by different methods. In this experiment, Algorithm 2 terminates after 550 iterations. For the algorithm R-SVD, the oversampling



(a) diag($U_{svd}^T U_{fsvd}$).diag($V_{svd}^T V_{fsvd}$) (b) diag($U_{svd}^T U_{rsvd(ov)}$).diag($V_{svd}^T V_{rsvd(ov)}$) (c) diag($U_{svd}^T U_{rsvd(def)}$).diag($V_{svd}^T V_{rsvd(def)}$)

(d) $\{|\sigma^i_{svd} - \sigma^i_{fsvd}|\}_{i=1}^{100}$ (e) $\{|\sigma^i_{svd} - \sigma^i_{rsvd(ov)}|\}_{i=1}^{100}$ (f) $\{|\sigma^i_{svd} - \sigma^i_{rsvd(def)}|\}_{i=1}^{100}$

Figure 1: Investigating the quality of the singular triplets of our proposed F-SVD (c and d), oversampled R-SVD (e and f) and default R-SVD (e and f) compared to the singular triplets found by traditional SVD. The input matrix is $A \in \mathbb{R}^{1e4 \times 1e4}$ with the numerical rank equal to 1000 and the goal is to determine 100 dominant singular triplets, and their singular values by an SVD algorithm: $A \approx U_{alg} \Sigma_{alg} V_{alg}^T$. In the plots of the first row, we are showing the values of the error vector computed by $\text{diag}(U_{svd}^T U_{alg}).\text{diag}(V_{svd}^T V_{alg})$, where *alg* denotes one of the three SVD methods, *svd* is the standard SVD method used by the Numpy Python library, diag$(\cdot)$ returns diagonal elements of a matrix as a vector, and dot represents element-wise multiplication of two vectors. In the plots of the second row, the values of the absolute difference between dominant singular values $\{|\sigma^i_{svd} - \sigma^i_{alg}|\}_{i=1}^{100}$ are shown. The red curves in the plots is the smoothed version of the original curves.

parameter was set to 800 (p=800 thus l=900). For the plots in the first row of this figure, a value close to 1.0 means that the corresponding vectors computed were satisfactorily accurate. It can be seen that Algorithm 2 outperforms its R-SVD competitor with respect to the accuracy for both the singular values as well as the singular vectors.

5.2 RSL application

In this section, we evaluate Algorithm 4 in the task of similarity learning for classification. We use two famous datasets MNIST (used as the source domain $\mathscr{D}_X$) and USPS (used as the target domain $\mathscr{D}_Y$). Both of these datasets include handwritten digits, but with a different number of pixels. Algorithm 4 is used for the task of RSL with the input parameter rank equal to five, and the number of iterations varying from 5000 to 20000. Figure 2 compares the running-time and accuracy results obtained by using the standard SVD and FSVD in the inner loop of Algorithm 4. As it is clear, using FSVD significantly leads to less

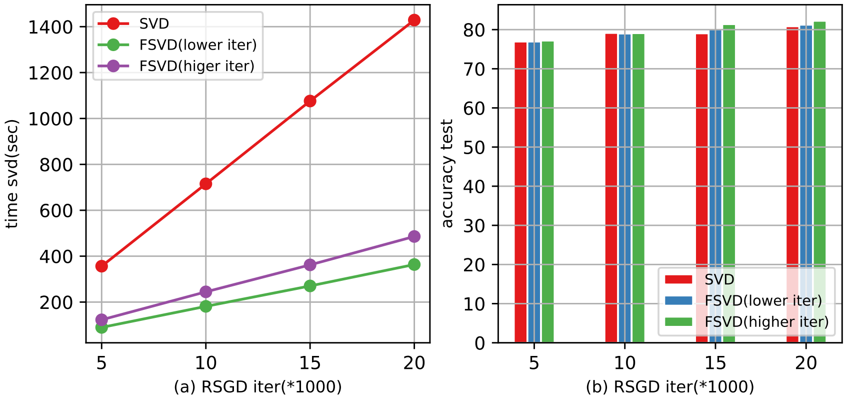


Figure 2: The execution time (a) and the test accuracy (b) of the RSL algorithm between two handwritten digit datasets (MNIST and USPS) for different number of iterations of the RSGD algorithm. Algorithm 4 is used for RSL with three different SVD methods. These three SVD methods are: (i) standard SVD method, (ii) our proposed F-SVD method with setting inner iteration of Algorithm 2 to 20 ("lower iter"), and (iii) F-SVD method with setting inner iteration to 35 ("higher iter").

computation time than the standard SVD, while the same accuracy is obtained.

The results in this figure is the median result for three executions of Algorithm 4. It should be mentioned that Riemannian optimization algorithms such as that of Algorithm 4 need a high level of accuracy for the retraction and the RSVD method can not be used in these types of algorithms.

6 Conclusion

Traditional method for solving SVD are accurate for matrix decomposition that has many applications in different sciences. This algorithm is not a suitable choice for huge matrices because of its computational complexity. Randomized algorithms are a group of methods that tackle this problem by decomposing a smaller matrix made from the main matrix. They successfully reduced the computational complexity of the traditional SVD algorithm and hence can be used with huge input matrices. However, such randomized algorithms may be inaccurate, or may require unknown values for their key parameters. Using key concepts from Krylov subspaces, we have devised SVD algorithms that execute quickly and provide accurate, reliable singular values and their corresponding singular vectors for huge input matrices. As a by-product, we obtained a fast and accurate rank estimation for huge matrices.

Acknowledgements

The authors wishes to thank the references for their insightful suggestions that improved the presentation.

References

[1] P.A. Absil, R. Mahony, R. Sepulchre, *Optimization Algorithms on Matrix Manifolds*, Princeton University Press, 2008.

[2] J. Baglama, L. Reichel, D. Richmond, *An augmented LSQR method*, Numer. Algorithms **64** (2013) 263–293.

[3] B.T. Bartell, G.W. Cottrell, R.K. Belew, *Latent semantic indexing is an optimal special case of multidimensional scaling*, Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval (1992) 161–167.

[4] S. Bonnabel, *Stochastic gradient descent on Riemannian manifolds*, IEEE Trans. Automat. Control **58** (2013) 2217–2229.

[5] J. Bennett, S. Lanning, *The Netflix prize*, Proceedings of KDD cup and workshop, 2007.

[6] N. Boumal, *An introduction to optimization on smooth manifolds*, Available online: http://www.nicolasboumal.net/book, 2020.

[7] L. Cambier, P.A. Absil, *Robust low-rank matrix completion by Riemannian optimization*, SIAM J. Sci. Comput. **38** (2016) S440–S460.

[8] E.J Candès, X. Li, Y. Ma, J. Wright, *Robust principal component analysis?*, J. Assoc. Comput. Math. **58** (2011) 1–37.

[9] G. Chechik, V. Sharma, U. Shalit, S. Bengio, *An online algorithm for large scale image similarity learning*, NIPS'09: Proceedings of the 22nd International Conference on Neural Information Processing Systems (2009) 306–314.

[10] G. Chechik, V. Sharma, U. Shalit, S. Bengio, *Large scale online learning of image similarity through ranking*, J. Mach. Learn. Res. **11** (2010) 11–14.

[11] A. Cherian, S. Sra, *Riemannian Dictionary Learning and Sparse Coding for Positive Definite Matrices*, IEEE Trans. Neural Netw. Learn. Syst. **28** (2015) 2859-2871.

[12] P.S. Dhillon, D.P. Foster, L.H. Ungar, *Eigenwords: Spectral word embeddings*, J. Mach. Learn. Res. **16** (2015) 3035–3078.

[13] G.H. Golub, W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, J. Soc. Ind. Appl. Math. Series B: Numer. Anal. **2** (1965) 205–224.

[14] G.H. Golub, C. Reinsch, *Singular value decomposition and least squares solutions*, Numer. Math. **14** (1971) 134–151.

[15] N. Halko, P.G. Martinsson, J.A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM rev. **53** (2011) 217–288.

[16] R. Hosseini, S. Sra, *Matrix manifold optimization for Gaussian mixtures*, Adv. Neural Inf. Process. Syst. **28** (2015) 910–918.

[17] H. Kasai, H. Sato, B. Mishra, *Riemannian stochastic recursive gradient algorithm with retraction and vector transport and its convergence analysis*, 35th International Conference on Machine Learning, ICML (2018) 2516–2524.

[18] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bur. Stand. **45** (1950) 255–282.

[19] I. Markovsky, *Low-Rank Approximation: Algorithms, Implementation, Applications*, Springer, 2019.

[20] P.G. Martinsson, *Randomized methods for matrix computations*, In The Mathematics of Data (Mahoney, M. W., Duchi, J. and Gilbert, A., eds), Vol. 25 of IAS/Park City Mathematics Series, AMS, pp. 187–229.

[21] P.G. Martinsson, V. Rokhlin, M. Tygert, *A randomized algorithm for the decomposition of matrices*, Appl. Comput. Harmon. Anal. **30** (2011) 47–68.

[22] B. Mishra, R. Sepulchre, *R3MC: A Riemannian three-factor algorithm for low-rank matrix completion*, 53rd IEEE Conference on Decision and Control (2014) 1137–1142.

[23] E. Oja, A. Hyvarinen, *Independent component analysis: Algorithms and applications*, Neural Netw. **13** (2000) 411–430.

[24] C. Paige, M. Saunders, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Software **8** (1982) 43-71.

[25] N.B. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[26] Y. Qiu, H. Jiang, W.K. Ching, *Unsupervised learning framework with multidimensional scaling in predicting epithelial-mesenchymal transitions*, IEEE/ACM transactions on computational biology and bioinformatics, 2020, doi:10.1109/TCBB.2020.2992605.

[27] J.D.M. Rennie, N. Srebro, *Fast maximum margin matrix factorization for collaborative prediction*, Proceedings of the 22nd international conference on Machine learning, (2005) 713–719.

[28] D.C. Sorensen, *Numerical methods for large eigenvalue problems*, Acta Numer. **11** (2002) 519–584.

[29] G. Stewart, *On the early history of the singular value decomposition*, SIAM Rev. **35** (1993) 551-566.

[30] A. Tulloch, *Fast randomized singular value decomposition*, http://research.facebook.com/blog/294071574113354/fast-randomized-svd/, 2014.

[31] M. Udell, A. Townsend, *Why are big data matrices approximately low rank?*, SIAM J. Math. Data Sci. **1** (2019) 144–160.

[32] B. Vandereycken, *Low-rank matrix completion by Riemannian optimization*, SIAM J. Optim. **23** (2013) 1214–1236.

[33] S. Wold, K. Esbensen, P. Geladi, *Principal component analysis*, Chemom. Intell. Lab. Syst. **2** (1987) 37–52.

[34] T. Zhang, Y. Yang, *Robust PCA by manifold optimization*, J. Mach. Learn. Res. **19** (2018) 3101–3139.

[35] X. Zhou, C. Yang, H. Zhao, W. Yu, *Low-rank modeling and its applications in image analysis*, ACM Comput. Surv. **47** (2014) 1–33.