Distributed Singular Value Decomposition Through Least Squares

by

Freddie Zhao

B.S. Computer Science and Engineering and Mathematics, MIT, 2023

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2024

Authored by:	Freddie Zhao Department of Electrical Engineering and Computer Science August 16, 2024
Certified by:	Devavrat Shah Professor of Electrical Engineering and Computer Science, Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

Distributed Singular Value Decomposition Through Least Squares

by

Freddie Zhao

Submitted to the Department of Electrical Engineering and Computer Science
on August 16, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

Singular value decomposition (SVD) is an essential matrix factorization technique that decomposes a matrix into singular values and corresponding singular vectors that form orthonormal bases. SVD has wide-ranging applications from principal component analysis (PCA) to matrix completion and approximation. Methods for computing the SVD of a matrix are extensive and involve optimization algorithms with some theoretical guarantees, though many of these techniques are not scalable in nature. We show the efficacy of a distributed stochastic gradient descent algorithm by implementing parallelized alternating least squares and prove theoretical guarantees for its convergence and empirical results, which allow for the development of a simple framework for solving SVD in a correct, scalable, and easily optimizable manner.

Thesis supervisor: Devavrat Shah
Title: Professor of Electrical Engineering and Computer Science

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Singular value decomposition (SVD) is a matrix factorization technique that decomposes a matrix into singular values and corresponding singular vectors that form orthonormal bases, which allows for dimensionality reduction and matrix approximation. SVD has wide-ranging applications from principal component analysis (PCA) to matrix completion, with various implications in data science and machine learning.

Traditionally, there have been several methods to compute the SVD of an arbitrary matrix. One such classical method is the QR algorithm [1], an algorithm that is suitable for dense matrices. For sparse matrices, other randomized algorithms such as those found in Halko et al. [2] are more efficient.

We focus on the distributed SVD problem as part of a larger algorithm for matrix completion. Matrix completion, the process of estimating unobserved entries of a matrix from noisy observations of other entries, is the problem in statistical data imputation that requires estimating the unobserved entries of a partially observed matrix. The problem has recently seen a massive growth in attention due to its various applications, including fields such as clinical trials [3], improving techniques used in machine learning such as reinforcement learning [4], as well as those in causal inference [5], and in recommender systems [6].

We will consider a scenario in which we have a large amount of computation power and

limited memory, as is true of more modern machines, while the matrix we are computing the SVD for is of low rank, which is quite common in the standard literature of matrix completion [6] [7]. We thus aim to give an efficient distributed algorithm for finding the SVD of a low rank matrix.

1.1 Problem Statement

We focus on a specific formulation of the distributed SVD problem that will help analyze the efficacy of algorithms for our purpose. Specifically, let us suppose that $M \in \mathbb{R}^{d \times d}$ be the square matrix that we aim to recover and factorize. We will also assume that M is low-rank so that the matrix has rank $r \ll d$, which is quite common in the standard of matrix completion. However, unlike much of the standard literature on matrix completion, we will assume that we are given noisy observations of all of the entries of M. We do so to focus on the technique of singular value decomposition instead of attempting to improve matrix completion methods at large. Initially, we will assume that we have access to the true matrix M_{true}, though we will later assume that we will only have access to a noisy observation $M_{\text{obs}} = M_{\text{true}} + M_{\text{err}}$, where M_{err} is a matrix with bounded spectral norm.

Given these assumptions, we aim to compute a diagonal matrix of singular values $\Sigma \in \mathbb{R}^{r \times r}$ and matrices of corresponding left and right singular vectors, $U, V \in \mathbb{R}^{d \times r}$ that minimizes the MSE of the entries of the original matrix to the approximation formed by the singular value and singular vectors. In other words, we aim to find $U, \Sigma, V^\top$ to minimize the error

$$f(U, \Sigma, V^\top) := ||M - U\Sigma V^\top||_F^2.$$

Throughout this document, we will often refer to $f(\cdot, \cdot, \cdot)$ as the objective we are trying to minimize.

In Chapter 4, in order to prove certain theoretical guarantees, we will show that it suffices to consider the case where M is symmetric and that finding solutions in the case where M

is symmetric will allow us to find solutions where the original matrix is asymmetric. Thus, in some cases, we aim to compute a matrix $X \in \mathbb{R}^{d \times r}$ that minimizes the error

$$g(X) := ||M - XX^\top||_F^2 = \sum_{i,j} (M_{ij} - X_i X_j^\top)^2.$$

Our main goal, regardless of the symmetry of M, is to find an algorithm that can empirically compute these singular values and singular vectors within a threshold of tolerance. Due to the size of M and memory constraints, we are often unable to store M in main memory. Thus, we require our algorithms to be distributed in nature with the ability to scale, so that we can run the computations required by the algorithm with limited memory regardless of the size of M.

1.2 Our Contributions

Now that we have described the problem in greater detail, we are ready to state our findings. In Sections 3.2 and 3.3, we introduce algorithms ALS-GD-Distributed and ALS-GD-Fully-Distributed that attempt to solve the problem of distributing the computation of the top singular values and singular vectors with low space. We will prove that the latter is mathematically sound.

We first present the results for the fully distributed gradient descent algorithm. With probability 1 with respect to the random initialization of the singular value and singular vectors, given that the threads execute in an order such that the certain threads known as gradient threads corresponding to the computation of each singular value execute before the first iteration and in between every two consecutive iterations of the main thread, also known as the aggregation thread, we establish that the algorithm correctly finds the r largest singular values and corresponding singular vectors for any constant $r \geq 1$. In addition, even if we only have access to a noisy observation of the true matrix M, given the same conditions as previous, we will be able to find the r largest singular vectors and singular values up to

an error bounded by a constant multiple of the spectral norm of the error matrix. We state these theorems more formally in Chapter 4 as Theorems 4.0.3 and 4.0.4.

We will also show that the algorithms ALS-GD-Distributed and ALS-GD-Fully-Distributed are empirically viable, where we show the correctness and scalability of the distributed algorithms to be able to find the r largest singular values of a matrix M that no longer fits in main memory.

1.3 Notations

Throughout the rest of the document, we will suppose that we fully observe the matrix $M \in \mathbb{R}^{d \times d}$ in the noiseless setting and fully observe $M_{\text{obs}} \in \mathbb{R}^{d \times d}$ as well in the noisy setting such that $M_{\text{obs}} = M_{\text{true}} + M_{\text{err}}$, where $M_{\text{true}} \in \mathbb{R}^{d \times d}$ is the ground truth matrix, generally assumed to be of low rank, and $M_{\text{err}} \in \mathbb{R}^{d \times d}$ is a matrix of bounded spectral norm.

We will also assume that matrix M in the noiseless setting or matrix M_{true} in the noisy setting has true singular value decomposition $U\Sigma V^\top$. Let the true singular values on the diagonal of Σ be $\sigma_{(1)}, \sigma_{(2)}, \ldots, \sigma_{(d)}$ in nonincreasing order, and let the corresponding true singular vectors in U and $V^\top$ be $u_{(1)}, u_{(2)}, \ldots, u_{(d)}$ and $v_{(1)}^\top, v_{(2)}^\top, \ldots, v_{(d)}^\top$, respectively. Additionally, we define U_r to denote the first r columns of U and $V_r^\top$ to denote the first r rows of $V^\top$, while Σ_r denotes the top left $r \times r$ submatrix of Σ, so that $U_r \Sigma_r V_r^\top$ is the rank r approximation of the true singular value decomposition of M.

As we aim to approximate the top r singular values and singular vectors, we will refer to algorithm outputs of these approximations as $\hat{U}_r, \hat{\Sigma}_r$, and $\hat{V}_r^\top$ composed of the singular values $\hat{\sigma}_{(1)}, \hat{\sigma}_{(2)}, \ldots, \hat{\sigma}_{(r)}$ and singular vectors $\hat{u}_{(1)}, \hat{u}_{(2)}, \ldots, \hat{u}_{(r)}$ and $\hat{v}_{(1)}^\top, \hat{v}_{(2)}^\top, \ldots, \hat{v}_{(r)}^\top$. In addition, when running an iterative algorithm, let $x^{(t)}$ be the value of the vector x at time step t of the algorithm.

We now discuss how large data structures are indexed throughout the document. For single entries, we will use subscripts, as we let x_i denote the i^{th} entry in a vector x and let A_{ij}

be the entry in the i^{th} row and j^{th} column of a matrix A. However, when referring to a slice of indices from a vector or matrix, we will use a notation similar to that of Python 3. Namely, for a vector x, let $x[i_{\text{start}} : i_{\text{end}}]$ be the entries of x between indices i_{start} and i_{end}, including index i_{start} and excluding index i_{end}. Similarly, for a matrix M, let $M[i_{\text{start}} : i_{\text{end}}, j_{\text{start}} : j_{\text{end}}]$ be the submatrix of M consisting of rows between indices i_{start} and i_{end}, including index i_{start} and excluding index i_{end}, and columns between indices j_{start} and j_{end}, including index j_{start} and excluding index j_{end}. If we do not give a start or end index, it is assumed that the slice will start from the first entry or end at the last entry, respectively.

The rest of the definitions are of miscellaneous category, which we give here. Let $|\cdot|$ denote the absolute value of a scalar, $||\cdot||$ denote the Euclidean norm of a vector or the spectral norm of a matrix, and let $||\cdot||_F$ denote the Frobenius norm of a matrix. Also, let $[r]$ denote the set $\{1, 2, \ldots, r\}$. Finally, let us write $f(x) = O(g(x))$ for functions f and g if $|f(x)|$ is at most a positive constant multiple of $g(x)$ for all sufficiently large values of x.

1.4 Organization

In Chapter 2, we give related works on SVD, reviewing previous algorithms. We will introduce a variety of algorithms used to solve SVD in certain scenarios where different properties of M can be exploited. However, we will also see that our specific formulation of SVD on a large matrix on a single machine with limited memory and virtually unlimited computation power requires a different approach.

In Chapter 3 of this work, we will first give a simple, centralized alternating least squares algorithm, ALS-GD, as a non-distributed way of minimizing the desired objective when solving for only the largest singular value and corresponding singular vectors. We will use this initial algorithm to motivate the creation of a distributed algorithm, ALS-GD-Distributed, that is very similar and potentially stochastic. Finally, we will show how to extend this algorithm to solve for the top r singular values and singular vectors for any constant r in

our fully distributed algorithm ALS-GD-Fully-Distributed.

Having introduced the algorithms, we will then discuss the theoretical correctness of the distributed algorithms we use. In Chapter 4, we will show that even the fully distributed algorithms are mathematically robust even when we only have noisy observations of the true matrix, as the algorithm will converge to singular values and singular vectors reasonably close to those of the true matrix.

Once we have given proof of the mathematical foundation of the algorithms, in Chapter 5, we test the empirical efficacy of the algorithms. To do so, we describe the settings of our experiment, including the parameters of the algorithms, the data we use, the benchmarks, and the metrics that we plan to analyze, before running the algorithms through a battery of tests with synthetically generated and real matrices. We will then analyze the results of our algorithms, and from these results, in Chapter 6, we conclude the strengths and weaknesses of the distributed algorithms and give possible next steps and optimizations to create a library for running distributed gradient descent via least squares.

Chapter 2

Related Work

Some of the oldest algorithms for SVD are classical algorithms such as the Jacobi method and the QR algorithm [1]. These algorithms do not take advantage of any structure of the matrix M in question and thus are robust when M is a dense matrix. Note that even though the QR algorithm is an eigenvalue algorithm, it is sufficient since finding the singular values of M is equivalent to finding the eigenvalues of the matrix $M^\top M$.

However, we will often assume certain properties about M, as is the case in the context of SVD for matrix completion. In these cases, it is assumed that M is low-rank or sparse, in which randomized SVD algorithms such as those in Halko et al. [2] take advantage of these properties, which were further improved [8]. These algorithms use techniques such as random projections in order to approximate the SVD, which is especially useful if the matrix M is sparse.

More recent algorithms have shifted to focus on empirical computation time and other constraints, such as computation power and space. Such algorithms made with the intent of CPU usage, such as ARPACK [9] and LAPACK [10], are packages developed to be used as part of libraries to calculate the SVD of any arbitrary matrix. In particular, libraries such as NumPy and SciPy use variations of the algorithms provided by LAPACK. These packages provide such optimized routines for the computation of SVD in a more distributed

setting, making it now feasible to compute the SVD empirically on a larger scale. This has also allowed SVD to be applied in real-time data processing, where parallel algorithms allow SVD to be computed much more efficiently.

There have also recently been updated SVD algorithms specifically for GPUs, as the GPU structures allow for some operations to be completed more efficiently. Algorithms such as the one found in Struski et al. [11] take advantage of this architecture in order to reduce the amount of computation time required for SVD.

Chapter 3

Algorithm

To formalize our intuition about the problem of distributed singular value decomposition, we first introduce a simple serialized algorithm that solves for the largest singular value of a matrix M using alternating least squares. We will then give a distributed version of the algorithm through parallelization that allows computation of the largest singular value of M. Finally, we give a distributed algorithm that calculates the largest r singular values for M for some constant r. Assuming that M is low-rank, this will be sufficient for finding the singular values we desire. We will prove the correctness of each of these algorithms in Chapter 4.

3.1 Centralized GD Algorithm for Largest SV

We first present a simple, centralized gradient descent algorithm that implements alternating least squares for the simple case where we only solve for the largest singular vector of a low-rank matrix M with true singular value decomposition $M = U\Sigma V^\top$. In this case, we can denote matrices of singular vectors U and $V^\top$ as unit vectors u and $v^\top$, while the matrix of singular values Σ will be denoted as the scalar σ. Letting $\tilde{u} = \sigma u$, we can rewrite the

objective as

$$f(\tilde{u}, v) = \sum_{i,j} (M_{ij} - \tilde{u}_i v_j^\top)^2.$$

We first present the outline of the simple algorithm using gradient descent via alternating least squares.

Algorithm 1 ALS-GD-Centralized

$t \leftarrow 0$

randomly initialize $\tilde{u}^{(0)}, v^{(0)}$

while convergence criteria not met **do**

$\tilde{u}^{(t+1)} \leftarrow \tilde{u}^{(t)} - \alpha_{\tilde{u}}^{(t)} \nabla_{\tilde{u}} f\left(\tilde{u}^{(t)}, v^{(t)}\right)$

$v^{(t+1)} \leftarrow v^{(t)} - \alpha_{v}^{(t)} \nabla_{v} f\left(\tilde{u}^{(t)}, v^{(t)}\right)$

$\tilde{u}^{(t+1)} \leftarrow \tilde{u}^{(t+1)} \cdot ||v^{(t+1)}||$

$v^{(t+1)} \leftarrow v^{(t+1)} \cdot ||v^{(t+1)}||^{-1}$

$t \leftarrow t + 1$

end while

$u^{(t)} \leftarrow \frac{\tilde{u}^{(t)}}{||\tilde{u}^{(t)}||}$

$\sigma \leftarrow ||\tilde{u}^{(t)}||$

return $u^{(t)}, \sigma, v^{(t)\top}$

In Section A of the appendix, we first discuss the motivation of the algorithm, the normalization requirement, and the choice of hyperparameters. Then, in Section A.1 of the appendix, we show that the gradient and Hessian matrices can be easily calculated and the matrix multiplications are also feasible to compute, making the algorithm viable in practice. Finally, we show the theoretical correctness of this simple algorithm in Chapter 4, and after discussing the convergence criteria and random initialization, give empirical results for this algorithm in Chapter 5.

3.2 Distributed SGD Algorithm for Largest SV

Next, we give a distributed gradient descent algorithm that implements alternating least squares to find the largest singular value of a large matrix M that no longer fits in main memory. Due to parallelization and minimal enforcement of the order threads can execute, this results in the gradient descent being stochastic in nature.

Since the matrix M is larger than main memory, calculating the gradient of M with respect to the entire vector $\tilde{u}$ or v, as seen in Section A.1 of the appendix, is no longer feasible by reading the entire matrix. Instead, we require M to be read in chunks of rows or columns, so that the gradients are also computed in chunks. Although we require at least one row of M to fit in main memory, this algorithm can easily be adapted to situations in which this is not the case through further parallelization.

In the distributed version of the algorithm, there will be two types of threads: *gradient threads* and the *aggregation thread*. Each gradient thread is given some rows r_{start} and r_{end} to compute $\nabla_{\tilde{u}} f\left(\tilde{u}^{(t)}, v^{(t)}\right)[r_{\text{start}} : r_{\text{end}}]$ or columns c_{start} and c_{end} to compute $\nabla_v f\left(\tilde{u}^{(t)}, v^{(t)}\right)[c_{\text{start}} : c_{\text{end}}]$. When the rows and columns of M are fully partitioned, the aggregation thread will take the partially computed gradients and combine the results into the fully computed gradients $\nabla_{\tilde{u}} f\left(\tilde{u}^{(t)}, v^{(t)}\right)$ and $\nabla_v f\left(\tilde{u}^{(t)}, v^{(t)}\right)$, using these to update $\tilde{u}$ and v.

In the following subroutines, each thread will need shared access to M, the current vectors $\tilde{u}^{(t)}$ and $v^{(t)}$, the vectors $\tilde{u}'$ and v' to store the partially computed results of $\nabla_{\tilde{u}} f\left(\tilde{u}^{(t)}, v^{(t)}\right)$ and $\nabla_v f\left(\tilde{u}^{(t)}, v^{(t)}\right)$, as well as certain hyperparameters including step size.

We present the distributed algorithm for each thread below.

Algorithm 2a ALS-GD-Distributed, gradient thread for $\tilde{u}$

Require: start row r_{start}, end row r_{end} to compute partial gradient

while convergence criteria not met **do**

$\tilde{u}'[r_{\text{start}} : r_{\text{end}}] \leftarrow \nabla_{\tilde{u}} f\left(\tilde{u}^{(t)}, v^{(t)}\right)[r_{\text{start}} : r_{\text{end}}]$

end while

Algorithm 2b ALS-GD-Distributed, gradient thread for v

Require: start column c_{start}, end column c_{end} to compute partial gradient

while convergence criteria not met **do**

$v'[c_{\text{start}} : c_{\text{end}}] \leftarrow \nabla_{v} f\left(\tilde{u}^{(t)}, v^{(t)}\right)[c_{\text{start}} : c_{\text{end}}]$

end while

Algorithm 2c ALS-GD-Distributed, aggregation thread

$t \leftarrow 0$

randomly initialize $\tilde{u}^{(0)}, v^{(0)}$

while convergence criteria not met **do**

$\tilde{u}^{(t+1)} \leftarrow \tilde{u}^{(t)} - \alpha_{\tilde{u}}^{(t)} \tilde{u}'$

$v^{(t+1)} \leftarrow v^{(t)} - \alpha_{v}^{(t)} v'$

$\tilde{u}^{(t+1)} \leftarrow \tilde{u}^{(t+1)} \cdot ||v^{(t+1)}||$

$v^{(t+1)} \leftarrow v^{(t+1)} \cdot ||v^{(t+1)}||^{-1}$

$t \leftarrow t + 1$

end while

$u^{(t)} \leftarrow \frac{\tilde{u}^{(t)}}{||\tilde{u}^{(t)}||}$

$\sigma \leftarrow ||\tilde{u}^{(t)}||$

return $u^{(t)}, \sigma, v^{(t)\top}$

In Section A.2 of the appendix, we show that the partial gradients of $\tilde{u}$ and v and Hessian matrices can be calculated and stored in main memory, making the algorithm viable when

computation power greatly exceeds memory. We will also discuss the read and write access requirements for each thread in more detail in the appendix.

As for the theoretical correctness of this algorithm, it is easily observed that when we force each of the gradient threads to execute before the first iteration of and in between every two consecutive iterations of the aggregation thread, the behavior of the distributed algorithm will be precisely the same as that of the centralized algorithm. To formalize this, we give a more rigorous version of the correctness argument of the distributed algorithm in Chapter 4. However, the enforcement of serialization is often infeasible in practice, and it is often more efficient for the aggregation thread to run before all of the gradient threads have finished updating. In Chapter 5, we will see that, empirically, using the distributed algorithm without forcing thread execution order allows for a faster run time without compromising much accuracy in computing the largest singular value of M.

3.3 Distributed SGD Algorithm for r Largest SVs

We are now finally ready to give a fully distributed gradient descent algorithm implementing alternating least squares to find the largest r singular values of a large matrix M that does not fit in main memory. The fully distributed algorithm is quite similar to the algorithm in Section 3.2 and can be done by performing it serially r times, replacing M with $M - u_{(i)}\sigma_{(i)}v_{(i)}^\top$, where $u_{(i)}, \sigma_{(i)}, v_{(i)}^\top$ are the values returned by Algorithm 2c after the i^{th} iteration. However, as we are not able to store the entire matrix $M - u_{(1)}\sigma_{(1)}v_{(1)}^\top$ after the first output of Algorithm 2c without rewriting M entirely, we will need a slightly different approach.

Instead, we notice that the above procedure can be parallelized. In order to do so, we will need to keep track of the current vectors $\tilde{u}_{(i)}^{(t)}$ and $v_{(i)}^{(t)}$ for $i \in [r]$, to keep track of the singular vectors corresponding to the i^{th} largest singular value, as well as $\tilde{u}'_{(i)}$ and $v'_{(i)}$ to store the partially computed results of $\nabla_{\tilde{u}_{(i)}} f\left(\tilde{u}_{(i)}^{(t)}, v_{(i)}^{(t)}\right)$ and $\nabla_{v_{(i)}} f\left(\tilde{u}_{(i)}^{(t)}, v_{(i)}^{(t)}\right)$, the partial gradients corresponding to the i^{th} largest singular value. The computation of the gradient

corresponding to the i^{th} largest singular value will no longer use only M but will need to use

$$M_{(i)} := M - \sum_{j=0}^{i-1} \tilde{u}_{(j)}^{(t)} v_{(j)}^{(t)\top},$$

where $\tilde{u}_{(j)}^{(t)}$ and $v_{(j)}^{(t)\top}$ are the most updated versions of the calculated vectors corresponding to the j^{th} largest singular value.

The rest of the algorithm is analogous to that of Section 3.2, as we read M in chunks of rows or columns to compute the required gradients in chunks. We will again have gradient threads and an aggregation thread appropriated to the computation of the i^{th} largest singular value for each $i \in [r]$, so there will be r times as many threads in the fully distributed algorithm.

We present the fully distributed algorithm to compute the r largest singular values below.

Algorithm 3a ALS-GD-Fully-Distributed, gradient thread for $\tilde{u}_{(i)}$

Require: start row r_{start}, end row r_{end} to compute partial gradient

Require: index $i \in [r]$ for which gradient is being calculated

while convergence criteria not met **do**

$\tilde{u}'_{(i)}[r_{\text{start}} : r_{\text{end}}] \leftarrow \nabla_{\tilde{u}_{(i)}} f\left(\tilde{u}_{(i)}^{(t)}, v_{(i)}^{(t)}\right)[r_{\text{start}} : r_{\text{end}}]$

end while

Algorithm 3b ALS-GD-Fully-Distributed, gradient thread for $v_{(i)}$

Require: start column c_{start}, end column c_{end} to compute partial gradient

Require: index $i \in [r]$ for which gradient is being calculated

while convergence criteria not met **do**

$v'_{(i)}[c_{\text{start}} : c_{\text{end}}] \leftarrow \nabla_{v_{(i)}} f\left(\tilde{u}_{(i)}^{(t)}, v_{(i)}^{(t)}\right)[c_{\text{start}} : c_{\text{end}}]$

end while

Algorithm 3c ALS-GD-Fully-Distributed, aggregation thread for index i

Require: index $i \in [r]$ for which gradient is being calculated

$t \leftarrow 0$

randomly initialize $\tilde{u}_{(i)}^{(0)}, v_{(i)}^{(0)}$

while convergence criteria not met **do**

$\tilde{u}_{(i)}^{(t+1)} \leftarrow \tilde{u}_{(i)}^{(t)} - \alpha_{\tilde{u}_{(i)}}^{(t)} \tilde{u}'_{(i)}$

$v_{(i)}^{(t+1)} \leftarrow v_{(i)}^{(t)} - \alpha_{v_{(i)}}^{(t)} v'_{(i)}$

$\tilde{u}_{(i)}^{(t+1)} \leftarrow \tilde{u}_{(i)}^{(t+1)} \cdot ||v_{(i)}^{(t+1)}||$

$v_{(i)}^{(t+1)} \leftarrow v_{(i)}^{(t+1)} \cdot ||v_{(i)}^{(t+1)}||^{-1}$

$t \leftarrow t + 1$

end while

$u_{(i)}^{(t)} \leftarrow \frac{\tilde{u}_{(i)}^{(t)}}{||\tilde{u}_{(i)}^{(t)}||}$

$\sigma_{(i)} \leftarrow ||\tilde{u}_{(i)}^{(t)}||$

return $u_{(i)}^{(t)}, \sigma_{(i)}, v_{(i)}^{(t)\top}$

In Section A.3 of the appendix, we show how to calculate and store partial gradients of $\tilde{u}_{(i)}$ and $v_{(i)}$ for each index i as well as the corresponding Hessian matrices in order for the algorithm to be viable. We also discuss the read and write requirements for each thread in greater detail, showing that the algorithm can be implemented practically. In practice, this fully distributed algorithm is quite computationally expensive, so we typically use the algorithm in Section 3.2 when $r > 2$, updating M as necessary. In addition, we show the theoretical correctness of this algorithm in Chapter 4 and further theoretical guarantees, including error bounds where we are only given a noisy observation of M, in Appendix B.4, and observe the empirical performance of the fully parallelized algorithm in Chapter 5.

Chapter 4

Results

Now that we have introduced the algorithms implementing alternating least squares in Chapter 3, we show the correctness of these algorithms via the following theorems. We start by proving the following theorem regarding the fully serialized algorithm.

Theorem 4.0.1. *With probability* 1 *with respect to the random initialization, Algorithm* 1*, the centralized gradient descent algorithm for finding the largest singular value of a matrix* M*, correctly finds the largest singular value* σ *and singular vectors* $u, v^\top$ *corresponding to the largest singular value.*

The proof of Theorem 4.0.1 will allow us to prove results on the distributed algorithms, which we establish below.

Theorem 4.0.2. *Suppose that the threads of Algorithm* 2*, the distributed gradient descent algorithm for finding the largest singular value of a matrix* M*, are executed in an order such that each of the gradient threads executes before the first iteration of and in between every two consecutive iterations of the aggregation thread. Then, with probability* 1 *with respect to the random initialization, the algorithm correctly finds the largest singular value* σ *and singular vectors* $u, v^\top$ *corresponding to the largest singular value.*

Theorem 4.0.3. *Suppose that the threads of Algorithm 3, the distributed gradient descent algorithm for finding the r largest singular values of a matrix M, are executed in an order such that each of the gradient threads corresponding to the computation of the i^{th} largest singular value executes before the first iteration of and in between every two consecutive iterations of the aggregation thread corresponding to the computation of the i^{th} largest singular value for each $i \in [r]$. Then, with probability 1 with respect to the random initialization, the algorithm correctly finds the r largest singular values Σ_r and matrices of singular vectors $U_r, V_r^\top$ corresponding to the largest r singular values.*

Finally, we will show results when we are only given a noisy observation M_{obs} of the true matrix M_{true}, where $M_{\text{obs}} = M_{\text{true}} + M_{\text{err}}$, and M_{err} is a matrix of noise satisfying some properties. In particular, we will prove the following theorem.

Theorem 4.0.4. *Suppose that we are only given a noisy observation M_{obs} of the true matrix M_{true}, where $M_{obs} = M_{true} + M_{err}$. Also, suppose that Σ_r is a diagonal matrix of the top r singular values $\sigma_{(1)}, \sigma_{(2)}, \ldots, \sigma_{(r)}$ of M_{true} and U_r, $V_r^\top$ are the corresponding matrices of singular vectors. Then, given bounds on the error matrix $||M_{err}|| < \left(1 - \frac{1}{\sqrt{2}}\right)\left(\sigma_{(r)} - \sigma_{(r+1)}\right)$, with probability 1 with respect to the random initialization, Algorithm 3 finds the matrix $\hat{\Sigma}_r$ of singular values $\hat{\sigma}_{(1)}, \hat{\sigma}_{(2)}, \ldots, \hat{\sigma}_{(r)}$ and corresponding matrices of singular vectors $\hat{U}_r, \hat{V}_r^\top$ such that*

$$|\sigma_{(i)} - \hat{\sigma}_{(i)}| \leq ||M_{err}||$$

for all $i \in [r]$ and

$$\max\{\mathsf{dist}(U_r, \hat{U}_r), \mathsf{dist}(V_r, \hat{V}_r)\} \leq \frac{2||M_{err}||}{\sigma_{(r)} - \sigma_{(r+1)}},$$

where $\mathsf{dist}(A, B)$ measures the sine of the angle between vector subspaces A and B.

We defer the proofs of these theorems to Appendix B.

We also note that the bounds in Theorem 4.0.4 are not sharp and can be improved via methods in Stewart et al. [12], amongst other bounds. However, we leave the improvements

up to the reader, since depending on the desired conclusions, such improvements on the bounds may have different requirements on the properties of M_{err} and other assumptions that must be made.

Having shown the mathematical robustness of the distributed gradient descent algorithms through least squares, we are now ready to see the performance of the distributed algorithms on real datasets. In Chapter 5, we run the algorithms given in Chapter 3 on benchmark datasets in order to show their empirical correctness and scalability.

Chapter 5

Experiments

We now conduct a numerical test by running the distributed algorithms implementing gradient descent via alternating least squares given in Sections 3.2 and 3.3, ALS-GD-Distributed and ALS-GD-Fully-Distributed, on various matrices to compute their largest singular values. In doing so, we aim to show that the algorithms not only correctly find the required singular values and vectors empirically but also are more scalable, as they are able to run on larger matrices than standard algorithms that require the entire matrix to fit in main memory.

We organize the rest of this chapter as follows. In Section 5.1, we describe the datasets we use in our experiment and explain why we choose this data. We also introduce the benchmark algorithms we are comparing our distributed algorithms against in Section 5.2. In Section 5.3, we give the initial setup for the experiment and list metrics to compare the correctness and other properties of the algorithm in Section 5.4. Finally, in Section 5.5, we give a summary of the conclusions we draw based on the results of the experiment.

5.1 Data

We use two types of data in our experiment: synthetic data and real-world data. Synthetic data is useful since we can create data where we know the ground truth of singular values and vectors to be solved for, and we can randomly generate these singular values and vectors

according to known distributions such as the Tracy-Widom distribution and according to other standards. Having established the correctness of the algorithm on synthetic data, we will also experiment on larger real-world datasets where we may desire the largest singular values, such as matrices that describe user ratings or recommendations. Running the experiment on these larger datasets will allow us to establish the scalability of the distributed algorithms, especially in conditions where the benchmark algorithms we are comparing fail to run.

We now describe precisely what data was used in our experiment. For synthetic data, we have created an array of matrices starting from size 5×5 and ending at size $10\,000 \times 10\,000$, incrementing the size by powers of 10, increasing the size of the increment only when the next power of 10 is reached. For the singular values Σ, we use the Tracy-Widom distribution to generate the largest singular value with all other smaller singular values generated randomly, and we generate random matrices of orthogonal vectors U and $V^\top$ using the SciPy package. This gives us a setting of ground truth for our synthetic datasets.

For the real-world datasets, we primarily use matrices of user ratings from MovieLens [13], standardized so that all entries are in the range $[-5, 5]$. We choose MovieLens as a benchmark of datasets since it is a compilation of ratings, so knowing the top singular values may be useful for applications such as matrix completion. Additionally, there are many sizes of matrices we can experiment with, which allows us to show the scalability of the algorithm when run on larger datasets.

5.2 Algorithms Compared

We compare ALS-GD-Distributed and ALS-GD-Fully-Distributed against the standard SVD algorithms NumPy and SciPy. We choose to compare against NumPy and SciPy since these are existing, well-established Python packages with well-documented algorithms that are often considered the standard for SVD computation.

5.3 Experiment Setup and Hyperparameters

We run ALS-GD-Distributed and ALS-GD-Fully-Distributed on the datasets described in Section 5.1, solving for the top 2 singular values. This will be enough to show the correctness of the fully parallelized version of the algorithm.

We run the experiment first on a small machine with 8 cores and 8 GB of RAM while later moving to a larger machine with 128 cores and 64 GB RAM. The optimal number of chunks we run the algorithm with differs on each machine, but we will typically choose the number of processes to be comparable to the number of cores of the machine being used to run the experiment. After choosing this number, we divide the rows and columns of the matrix M as evenly as possible to ensure the chunks of gradients are calculated in roughly the same amount of time.

For the random initialization of $\tilde{u}_{(i)}^{(0)}, v_{(i)}^{(0)}$ for $i = 1, 2$ in the algorithms, we pick 2 random orthonormal vectors in $\mathbb{R}^d$, and randomly choose the magnitudes of the $\tilde{u}_{(i)}$ via sampling from the Tracy-Widom singular value distribution. This will give us a random initialization that is representative of the true rank 2 approximation of the SVD. We will also run the algorithm many times from multiple randomly chosen initializations in order to prove the empirical correctness of the algorithms.

For the convergence criteria to determine when the algorithm terminates, we terminate when the gradients computed by the algorithm, $\nabla_{\tilde{u}} f\left(\tilde{u}^{(t)}, v^{(t)}\right)$ and $\nabla_v f\left(\tilde{u}^{(t)}, v^{(t)}\right)$ both have magnitudes smaller than 10^{-6} or at 10^6 iterations of the aggregation thread, whichever happens first. Empirically, this allows us to terminate the algorithm while giving correct results up to a threshold while not expending too much time near the minimum with oscillations or minimal improvement of the objective.

Finally, we choose step sizes $\alpha_{\tilde{u}}^{(t)} = \beta_{\tilde{u}}^{(t)} \left(\nabla_{\tilde{u}}^2 f\left(\tilde{u}^{(t)}, v^{(t)}\right)\right)^{-1}$ and $\alpha_v^{(t)} = \beta_v^{(t)} \left(\nabla_v^2 f\left(\tilde{u}^{(t)}, v^{(t)}\right)\right)^{-1}$, where $\beta_{\tilde{u}}^{(t)}, \beta_v^{(t)} = \frac{1}{t^{0.5+O(1)}}$. We believe that choosing decreasing sequences $\left\{\beta_{\tilde{u}}^{(t)}\right\}, \left\{\beta_v^{(t)}\right\}$ such that for all t, $0 < \beta_{\tilde{u}}^{(t)}, \beta_v^{(t)} < 1$, and that $\sum_t \beta_{\tilde{u}}^{(t)}, \sum_t \beta_v^{(t)} = \infty$ while $\sum_t \beta_{\tilde{u}}^{(t)2}, \sum_t \beta_v^{(t)2} <$

∞ will give similar results. However, we have found that our choice of step sizes has reliably given the convergence of the algorithms in the least number of iterations without compromising their correctness, so we thus give the results corresponding to our choice of hyperparameters below.

We remark that in our analysis of the algorithms, we require certain restrictions on the thread execution order. However, this requirement is often infeasible to enforce in practice, as the differing speeds of calculations done by the threads may cause inefficient use of resources via waiting. While it may be possible to distribute the gradient calculation further, we find that the algorithms perform with similar accuracy in Section 5.5 and thus see no reason to enforce the requirements on thread execution order.

5.4 Metrics

We are mainly concerned about the correctness and scalability of the algorithms ALS-GD-Distributed and ALS-GD-Fully-Distributed and give metrics on how they are measured.

We first introduce the correctness measures for the distributed algorithms as *singular value distance* and *reconstruction error*. When running an algorithm, we define the singular value distance as $\max_{i\in[r]} |\sigma_{(i)} - \hat{\sigma}_{(i)}|$ and reconstruction error as $||U_r\Sigma_r V_r^\top - \hat{U}_r\hat{\Sigma}_r\hat{V}_r^\top||_F^2$. These definitions will allow us to quantify the correctness of the singular values by showing their proximity to the true singular values, and the singular vectors by showing that the rank r approximation of M does not differ by a large amount when compared to the true rank r approximation.

To measure scalability, we will see that using the sizes of the matrices M as our input suffices. As we will see in Section 5.5, the inability of the NumPy/SciPy benchmark algorithms to run on larger matrices on a machine with insufficient RAM to store M in main memory indicates that our distributed algorithms can be used to scale.

5.5 Results

We now give the results of the experiments run by ALS-GD-Distributed when compared to NumPy and SciPy. We note that the results from the fully distributed algorithm, ALS-GD-Fully-Distributed, are nearly identical and require a longer computation time for convergence, so we will not give separate results. A graph of results is displayed below.

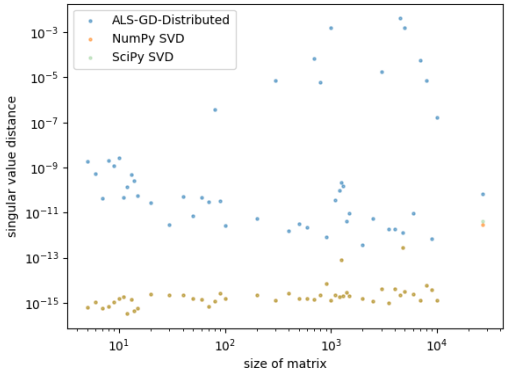

Figure 5.1: Singular value distance for various algorithms on experimental data

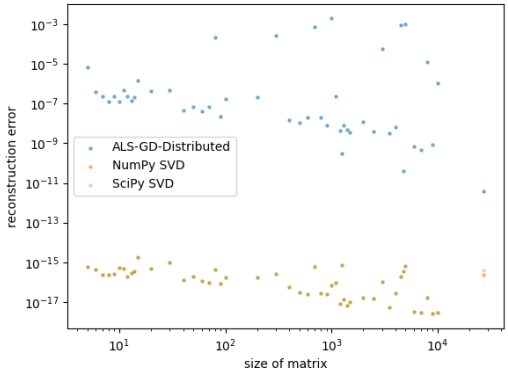

Figure 5.2: Reconstruction error for various algorithms on experimental data

Note that in Figures 5.1 and 5.2, both the x and y-axes are on a log scale for the sake of the interpretability of the graphs. We also summarize the results in the table below.

Algorithm	ALS-GD-Distributed	NumPy	SciPy
max d, 8 GB RAM	$> 30\,000$	$< 10\,000$	$< 10\,000$
max d, 128 GB RAM	$> 30\,000$	$> 30\,000$	$> 30\,000$
max singular value distance	1.97×10^{-3}	1.73×10^{-15}	1.73×10^{-15}
max reconstruction error	4.13×10^{-3}	2.96×10^{-12}	4.07×10^{-12}

Table 5.1: Summary results for singular value distance and reconstruction error for various algorithms on experimental data

As we can see in Figures 5.1 and 5.2, even though the errors from our algorithm are not as small as those obtained from NumPy or SciPy, given the number of iterations of the

algorithm run, the errors are acceptable. As can be seen in Table 5.1, the largest singular value distances and reconstruction errors we have achieved over tests on all of the matrices are on the order of 10^{-3}, a tolerable threshold.

Most of the larger singular value distances, on the order of 10^{-4}, are due to stopping after running all of the allowed iterations, which we see in Figure 5.3 below. Even though we have relatively smooth convergence to the true singular value, due to only running the algorithm for 10^6 iterations, this will cause the algorithm to terminate prematurely in some cases such as the one when $d = 7\,000$. Also, most of the smaller errors, on the order of 10^{-11}, are due to our early stopping condition, where we stop the algorithm as both of the partial gradients have magnitude $< 10^{-6}$. By adjusting this threshold, we are able to reach a lower singular value distance if we desire.

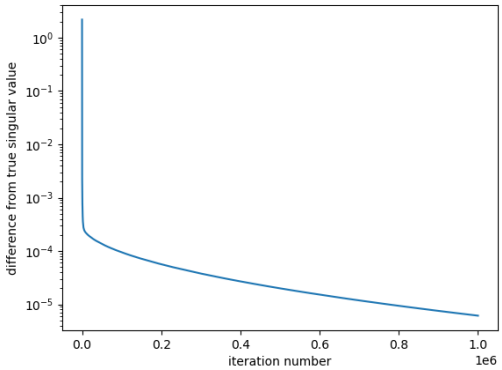

Figure 5.3: Singular value distance for ALS-GD-Distributed for intermediate iterations when $d = 7\,000$

However, we note that our algorithm, ALS-GD-Distributed, scales very well as it is theoretically able to run on matrices of any size. For $d \times d$ matrices M with $d \geq 10\,000$, NumPy and SciPy no longer run the algorithm due to a lack of RAM on the small machine. However, since our algorithm needs only the amount of RAM to store a few rows and columns of the matrix M, it runs without difficulty.

We also see that the distributed algorithms run reasonably well when the thread execution order is not enforced. As an example, in Figure 5.4 below, when thread execution order

is unrestricted, though the convergence from the top singular value as computed by the algorithm to the true top singular value is not smooth, it eventually occurs nonetheless. In fact, we remark that the unrestricted thread execution order or similar SGD algorithms may provide additional preventative measures against the GD-based procedure from converging to a saddle point.

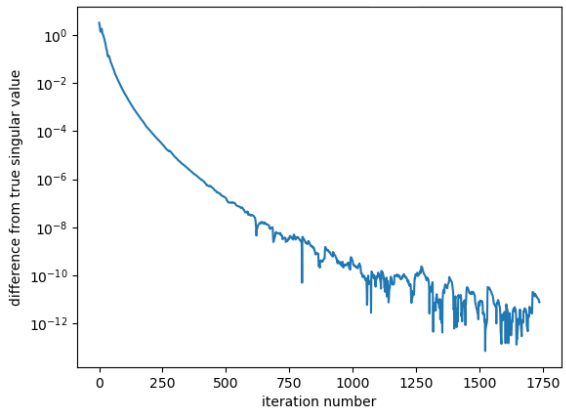

Figure 5.4: Singular value distance for ALS-GD-Distributed for intermediate iterations when $d = 8\,000$ with unrestricted thread execution order

Thus, we have shown that our algorithm ALS-GD-Distributed is correct up to thresholding and scalable, capable of running on larger matrices even with limited memory resources. This concludes our argument that our distributed algorithm finds the required singular values and vectors empirically and can run on larger matrices that do not fit entirely in main memory.

Chapter 6

Conclusion and Future Work

We have introduced distributed algorithms ALS-GD-Distributed and ALS-GD-Fully-Distributed, Algorithms 2 and 3 in Sections 3.2 and 3.3, respectively, in an attempt to find a method for parallelized computation of the top r singular values of a matrix M, for constant r. We have proved that these algorithms are mathematically robust even when the matrix M is noisy, and we have also shown the correctness of the algorithms and their scalability in experiments on synthetic and real-world data. Thus, we believe that ALS-GD-Distributed and ALS-GD-Fully-Distributed are viable gradient descent-based algorithms to run when attempting to compute the largest singular values of a matrix, especially on machines with high computation power and limited memory.

Since these algorithms are still in the early stages of development, various optimizations can be made to decrease their runtime. For example, though much of the code for this algorithm uses standard libraries such as NumPy, much of it is written entirely in Python. We believe that writing this code in languages such as C will speed up many aspects of this algorithm, including memory accesses to M, which we measure to be the main reason for the algorithm's comparatively large runtime. In addition, further optimizations include possible dynamic hyperparameter tuning to decrease the number of iterations it takes for the algorithm to converge, databases or threadsafe shared memory that allows for faster

access to the values $\tilde{u}_{(i)}^{(t)}, v_{(i)}^{(t)}$ and the partial gradients $\nabla_{\tilde{u}_{(i)}} f\left(\tilde{u}_{(i)}^{(t)}, v_{(i)}^{(t)}\right)$, $\nabla_{v_{(i)}} f\left(\tilde{u}_{(i)}^{(t)}, v_{(i)}^{(t)}\right)$, and possible caching to decrease the number of reads required from the disk where M is stored. Sufficient optimization of the algorithm will allow us to create a functional library with the desired algorithms and features as a new method for running distributed gradient descent via least squares to compute the top singular values and singular vectors of any matrix.

Appendix A

Gradient Calculation and Further Discussion of Algorithms

In this section of the appendix, we first discuss the motivation behind the gradient descent algorithms. We will also first discuss the normalization step common to all of the algorithms in Chapter 3 and discuss the hyperparameter choices. Then, we will calculate the partial gradients for each algorithm in the following sections, as these will differ slightly between the algorithms.

The motivation behind the simple GD algorithm is to notice that if we are to hold v as fixed, we can use gradient descent to find the optimal $\tilde{u}$. The procedure of GD from time step $t \to t+1$ is as follows:

$$\tilde{u}^{(t+1)} \leftarrow \tilde{u}^{(t)} - \alpha_{\tilde{u}}^{(t)} \nabla_{\tilde{u}} f\left(\tilde{u}^{(t)}, v^{(t)}\right),$$

for an appropriately chosen step size $\alpha_{\tilde{u}}^{(t)}$. Similarly, by fixing $\tilde{u}^{(t)}$, we can compute the analogous update for v as

$$v^{(t+1)} \leftarrow v^{(t)} - \alpha_{v}^{(t)} \nabla_{v} f\left(\tilde{u}^{(t)}, v^{(t)}\right).$$

However, we note that after updating v, it may not necessarily be true that $||v^{(t+1)}|| = 1$. Thus, we require a normalization step, in which the magnitude from v is multiplied into the corresponding iteration of $\tilde{u}$ before v is normalized so that the evaluation of the objective remains unchanged.

We typically choose step sizes $\alpha_{\tilde{u}}^{(t)}$ and $\alpha_v^{(t)}$ as multiples of the inverse Hessian matrices with respect to $\tilde{u}$ and v so that $\alpha_{\tilde{u}}^{(t)} = \beta_{\tilde{u}}^{(t)} \left(\nabla_{\tilde{u}}^2 f\left(\tilde{u}^{(t)}, v^{(t)}\right)\right)^{-1}$ and $\alpha_v^{(t)} = \beta_v^{(t)} \left(\nabla_v^2 f\left(\tilde{u}^{(t)}, v^{(t)}\right)\right)^{-1}$, where $\beta_{\tilde{u}}^{(t)}$ and $\beta_v^{(t)}$ are functions of t that take on values between 0 and 1. In practice, we also choose to stop when the magnitude of the gradient is smaller than a certain threshold or when a set number of iterations has been reached.

In the remainder of Section A of the appendix, we calculate all of the relevant partial gradients and Hessian matrices for each of the algorithms presented in Chapter 3. We demonstrate that each of these computations can be done efficiently, making the algorithms viable in practice. For the distributed algorithms, we will also state the shared memory requirements for read/write access to show that the overall memory requirement is not excessive, leading to feasible implementations.

A.1 Gradients for Centralized GD Algorithm

Recall that in Chapter 3, we rewrite the objective as

$$f(\tilde{u}, v) = \sum_{i,j} (M_{ij} - \tilde{u}_i v_j^\top)^2 = ||M - \tilde{u}v^\top||_F^2,$$

where $\tilde{u} = \sigma u$. From here, it is easy to calculate that the partial gradients can be written as

$$\nabla_{\tilde{u}} f\left(\tilde{u}, v\right) = -2(M - \tilde{u}v^\top)v,$$

$$\nabla_v f\left(\tilde{u}, v\right) = -2(M - \tilde{u}v^\top)^\top \tilde{u}.$$

We also compute the Hessian matrices as

$$\nabla^2_{\tilde{u}} f\left(\tilde{u}, v\right) = 2||v||_2^2 I = 2I,$$

$$\nabla^2_{v} f\left(\tilde{u}, v\right) = 2||\tilde{u}||_2^2 I,$$

where I is the $d \times d$ identity matrix.

When M is small enough to fit in main memory, each computation of the gradient takes only $O(d^2)$ time as the only expensive multiplication operations include at most one matrix with at least one multiplicand being a vector. The computation of the Hessian matrices is even simpler and takes $O(d)$ time.

Thus, each iteration of the while loop in Algorithm 1 takes $O(d^2)$ time, making the algorithm quite practical to use. For the sake of comparison, this is also the time required to read M once, which is the minimum amount of time we expect to take in the distributed versions of this algorithm where we are no longer able to store M in main memory.

A.2 Gradients and Read/Write Requirements for Distributed SGD Algorithm

Again, we rewrite the objective as can be seen in Section A.1, where it is then straightforward to compute the following chunks of partial gradients. We find that

$$\begin{aligned}\nabla_{\tilde{u}} f\left(\tilde{u}, v\right)\left[r_{\text{start}} : r_{\text{end}}\right] &= \left(-2(M - \tilde{u}v^\top)v\right)\left[r_{\text{start}} : r_{\text{end}}\right] \\ &= -2(M[r_{\text{start}} : r_{\text{end}}, :] - (\tilde{u}[r_{\text{start}} : r_{\text{end}}])v^\top)v.\end{aligned}$$

Similarly, we compute the corresponding chunk for columns as

$$\nabla_{v} f\left(\tilde{u}, v\right)\left[c_{\text{start}} : c_{\text{end}}\right] = \left(-2(M - \tilde{u}v^\top)^\top\tilde{u}\right)\left[c_{\text{start}} : c_{\text{end}}\right]$$

$$= (-2(M[:, c_{\text{start}} : c_{\text{end}}] - \tilde{u}(v[c_{\text{start}} : c_{\text{end}}])^\top)^\top \tilde{u}.$$

The computation of the Hessian matrices remains unchanged from that of Section A.1.

From these calculations, it is easy to see that the total amount of time required for one entire iteration of the computation of the gradient is $O(d^2)$ since the amount of work being done by calculating the gradients in chunks is the exact same as that of calculating the gradients all at once in Section 3.1. However, we note that the distributed algorithm only requires $O(d)$ memory in order to calculate the entire partial gradient piece by piece. Thus, the distributed algorithm is a good alternative where M does not fit into main memory or computation power greatly exceeds usable memory.

We now analyze the shared read and write requirements of the individual threads in the algorithm. Though the threads will only need read-only access to M and the hyperparameters, the gradient threads will need write access to $\tilde{u}'$ and v' and read access to $\tilde{u}$ and v, while the aggregation thread will need read access to $\tilde{u}'$ and v' and write access to $\tilde{u}$ and v. Thus, the distributed algorithm must be run with a database or some other structure that can function correctly and safely with concurrency operations of this nature. Though the time needed to access the shared data may increase, we argue that this is offset by the vast decrease in memory required for the distributed algorithm to run.

A.3 Gradients and Read/Write Requirements for Fully Distributed SGD Algorithm

As stated in Section 3.3, finding the partial gradients required for the i^{th} largest singular value, namely $\nabla_{\tilde{u}_{(i)}} f\left(\tilde{u}_{(i)}, v_{(i)}\right)$ and $\nabla_{v_{(i)}} f\left(\tilde{u}_{(i)}, v_{(i)}\right)$, can be done by a process identical to that of Section A.2 after removing the components corresponding to the first $i-1$ largest singular values from M by subtracting $\sum_{j=0}^{i-1} \tilde{u}_{(j)} v_{(j)}^\top$, the appropriate rank $i-1$ approximation.

Thus, when fully parallelized, the chunks of partial gradients we desire are just

$$\begin{aligned}\nabla_{\tilde{u}_{(i)}} f\left(\tilde{u}_{(i)}, v_{(i)}\right)[r_{\text{start}} : r_{\text{end}}] &= \left(-2\left(M - \sum_{j=0}^{i-1} \tilde{u}_{(j)} v_{(j)}^\top - \tilde{u}_{(i)} v_{(i)}^\top\right) v_{(i)}\right)[r_{\text{start}} : r_{\text{end}}] \\ &= -2\left(M[r_{\text{start}} : r_{\text{end}}, :] - \sum_{j=0}^{i} (\tilde{u}_{(j)}[r_{\text{start}} : r_{\text{end}}]) v_{(j)}^\top\right) v_{(i)},\end{aligned}$$

and

$$\begin{aligned}\nabla_{v_{(i)}} f\left(\tilde{u}_{(i)}, v_{(i)}\right)[c_{\text{start}} : c_{\text{end}}] &= \left(-2\left(M - \sum_{j=0}^{i-1} \tilde{u}_{(j)} v_{(j)}^\top - \tilde{u}_{(i)} v_{(i)}^\top\right)^\top \tilde{u}_{(i)}\right)[c_{\text{start}} : c_{\text{end}}] \\ &= -2\left(M[:, c_{\text{start}} : c_{\text{end}}] - \sum_{j=0}^{i} \tilde{u}_{(j)} (v_{(j)}[c_{\text{start}} : c_{\text{end}}])^\top\right)^\top \tilde{u}_{(i)},\end{aligned}$$

with the computation of the Hessian matrices unchanged.

When solving for the top r singular values, treating r as a constant, we can again see that the total amount of time required for one full iteration of computation of the gradient is $O(d^2)$, and the memory needed for any specific thread to calculate the partial gradient is $O(d)$, which are exactly the same requirements as in the algorithm in Section 3.2. Thus, we again find the distributed algorithm as a practical alternative with large computation power and strict memory constraints.

We again analyze the shared read and write requirements of the threads in the algorithm. Each thread will need read access to M, the gradient threads calculating the partial gradients for the i^{th} largest singular value will need write access to $\tilde{u}'_{(i)}$ and $v'_{(i)}$ and read access to $\tilde{u}_{(j)}$ and $v_{(j)}$ for all $j \leq i$, while the aggregation thread will need read access to $\tilde{u}'_{(i)}$ and $v'_{(i)}$ and write access to $\tilde{u}_{(i)}$ and $v_{(i)}$. Thus, similar to the requirements of Section A.2, the distributed algorithm must be run with a database or some other structure that can function correctly and safely with concurrency operations of this nature. However, the time needed for access to the shared data may be even larger than that of Section A.2 due to the increased number of threads accessing the same data in memory, which may cause the algorithm to be less

efficient when r is a large constant due to the overhead costs of thread-safe data structures. In practice, running the algorithm in Section 3.2 serially may be more efficient than running the fully parallelized algorithm presented in Section 3.3.

Appendix B

Omitted Proofs in Chapter 4

In this section of the appendix, we prove the main theorems given in Chapter 4. In order to do so, we first provide the motivation and sketch of the required steps of the proof.

In Section B.1, we will first reduce the problem of finding singular values when M is a symmetric matrix, which will allow us to simplify many of the calculations and take advantage of the properties of symmetric matrices. In Section B.2, we show fundamental properties required in order for all of the gradient descent algorithms to converge. In Section B.3, we will prove Theorems 4.0.1, 4.0.2, and 4.0.3 by showing that the gradient descent algorithms given in Sections 3.1 and 3.2 will converge with certainty. In Section B.4, we will prove Theorem 4.0.4 by showing that our results hold even when given a noisy observation of M.

B.1 Reduction to when M is Symmetric

We first show that it suffices to consider the case where M is symmetric. More formally, we prove the following lemma.

Lemma B.1.1. *Suppose that we have an algorithm A that, given a symmetric matrix $M \in \mathbb{R}^{d\times d}$ and integer r as input, is able to find a matrix $X \in \mathbb{R}^{d\times r}$ such that $g(X) = ||M -$*

$XX^\top||_F^2$ is minimized. Then, given any $M' \in \mathbb{R}^{d\times d}$, we can find the largest singular value σ of M'. Moreover, if $u, v^\top$ are the left and right singular vectors corresponding to the top singular value, we can find the vector u up to sign such that $f(u, \sigma, v^\top) = ||M' - u\sigma v^\top||_F^2$ is minimized.

Proof. We claim the following algorithm B finds the top singular values σ and left singular vector u up to sign.

$B =$ On input M',

1. Compute $M = M'M'^\top$.
2. Run A on M with $r = 1$ and read its output as X.
3. Write $X = u_x\sigma_x,$ where $\sigma_x \in \mathbb{R}^{1\times 1}$ is a nonnegative scalar and $u_x \in \mathbb{R}^{d\times 1}$ is a vector with unit Euclidean norm.
4. Output u_x, σ_x.

We now prove that algorithm B correctly finds the desired top singular value and singular vector of M'. Suppose that $M' = U\Sigma V^\top$ is the true singular value decomposition of M'. Letting $M = M'M'^\top$, we find that

$$\begin{aligned} M &= M'M'^\top \\ &= (U\Sigma V^\top)(U\Sigma V^\top)^\top \\ &= U\Sigma V^\top V\Sigma U^\top \\ &= U\Sigma^2 U^\top \end{aligned}$$

by properties of U and V in the singular value decomposition.

Since M is symmetric, we can use the algorithm to find $X \in \mathbb{R}^{d\times 1}$ such that $||M - XX^\top||_F^2$

is minimized. Writing $X = u_x\sigma_x$ as given by the algorithm, we find that

$$\begin{aligned} g(X) &= ||M - XX^\top||_F^2 \\ &= ||U\Sigma^2U^\top - u_x\sigma_x^2u_x^\top||_F^2 \\ &= ||U^\top(U\Sigma^2U^\top - u_x\sigma_x^2u_x^\top)U||_F^2 \\ &= ||\Sigma^2 - (U^\top u_x)\sigma_x^2(U^\top u_x)^\top||_F^2, \end{aligned}$$

where the third equality is due to the invariance of the Frobenius norm under orthogonal transformations.

We now notice that $U^\top u_x$ is an orthogonal transformation, $U^\top$, applied to a unit norm vector, u_x, so $U^\top u_x$ itself is a unit norm vector. Thus, since the SVD of a diagonal matrix Σ^2 has the same left and right singular vectors, the value of $g(X)$ cannot be smaller than when $(U^\top u_x)\sigma_x^2(U^\top u_x)^\top$ is the rank 1 approximation for the singular value decomposition of Σ^2.

Now, let σ be the true largest singular value of M' and u be the associated left singular vector. Then we have that $(u_x\sigma_x)(u_x\sigma_x)^T = (u\sigma)(u\sigma)^T$, it follows that $u_x\sigma_x$ must be an orthogonal transformation of $u\sigma$. However, the only orthogonal transformation with dimension 1 is multiplication by ± 1, so the algorithm correctly finds the singular value $\sigma_x = \sigma$ and finds the corresponding left singular vector $u_x = \pm u$ correctly up to sign, which completes the proof. □

Note that after uniquely determining the top singular value and the corresponding left singular vector using the result from Lemma B.1.1, it is fairly straightforward to compute the right singular vector by setting the partial gradient equal to 0 or simple gradient descent. We also note that the proof above does not generalize well for finding the $r > 1$ largest singular values, since the algorithm only finds $u\sigma$ up to an orthogonal transformation, and such orthogonal transformations are no longer trivial for $r > 1$.

From now on, we will assume that M is symmetric and that we are trying to find a

matrix $X \in \mathbb{R}^{d \times r}$ such that the new objective function

$$g(X) = \sum_{i,j} (M_{ij} - X_i X_j^\top)^2 = ||M - XX^\top||_F^2$$

is minimized. This allows us to use previous results involving singular value decomposition for symmetric matrices as well as use nice properties of the matrix M due to its symmetry. We will see that the symmetry of M is crucial in our proof of the correct convergence of the gradient descent algorithms, which can be found in the following section.

B.2 Requirements for Optimality

Before proving the correctness of the algorithms in Section 3, we first prove some optimality conditions that are required for convergence. We first show that all local minima of the objective function $g(X) = ||M - XX^\top||_F^2$, introduced in Section B.1, are global minima, or that there is only one local minima up to sign. More formally, we will prove the following lemma, where we have replaced X with x since we only require the result to hold for matrices $X \in \mathbb{R}^{d \times r}$ with $r = 1$.

Lemma B.2.1. *There is at most one vector x up to sign such that $\nabla g(x) = 0$ and $\nabla^2 g(x) \geq 0$.*

Proof. The proof of this lemma is relatively straightforward; we will use a slight modification of the proof given for Lemma 3.2 in Ge et al. [14], restated as Lemma C.0.2 in Section C of the appendix for clarity, due to minor changes in the objective and typos in the original proof.

Recall that $g(x) = ||M - xx^\top||_F^2$, where M is symmetric with all of its entries observed. We compute $\nabla g(x)$ and $\nabla^2 g(x)$ to find that

$$\nabla g(x) = 4(||x||^2 x - Mx) \text{ and } \nabla^2 g(x) = 4(2xx^\top + ||x||^2 I - M),$$

where I is the identity matrix.

The rest of the proof proceeds exactly like that of Lemma 3.2 in Ge et al. [14] with slightly different assumptions, which we repeat here for clarity.

In order for $\nabla g(x) = 0$, we find that x must be an eigenvector of M with eigenvalue $||x||^2$.

We now show that the only x for which $\nabla^2 g(x) \geq 0$ is where x is the eigenvector with the largest magnitude. Suppose, for the sake of contradiction, that there exists z such that z is an eigenvector of M and $||z||^2 > ||x||^2$. Then we compute that

$$\begin{aligned} z^\top \nabla^2 g(x) z &= 4z^\top (2xx^\top + ||x||^2 I - M)z \\ &= 4(z^\top ||x||^2 I z - z^\top M z) \\ &= 4(||x||^2 ||z||^2 - ||z||^4) \\ &< 0, \end{aligned} \tag{B.2.1}$$

where the equality in line B.2.1 is due to the orthogonality of eigenvectors of a symmetric matrix.

This contradicts the positive semi-definite condition required for $\nabla^2 g(x)$. Thus, the only possible solutions are the eigenvectors with the largest magnitude.

We now verify that such eigenvectors x with the largest eigenvalue are indeed solutions. To show this, for any arbitrary vector v, note that we can write $v = cx + \sum_i c_i w_i$, where w_i are the orthonormal eigenvectors of M excluding x with eigenvalues λ_i. We compute that

$$\begin{aligned} v^\top \nabla^2 g(x) v &= 4v^\top (2xx^\top + ||x||^2 I - M)v \\ &= 4(2c^2 ||x||^4 + v^\top ||x||^2 I v - v^\top M v) \\ &\geq 4(v^\top ||x||^2 I v - v^\top M v) \\ &\geq 4\left(||x||^2 ||v||^2 - \left(cx + \sum_i c_i w_i \right)^\top M \left(cx + \sum_i c_i w_i \right) \right) \end{aligned} \tag{B.2.2}$$

$$
\begin{aligned}
&= 4\left(||x||^2||v||^2 - \left(cx + \sum_i c_i w_i\right)^\top \left(c||x||^2 x + \sum_i c_i \lambda_i w_i\right)\right) \\
&= 4\left(||x||^2||v||^2 - \left(c^2||x||^4 + \sum_i c_i^2 \lambda_i\right)\right) \\
&\geq 4\left(||x||^2||v||^2 - \left(c^2||x||^4 + \sum_i c_i^2 ||x||^2\right)\right) \qquad \text{(B.2.3)} \\
&= 4\left(||x||^2||v||^2 - ||x||^2\left(c^2||x||^2 + \sum_i c_i^2\right)\right) \\
&= 4(||x||^2||v||^2 - ||x||^2||v||^2) \\
&= 0,
\end{aligned}
$$

where the equality in the line B.2.2 is due to the orthogonality between x and $\sum_i c_i w_i$ and the inequality in line B.2.3 is due to $||x||^2$ being the largest eigenvalue.

Thus, we indeed find that $\nabla^2 g(x)$ is positive semi-definite, so the only local minima are $\pm x$, which completes the proof. □

Notice that in the above proof, it is not necessary for M to have rank 1 for us to achieve the same result. Thus, this proof will be easily generalizable to the case where M has arbitrary rank, and even when we observe noisy entries of M.

B.3 Convergence of GD Algorithms

We now prove the main theorems given in Section 4, or that Algorithm 1 of Section 3.1, the centralized gradient descent algorithm, will converge to the correct singular value and associated singular vectors with certainty, and that the algorithms in Sections 3.2 and 3.3 will also converge with probability 1 given sufficient conditions on thread execution order.

Theorem 4.0.1. *With probability* 1 *with respect to the random initialization, Algorithm* 1*, the centralized gradient descent algorithm for finding the largest singular value of a matrix*

M, correctly finds the largest singular value σ and singular vectors $u, v^\top$ corresponding to the largest singular value.

Proof. Given the result of Lemma B.2.1, we can assume that M is symmetric. This allows us to set $x = u\sigma^{\frac{1}{2}} = v^\top \sigma^{\frac{1}{2}}$ in the algorithm so that $f(\tilde{u}, v) = g(x)$. Now, running the algorithm without the normalization step, it suffices to avoid premature convergence to points where $\nabla g(x) = 0$ but $\nabla^2 g(x) \not\succeq 0$, as convergence to a local minima implies convergence to the global minima as well.

We find that with a suitable choice of step sizes and a slight modification of the simple gradient descent algorithm by using other techniques such as regularization or stochastic gradient descent, we can guarantee avoiding this premature convergence. Examples of these methods include cubic regularization as shown in Nesterov et al. [15] and stochastic gradient descent as shown in Ge et al. [16] and Bottou et al. [17]. Additionally, we can use modifications such as proximal algorithms as shown in Parikh et al. [18] and Attouch et al. [19], trust-region methods as shown in Sun et al. [20] if we choose to refine the process further.

A particularly useful black box result is Theorem 2.3 in Ge et al. [14], which we restate as Theorem C.0.1 in Section C of the appendix for clarity. It is clear that our function g satisfies the required constraints as all local minima are global minima, so we can thus apply Theorem 2.3 in [14] to generate algorithms that find $u, \sigma, v^\top$ as desired. □

Having shown the convergence for the simple gradient descent algorithm, we are now ready to show analogous results for the distributed gradient descent algorithms.

Theorem 4.0.2. *Suppose that the threads of Algorithm 2, the distributed gradient descent algorithm for finding the largest singular value of a matrix M, are executed in an order such that each of the gradient threads executes before the first iteration of and in between every two consecutive iterations of the aggregation thread. Then, with probability 1 with respect to the random initialization, the algorithm correctly finds the largest singular value σ and singular vectors $u, v^\top$ corresponding to the largest singular value.*

Proof. The proof here is relatively straightforward as it reduces to that of Theorem 4.0.1. When each of the gradient threads executes before the first iteration and in between every two consecutive iterations of the aggregation thread, each time the aggregation thread runs, the entirety of both partial gradients $\nabla_{\tilde{u}} f\left(\tilde{u}^{(t)}, v^{(t)}\right)$ and $\nabla_v f\left(\tilde{u}^{(t)}, v^{(t)}\right)$ have been updated to reflect the correct value given $\tilde{u}^{(t)}$ and $v^{(t)}$ from the current iteration. Note that it does not matter if a gradient thread runs multiple times in between two consecutive iterations of the aggregation thread, since any update after the first time the gradient thread runs will not change the chunk of the partial gradient that the gradient thread is responsible for computing.

Thus, the distributed algorithm in Section 3.2, Algorithm 2, will have precisely the same behavior as that of the centralized algorithm in Section 3.1. Thus, by Theorem 4.0.1 and potential small modifications that can be parallelized, we thus find the largest singular value σ and singular vectors $u, v^\top$ corresponding to the largest singular value, as desired. □

Using the previous theorem, the proof for the convergence of the fully distributed algorithm also follows. We restate Theorem 4.0.3 for convenience here before giving the proof.

Theorem 4.0.3. *Suppose that the threads of Algorithm* 3*, the distributed gradient descent algorithm for finding the r largest singular values of a matrix M, are executed in an order such that each of the gradient threads corresponding to the computation of the i^{th} largest singular value executes before the first iteration of and in between every two consecutive iterations of the aggregation thread corresponding to the computation of the i^{th} largest singular value for each $i \in [r]$. Then, with probability* 1 *with respect to the random initialization, the algorithm correctly finds the r largest singular values Σ_r and matrices of singular vectors $U_r, V_r^\top$ corresponding to the largest r singular values.*

Proof. The proof of this theorem follows directly from an inductive argument using Theorem 4.0.2, assuming that we do not prematurely stop the algorithm before convergence. By

Theorem 4.0.2, we will eventually converge to the correct singular value and singular vectors when $i = 1$, and when we have converged to the correct singular value and singular vectors for $i = 1, 2, \ldots, k$ for some $k \in [r]$, we see that

$$M_{(k+1)} = M - \sum_{i=0}^{k} u_{(i)} \sigma_{(i)} v_{(i)}^{\top}$$

is indeed the true matrix when the components corresponding to the top k singular values have been removed. Since Algorithm 2, the distributed algorithm, finds the top singular value and singular vectors associated with this singular value for $M_{(k+1)}$, the threads responsible for computing the $(k+1)^{\text{th}}$ largest singular value in Algorithm 3 will indeed correctly converge to the top singular value and singular vectors for $M_{(k+1)}$ after some time, which are precisely the $(k+1)^{\text{th}}$ largest singular value and singular vectors of M, completing the induction step.

Thus, after some time, we will eventually reach the state where all the gradients and aggregation threads corresponding to all r singular values have converged correctly to the top r singular values and singular vectors corresponding to the largest r singular values, as desired. □

Note that the argument relies on the fact that threads tasked with computing the k^{th} largest singular value and singular vectors use values from computations of the i^{th} largest singular value and singular vectors for $i \leq k$. Since there are no cyclic dependencies, the proof of the convergence of all r singular values follows easily.

We have thus shown the correct convergence of certain distributed algorithms for finding the top r singular values and singular vectors in the singular value decomposition of a matrix M. However, we often will not have the true values M_{true} but will only have noisy observations of M_{obs}, where $M_{\text{true}} = M_{\text{obs}} + M_{\text{err}}$ and M_{err} is bounded with nice properties. In order to handle the noise introduced by M_{err}, we show that our results are robust to small perturbances in the following section.

B.4 Handling Noise

Now that we have shown the convergence of the gradient descent algorithm to the correct singular value and singular vectors when given the true matrix M_{true}, it is only natural to ask for similar guarantees when given a noisy observation M_{obs} of M_{true}, where $M_{\text{obs}} = M_{\text{true}} + M_{\text{err}}$ for some bounded matrix M_{err}. We thus prove the following theorem regarding the noise of our observed matrix.

Theorem 4.0.4. *Suppose that we are only given a noisy observation M_{obs} of the true matrix M_{true}, where $M_{obs} = M_{true} + M_{err}$. Also, suppose that Σ_r is a diagonal matrix of the top r singular values $\sigma_{(1)}, \sigma_{(2)}, \ldots, \sigma_{(r)}$ of M_{true} and U_r, $V_r^\top$ are the corresponding matrices of singular vectors. Then, given bounds on the error matrix $||M_{err}|| < \left(1 - \frac{1}{\sqrt{2}}\right)\left(\sigma_{(r)} - \sigma_{(r+1)}\right)$, with probability 1 with respect to the random initialization, Algorithm 3 finds the matrix $\hat{\Sigma}_r$ of singular values $\hat{\sigma}_{(1)}, \hat{\sigma}_{(2)}, \ldots, \hat{\sigma}_{(r)}$ and corresponding matrices of singular vectors $\hat{U}_r, \hat{V}_r^\top$ such that*

$$|\sigma_{(i)} - \hat{\sigma}_{(i)}| \leq ||M_{err}||$$

for all $i \in [r]$ and

$$\max\{\mathsf{dist}(U_r, \hat{U}_r), \mathsf{dist}(V_r, \hat{V}_r)\} \leq \frac{2||M_{err}||}{\sigma_{(r)} - \sigma_{(r+1)}},$$

where $\mathsf{dist}(A, B)$ measures the sine of the angle between vector subspaces A and B.

Informally, this shows that even with some small amount of noise in our observation, we will still find singular values and corresponding singular vectors close to those of the true matrix M_{true}.

Proof. By Theorem 4.0.3, Algorithm 3 correctly finds the top r singular values and corresponding singular vectors of M_{obs}. Thus, it suffices to show that the singular values and vectors computed from the algorithm cannot differ too much from those of the true matrix M_{true}.

This is quite simple when $||M_{\text{err}}||$ is bounded, as we can use previous findings on matrix perturbations to show precisely what we need. Some results that suffice are Weyl's inequality for singular values and Wedin's $\sin\theta$ theorem, which we have restated in Section C as Theorems C.0.3 and C.0.4 for convenience. Applying these theorems directly allows us to conclude precisely the above statements as given in the theorem, which completes the proof. □

Appendix C

Helper Lemmas/Theorems

We dedicate this chapter of the appendix to significant results from other sources used in the proofs of our findings.

Theorem C.0.1 (Theorem 2.3 of [14]). *[15, 16, 20] Let f be a twice-differentiable function from $\mathbb{R}^d$ to $\mathbb{R}$. Suppose there exist ε_0, τ_0 and a universal constant $c > 0$ such that if a point x satisfies $||\nabla f(x)|| \leq \varepsilon \leq \varepsilon_0$ and $\nabla^2 f(x) \succeq -\tau_0 \cdot I$, then x is ε^c-close to a global minimum of f. Then, many optimization algorithms including cubic regularization, trust-region, and stochastic gradient descent, can find a global minimum of f up to δ error in l_2 norm in the domain in time* poly $\left(\frac{1}{\delta}, \frac{1}{\tau_0}, d\right)$.

Lemma C.0.2 (Lemma 3.2 of [14], slightly modified). *Suppose that $M = zz^\top$ is a fully observed matrix with rank $r = 1$. Then the function $g(x) := \frac{1}{2}||M - xx^\top||_F^2$ has only two local minima $\{\pm z\}$.*

Theorem C.0.3 (Weyl's inequality for singular values, as stated in Lemma 2.3 of [21] and in [22]). *Suppose that we have a matrix M_{true} and a noisy observation of the matrix $M_{obs} = M_{true} + M_{err}$. Let $\sigma_{(i)}, \hat{\sigma}_{(i)}$ denote the i^{th} largest singular values of M_{true} and M_{obs}, respectively. Then, for all $i \in [d]$, it is true that*

$$|\sigma_{(i)} - \hat{\sigma}_{(i)}| \leq ||M_{err}||.$$

Theorem C.0.4 (Wedin's $\sin\theta$ theorem, as stated in Theorem 2.9 of [21], Line 2.26)**.** *Suppose that we have a matrix M_{true} and a noisy observation of the matrix $M_{obs} = M_{true} + M_{err}$. Also, suppose that Σ_r is a matrix of the top r singular values $\sigma_{(1)}, \sigma_{(2)}, \ldots, \sigma_{(r)}$ of M_{true} with corresponding matrices of singular vectors $U_r, V_r^\top$, and that $\hat{\Sigma}_r$, $\hat{U}_r, \hat{V}_r^\top$ are the corresponding matrices for M_{obs}. If $||M_{err}|| < \left(1 - \frac{1}{\sqrt{2}}\right)\left(\sigma_{(r)} - \sigma_{(r+1)}\right)$, then*

$$\max\{\mathsf{dist}(U_r, \hat{U}_r), \mathsf{dist}(V_r, \hat{V}_r)\} \leq \frac{2||M_{err}||}{\sigma_{(r)} - \sigma_{(r+1)}},$$

where $\mathsf{dist}(A, B)$ measures the sine of the angle between vector subspaces A and B.

References

[1] J. Francis. "The QR transformation: A unitary analogue to the LR transformation". In: *Numerische Mathematik* 4.3 (1961), pp. 254–282.

[2] N. Halko, P. G. Martinsson, and J. A. Tropp. "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions". In: *SIAM Review* 53.2 (2011), pp. 217–288.

[3] A. D. Alberto Abadie and J. Hainmueller. "Synthetic Control Methods for Comparative Case Studies: Estimating the Effect of California's Tobacco Control Program". In: *Journal of the American Statistical Association* 105.490 (2010), pp. 493–505. DOI: 10.1198/jasa.2009.ap08746. eprint: https://doi.org/10.1198/jasa.2009.ap08746. URL: https://doi.org/10.1198/jasa.2009.ap08746.

[4] D. Shah, D. Song, Z. Xu, and Y. Yang. *Sample Efficient Reinforcement Learning via Low-Rank Matrix Estimation*. 2020. arXiv: 2006.06135 [cs.LG]. URL: https://arxiv.org/abs/2006.06135.

[5] A. Agarwal, A. Alomar, and D. Shah. *On Multivariate Singular Spectrum Analysis and its Variants*. 2022. arXiv: 2006.13448 [cs.LG].

[6] A. Agarwal, M. Dahleh, D. Shah, and D. Shen. *Causal Matrix Completion*. 2021. arXiv: 2109.15154 [econ.EM].

[7] S. Bhattacharya and S. Chatterjee. *Matrix completion with data-dependent missingness probabilities*. 2022. arXiv: 2106.02290 [math.ST].

[8] N. Halko and P. G. Martinsson. "Randomized Algorithms for Large-Scale Matrix Computations". In: *Proceedings of the 2018 IEEE International Conference on High Performance Computing*. 2018, pp. 22–30.

[9] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, 1998.

[10] E. Anderson, Z. Bai, C. Bischof, L. Blackford, J. Demmel, J. Dongarra, ..., and D. Sorensen. *LAPACK Users' Guide*. SIAM, 1999.

[11] L. Struski, P. Morkisz, P. Spurek, S. R. Bernabeu, and T. Trzcinski. "Efficient GPU implementation of randomized SVD and its applications". In: *Expert Systems with Applications* 248 (2024), p. 123462. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2024.123462. URL: https://www.sciencedirect.com/science/article/pii/S0957417424003270.

[12] G. W. Stewart and J. guang Sun. *Matrix Perturbation Theory*. Academic Press, 1990.

[13] *MovieLens Dataset from Tensorflow*. 2024. URL: https://www.tensorflow.org/datasets/catalog/movielens.

[14] R. Ge, J. D. Lee, and T. Ma. "Matrix Completion has No Spurious Local Minimum". In: *CoRR* abs/1605.07272 (2016). arXiv: 1605.07272. URL: http://arxiv.org/abs/1605.07272.

[15] Y. Nesterov and B. Polyak. "Cubic regularization of Newton method and its global performance". In: *Math. Program.* 108 (Aug. 2006), pp. 177–205. DOI: 10.1007/s10107-006-0706-8.

[16] R. Ge, F. Huang, C. Jin, and Y. Yuan. *Escaping From Saddle Points — Online Stochastic Gradient for Tensor Decomposition*. 2015. arXiv: 1503.02101 [cs.LG]. URL: https://arxiv.org/abs/1503.02101.

[17] L. Bottou, F. E. Curtis, and J. Nocedal. *Optimization Methods for Large-Scale Machine Learning*. 2018. arXiv: 1606.04838 [stat.ML]. URL: https://arxiv.org/abs/1606.04838.

[18] N. Parikh and S. P. Boyd. "Proximal Algorithms". In: *Found. Trends Optim.* 1 (2013), pp. 127–239. URL: https://api.semanticscholar.org/CorpusID:51791656.

[19] H. Attouch, J. Bolte, and B. Svaiter. "Convergence of descent methods for semi-algebraic and tame problems: Proximal algorithms, forward-backward splitting, and regularized Gauss-Seidel methods". In: *Mathematical Programming* 137 (Jan. 2011). DOI: 10.1007/s10107-011-0484-9.

[20] J. Sun, Q. Qu, and J. Wright. *When Are Nonconvex Problems Not Scary?* 2016. arXiv: 1510.06096 [math.OC]. URL: https://arxiv.org/abs/1510.06096.

[21] Y. Chen, Y. Chi, J. Fan, and C. Ma. "Spectral Methods for Data Science: A Statistical Perspective". In: *Foundations and Trends in Machine Learning* 14.5 (2021), pp. 566–806. ISSN: 1935-8245. DOI: 10.1561/2200000079. URL: http://dx.doi.org/10.1561/2200000079.

[22] T. Tao. *Topics in Random Matrix Theory*. American Mathematical Society, 2012.