Applications of Stochastic Gradient Descent to Nonnegative Matrix Factorization

by

Matthew William Slavin

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2019

Abstract

We consider the application of stochastic gradient descent (SGD) to the nonnegative matrix factorization (NMF) problem and the unconstrained low-rank matrix factorization problem. While the literature on the SGD algorithm is rich, the application of this specific algorithm to the field of matrix factorization problems is an unexplored area. We develop a series of results for the unconstrained problem, beginning with an analysis of standard gradient descent with a known zero-loss solution, and culminating with results for SGD in the general case where no zero-loss solution is assumed. We show that, with initialization close to a minimizer, there exist linear rate convergence guarantees.

We explore these results further with numerical experiments, and examine how the matrix factorization solutions found by SGD can be used as machine learning classifiers in two specific applications. In the first application, handwritten digit recognition, we show that our approach produces classification performance competitive with existing matrix factorization algorithms. In the second application, document topic classification, we examine how well SGD can recover an unknown words-to-topics matrix when the topics-to-document matrix is generated using the Latent Dirichlet Allocation model. This approach allows us to simulate two regimes for SGD: a fixed-sample regime where a large set of data is iterated over to train the model, and a generated-sample regime where a new data point is generated at each training iteration. In both regimes, we show that SGD can be an effective tool for recovering the hidden words-to-topic matrix. We conclude with some suggestions for further expansion of this work.

Acknowledgements

I would like to thank everyone at the University of Waterloo who have aided me throughout my degree and helped smooth my transition from industry into graduate studies.

I would like to thank my supervisor, Professor Steve Vavasis, for his guidance, patience, and imparted wisdom; it has been a privilege to explore the field of optimization together over the past two years. I thank him for supporting me in my transition to a new field, and for the opportunities this has provided for my future.

I would like to thank the readers of my thesis, Professors Henry Wolkowicz and Levent Tunçel, for thoughtful comments and suggestions. Thank you to the various instructors within the Combinatorics and Optimization department who I have had the chance to work with or learn from; this university is a more incredible learning environment than I ever could have appreciated coming in.

Thank you to my classmates and peers with whom I have shared this journey, through all of the ups and downs. In particular, I would like to thank my officemates Stefan, Alex, Jimit, and Kris for their collaborations and crossword prowess.

Finally, thank you to my friends from both within and outside the university community for making these last two years memorable beyond belief. From late night frisbee to dinners at the Club, you are the foundation on which my graduate studies experience has been built. Bryce, Goose, Angus, Andrew, Jon, Bradley, Peter (x2), Jacquie, and many others: I could fill a second thesis with all of our stories and memories, and for that you have my eternal gratitude.

Dedication

To my family and friends, who give meaning to my search for a global optimum.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Our Problem

In this thesis, we consider the following problem: given $V \in \mathbb{R}^{m \times n}$ and a positive integer r, find factor matrices $W \in \mathbb{R}^{m \times r}$ and $H \in \mathbb{R}^{r \times n}$ to minimize:

$$\begin{aligned} &\min ||V - WH||_F^2 \\ &\;s.t.\ W \geq 0,\ \ H \geq 0, \end{aligned} \tag{1.1}$$

where $||.||_F$ denotes the Frobenius norm. This is the nonnegative matrix factorization (NMF) problem [30], and in recent years it has gained significant popularity within both the computer science and optimization communities due to its applicability to modern day data science problems.

While there exists a wide literature on approaches for solving this problem, we consider a novel approach: applying the popular Stochastic Gradient Descent (SGD) method to solve (1.1). SGD has gained popularity in recent years due to its wide applicability to many data science problems [10], particularly within the deep learning community as a model training algorithm.

We aim to use the NMF problem as a means to study the properties of SGD by both directly analyzing the algorithm itself and conducting numerical experiment to understand its applications. Through this, we study the applications of both the NMF problem and the SGD algorithm through the lenses of mathematical optimization, data science, and machine learning.

1.2 Motivation

The use of SGD to solve matrix factorization problems has gained interest in recent years, especially in applications where the size of the data set makes Gradient Descent less tractable. A particular application of note, partially due to its popularity outside of the optimization community, is the so-called "Netflix Prize" competition, where researchers were challenged to drive a significant improvement in the performance of the recommender systems used by Netflix to suggest content to viewers [43]. Recommender systems utilize data on user preferences to recommend "items (e.g. products, services, etc.) that the user is more likely to be interested in. The most popular technique for building a recommender system is called "collaborative filtering"; here, a new user is compared to existing users that the data suggests have similar tastes, and the new user is recommended items that the similar existing users have rated highly [62].

In their prize-winning submission in the Netflix competition, Koren, Bell, and Volinsky developed a collaborative filtering system that involved performing SGD to solve a matrix factorization problem [43]. While their approach involved some problem-specific modification (including regularization, introduced biases, etc.), the core approach was driven by randomly selecting an entry of the data matrix, and performing an SGD step based on this entry.

Although the problem addressed by Koren et al. did not involve a nonnegativity constraint, it provided justification that a stochastic approach to matrix factorization can lead to success. Below, we discuss some specific examples similar to the approach we propose, and note the differences that makes our approach novel.

In our work, we note that our primary goal is not to identify a more efficient or better performing NMF algorithm or general matrix factorization algorithm. This is a highly active research area, and there are many existing algorithms that are highly efficient; indeed, in the general matrix factorization case, one would simply default to a singular value decomposition to solve the problem efficiently (see 2.1.1). Instead, our goal is to specifically understand how SGD behaves when applied to these problems, and use these findings to better understand the SGD algorithm itself. This goal is specifically reflected in the Experiments Chapter 5, where we do not seek to compare SGD to existing NMF algorithms from a computer-time or efficiency standpoint.

1.3 Related Work

Gemulla et al. considered the problem of solving very large-scale matrix factorization using SGD that could be distributed and run in a decentralized manner [28]. In their paper, motivated by the work of Koren et al. discussed above, they discussed finding low-rank matrix approximations, with the addition of a "stratification" of SGD that allowed the algorithm to proceed in parallel on multiple devices. The authors formulate SGD for matrix factorization in the centralized case, but do not offer any further analysis on this case. In addition, they briefly mention using a nonnegative orthant projection to extend their approach to solving the NMF problem, but do not discuss this extension in detail.

In 2007, Lin addressed using Projected Gradient Descent to solve the NMF problem [49]. His approach consists of a full Gradient Descent step to update W and H, followed by a projection onto the nonnegative orthant. In addition, Lin employs a line-search method for updating the step-size for each iteration, based on the Armijo condition [49] [55]. This approach demonstrates how full Gradient Descent can be used to solve the NMF problem, but does not discuss the stochastic variant for Gradient Descent in any detail.

More recently, Davis and Drusvyatskiy analyzed the convergence rate of the projected stochastic subgradient method [18]. The algorithm they study involves selecting a stochastic subgradient for each iterate, and selecting the next iterate using a proximal mapping after the subgradient update. While we do not discuss the theory of proximal mappings in detail here (more detail on proximal mappings is available in [59]), we recognize that for the NMF problem, the proximal mapping is simply the projection onto the nonnegative orthant; thus, their algorithm can be seen as a generalization of our main algorithm.

Davis and Drusvyatskiy show that, under mild assumptions on the objective function and stochastic gradients, the algorithm will converge to a point with small proximal gradient at a sublinear rate. Drusvyatskiy and Paquette showed that a small proximal gradient is equivalent to the current iterate being "near-stationary", or "close" to a stationary point in an appropriate distance measure [23].

While this result provides a guarantee on the size of the stochastic gradient, this is insufficient for our problem, where a vanishing gradient does not immediately guarantee a solution to (1.1). For this reason, in our analysis, we aim to prove results that speak specifically to how well the product WH approximates the data matrix V.

1.4 Organization of this Thesis

In Chapter 2, we provide the background material necessary to approach this problem. We introduce the NMF problem formally (2.1.2), as a subset of a family of matrix factorization problems. We consider other matrix factorization problems, including the weighted low-rank matrix factorization problem (2.1.3) and the L_1 norm low-rank matrix factorization problem (2.1.4). We also introduce the family of first-order optimization methods, including Gradient Descent (2.2.1), and its stochastic variant, SGD (2.2.2). We also introduce some fundamental machine learning concepts (2.3.1), and explain how matrix factorization has been used as a machine learning tool in the literature (2.3.3).

In Chapter 3, we formally define the problem we seek to solve (3.1), and develop the algorithms required to approach it (3.2). In Chapter 4, we state and prove a number of results related to the application of these algorithms to the matrix factorization problems introduced.

In Chapter 5, we demonstrate our results numerically (5.1), and consider two specific machine learning applications for our problem: handwritten digit classification (5.2), and latent topic identification (5.3). Finally, in Chapter 6, we state some possible next steps for this work.

Chapter 2

Background

2.1 Matrix Factorization Preliminaries

2.1.1 Dimensionality Reduction

Matrix factorization techniques are an example of a broader range of so-called dimensionality reduction techniques. Dimensionality reduction seeks to transform a dataset from a high-dimensional space into a low-dimensional space, with the goal of using the lower-dimensional representation to isolate important properties of the dataset [30]. In this sense, dimensionality reduction can be seen as a fundamental process in the field of data science.

A wide variety of commonly-used data science tools have their roots in dimensionality reduction. Principal component analysis (PCA) aims to represent a dataset as combinations of uncorrelated principal components; here, dimensionally reduction is necessary to ensure the number of components is small [8]. Other algorithms such as nearest-neighbour classification [65] and various regression techniques use dimensionality reduction to simplify models and reduce overfitting. Dimensionality reduction techniques are critical in the so-called Big Data regime, where large amount of (potentially) high-dimensional data is readily available. While large datasets are unwieldy and difficult to interpret, their low-dimensional approximations are more easily stored and transported electronically, and are often easier to represent visually in charts and diagrams.

Given a set of data vectors v_i, a straightforward linear dimensionality reduction model can be built as follows [30]:

$$v_i \approx \sum_{j=1}^{r} y_j c_i(j), \tag{2.1}$$

where y_i represents a basis vector in the reduced dimension space, and c_i represents the coefficients for representing v_i in that basis. By assembling all i equations into a matrix, we obtain:

$$V \approx YC. \tag{2.2}$$

Note that we have adjusted the formulation to an approximation. This is to recognize that most general data matrices V will not have an exact low-rank factorization, so the best we can hope for is a low-rank approximation.

This formulation is the low-rank matrix factorization problem: the goal is to determine both the best lower-dimensional basis elements that represent the data, as well as the coefficients that represent each data point using these basis elements. Specifically, we define the following low-rank matrix factorization problem:

$$\begin{aligned} &\min\ f(W,H) \\ &s.t.\ rank(W) \leq r, \end{aligned} \tag{2.3}$$

where f represents a loss function used to model how accurate the approximation is, and r is a rank parameter selected in advance. In the literature for this problem, the most common loss function used is the squared Frobenius norm loss:

$$f(W,H) = ||V - WH||_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} (V(i,j) - W(i,:)H(:,j))^2, \tag{2.4}$$

where $X(i,j)$ indicated the ith row, jth column entry of V. While other loss functions are possible, the use of the Frobenius Norm is well-motivated by its compatibility with the Singular Value Decomposition (see below); as an example, given the singular values of a matrix X, it is straightforward to compute the value $||X||_F$. [34].

The low-rank matrix factorization problem (2.3) above can be efficiently solved using the Singular Value Decomposition (SVD) of the matrix V. For all real-valued matrices, we have the following result [34]:

Theorem 1. *If $V \in \mathbb{R}^{m \times n}$, then $\exists$ orthogonal matrices $A \in \mathbb{R}^{m \times m}$ and $B \in \mathbb{R}^{n \times n}$, and a diagonal matrix $\Sigma \in \mathbb{R}^{m \times n}$ such that*

$$V = A\Sigma B^T,$$

where we denote the diagonal entries of Σ as $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_{min(m,n)} \geq 0$.

The SVD is an exact decomposition of the full-rank matrix. It can be thought of in a similar vein to representing a given vector in an orthonormal basis [8]. The following theorem from Eckart, Young, and Mirsky [24] makes it clear how the SVD is valuable for dimensionality reduction:

Theorem 2. *Let $V \in \mathbb{R}^{m \times n}$ with $m \geq n$, and let $V = A\Sigma B^T$ be the singular value decomposition of V. Let $r \leq n$. Then a best rank-r approximation to V in terms of the Frobenius norm, (which we denote V_r):*

$$V_r := \underset{X}{\operatorname{argmin}}\{||V - X||_F : \ rank(X) \leq r\},$$

is given by

$$V_r = A(:, 1:r)\Sigma(1:r, 1:r)B(:, 1:r)^T.$$

In the above theorem, we use MATLAB notation for indexing matrices; specifically, $A(:, 1:r)$ indicates the sub-matrix of A consisting of all rows (indicated by the ":"), and the columns 1 through r (indicated by the $1:r$).

In short, we can find the best r-dimensional approximation to the matrix V by finding the SVD of V, truncating A, Σ, and B after the first r singular vectors, and multiplying the truncated matrices back together to obtain V_r. Furthermore, the SVD provides us with an easy tool to determine the Frobenius norm (or the Frobenius norm squared) of a matrix V. For $V \in \mathbb{R}^{m \times n}$, we have [34]:

$$||V||_F^2 = \sigma_1^2 + ... + \sigma_{min(m,n)}^2. \tag{2.5}$$

Given that a straightforward best rank-r approximation exists through the truncated SVD, one can ask why the matrix factorization problem is so extensively studied. One reason is that, in general, the low-rank matrix factorization problem does not permit a unique solution. Observe that a pair of matrices that are inverses of each other can be inserted in between the W and H factors:

$$WH = W(RS)H = (WR)(SH) = \tilde{W}\tilde{H} \text{ (where } RS = I\text{)}. \tag{2.6}$$

As a result of this non-uniqueness, it is difficult to expect the optimizer factor matrices to have any specific or helpful structure. As an example, consider the PCA application introduced above. We would expect a traditional solution to this problem to have many non-zero components; however, for practical applications, we may desire a PCA solution with only a few non-zero components. By enforcing a sparsity constraint (example of which include so-called sparse PCA [76], or the LASSO approach [38]), each principal component can be viewed as a combination of fewer original variables, thus making the results easier to interpret. In the next few sections, we discuss some specific extensions of the low-rank matrix factorization problem, beginning with nonnegative matrix factorization.

2.1.2 Nonnegative Matrix Factorization

Nonnegative matrix factorization (NMF) was first introduced in 1974 by Thomas, in answering a question posed by Plemmons and Berman [3] [69]. It gained popularity due to the work of Paatero and Tapper [57] and Lee and Seung [46] as a method for representing a dataset as a nonnegative combination of a set of basis vectors with reduced rank r. This was referred to by Lee and Seung as a “parts-based” representation. NMF imposes an additional (and seemingly straightforward) constraint on the low-rank matrix factorization problem that the two product matrices contain entirely nonnegative entries. The authors argued that this parts-based representation permitted a variety of easily-interpretable real-world applications, (one of which is discussed below). In particular, the NMF formulation offers a different interpretation from PCA by not allowing negative coefficients for the basis vectors [46].

A natural question to ask is which common data science problems (if any) permit solutions with a parts-based representation that NMF requires. The answer is yes; while we will discuss only one example in particular, there are numerous immediate applications of the NMF problem to the fields of gene profiling [72], feature extraction in image processing [36], and unmixing of spectral signals [6].

We highlight a particular example to demonstrate the benefits of the nonnegativity constraint: using NMF to identify topics in a document corpus. Let V represent a collection of text documents, with one row for each word used in the documents and one column for each document (note that the row dimension m will be very large, so this example is appropriate in the Big Data regime). Each matrix entry denotes the frequency of a given word appearing in a specific document; as a result, V will be nonnegative. We now assume that an approximate factorization of the following form can be found:

$$\begin{aligned} V \approx WT,\ (W \in \mathbb{R}^{m\times r},\ T \in \mathbb{R}^{r\times n}) \\ s.t.\ W \geq 0,\ T \geq 0. \end{aligned} \tag{2.7}$$

By preserving the nonnegativity of the product matrices, we have two product matrices that can be interpreted in a similar light to V. The W matrix can be seen to represent the number of words that appear in r topics, while the T matrix can be seen to represent how often each topic is discussed in each document [30]. Thus, the NMF solution to this problem has both identified latent topics (i.e. the r basis vectors) in the document corpus, and has also classified each document by how frequently it discusses each topic. This interpretation would not have been possible without the nonnegativity constraints, which assures that the documents can only be interpreted as a nonnegative combination of topics [64].

We now define the NMF problem formally: given $V \in \mathbb{R}^{m\times n}$, we seek factor matrices $W \in \mathbb{R}^{m\times r}$ and $H \in \mathbb{R}^{r\times n}$ to minimize:

$$\begin{aligned} \min\ &||V - WH||_F^2 \\ s.t.\ &W \geq 0,\ H \geq 0. \end{aligned} \tag{2.8}$$

We note that, in this definition, r must be pre-selected as a parameter for the problem. As in the case for the low-rank matrix factorization problem, the use of the squared Frobenius norm as the loss-function is a natural choice.

When they first considered NMF, Lee and Seung also proposed a family of methods to solve the problem. These are commonly referred to as multiplicative updates [47]:

$$W_{k+1}(i,j) = W_k(i,j)\frac{(VH_k^T)(i,j)}{(W_kH_kH_k^T)(i,j)}, \quad H_{k+1}(i,j) = H_k(i,j)\frac{(W_k^TV)(i,j)}{(W_k^TW_kH_k)(i,j)}. \tag{2.9}$$

These methods update W and H individually (a procedure mirrored by almost all standard NMF algorithms), and in such a way that the Frobenius norm loss in NMF above is guaranteed to shrink on each step. However, the fact that multiple matrix products need to be computed on each step suggests that there are potential computation savings available [30]. In addition, when an element of W or H is set to zero in this algorithm, it will remain zero for all future iterates. This will bias the algorithm in the direction of the first fixed point that the algorithm moves towards, regardless of whether this is a "good" fixed point to pursue [4].

These limitations motivate an alternative set of methods, where each update step is determined by solving the least-squares subproblem. A straightforward approach is to solve the least-squares subproblem without the nonnegativity constraints, and then project each entry back onto the nonnegative orthant [17] [4]. Specifically, we have:

$$\begin{aligned} W_{k+1} &= \max\{\underset{Z}{\operatorname{argmin}}\{||V - ZH_k||_F^2\}, 0\}, \\ H_{k+1} &= \max\{\underset{Z}{\operatorname{argmin}}\{||V - W_k Z||_F^2\}, 0\}. \end{aligned} \tag{2.10}$$

This update has the benefit of naturally introducing sparsity into the solution matrices [4], which can be beneficial for many real-world applications. While this is computationally inexpensive, performance can often be poor [30]; in addition, it can be difficult to analyze the performance of this approach, since this approach essentially solves a constrained problem using unconstrained methods [42]. More commonly, algorithms will seek to solve the least-squares subproblem exactly <u>including</u> the nonnegativity constraint for each product matrix [49]:

$$\begin{aligned} W_{k+1} &= \underset{Z\geq 0}{\operatorname{argmin}}\{||V - ZH_k||_F^2\}, \\ H_{k+1} &= \underset{Z\geq 0}{\operatorname{argmin}}\{||V - W_k Z||_F^2\}. \end{aligned} \tag{2.11}$$

A helpful feature of this approach is that, since the desired constraints are built directly into the subproblems, the limit points of any algorithm that solves the subproblems will also be stationary points for the NMF problem [42]. The selection of methods for how to solve this problem is broad (see [30] [17] [42]); one method, known as the Active Set method, is very popular and is implemented as a standard function in MATLAB [42]. An additional method, Hierarchical Alternating Least Squares (HALS), simplifies the nonnegative subproblem by updating each column of W and row of H individually [17]. In this thesis, our main focus is on gradient-based methods (see 2.2.1).

Despite the existing methods for solving NMF, there are some key challenges that make the problem worth further study. For one, the uniqueness issue that exists in the low-rank matrix factorization problem persists in the NMF problem [29]. In some specific cases of non-uniqueness, such as permutations or scalings of the columns of W and H, this non-uniqueness is less concerning and can be easily controlled [32]. In the general case, however, this poses issues when trying to apply the NMF solution to a real-world application. There are methods of ensuring that a data matrix will have a unique NMF up to permutations and scaling based on the structure of the data matrix itself [21] [44], but these are not considered further here.

In addition, NMF is NP-hard to solve in general [71]. Thus, we will either work with instances where the NMF is initialized such that a known zero-loss solution exists in, or be content with finding a low-rank approximation. Under specific assumptions about the structure of the data matrix V (specifically the separability assumption, see below), there exist algorithms that are guaranteed to find an NMF solution in linear time [32].

NMF: The Separable Case

Donoho and Stodden [21] introduced a special case of the NMF problem that is of significant note for both its applications to real-world problems and performance under specific algorithms. Separable NMF assumes that each column of W appears as a column of V, up to a scaling factor. As a result, we would expect to recover an H matrix that contains r identity columns, corresponding to these columns of V that appear in W. This assumption has a very elegant geometric interpretation, and many applications of NMF exhibit this separable assumption naturally in the data [32].

The study of the separable case is useful, in part, because the separability assumption appears naturally in many real-world applications [32]. As a specific example, consider the text classification example introduced above. The separable assumption require that, for each topic, there is a document that discusses only that topic [30]. This turns out to be a very reasonable assumption; this is of note, because as mentioned above, there exist linear time algorithms for solving NMF under the separable assumption.

While we do not analyze the separable case specifically, we do mention additional applications where the separable assumption is relevant (see 2.3.3), and it provides motivation for how to formulate our main algorithm of interest (see 3.2.1).

2.1.3 Weighted Low-Rank Matrix Factorization

An alternative extension to the low-rank matrix factorization problem is to consider a weighted variation, where each entry of the data matrix has a corresponding weight [13] [31]. In this variation, given a data matrix V, a matrix of nonnegative weights Ω, and a desired rank r, we seek matrices $W \in \mathbb{R}^{m \times r}$ and $H \in \mathbb{R}^{r \times n}$ that minimizes:

$$min \sum_{i,j} \Omega_{ij}^2 (V - WH)_{ij}^2 = ||\Omega \odot (V - WH)||_F^2, \tag{2.12}$$

where $\odot$ represents the Hadamard, or element-wise product. This generalization provides the ability to model situations where some entries of V are more accurately known than others, and thus a more significant penalty should be incurred if these entries are not well approximated. In addition, by restricting the entries of Ω to be 0 or 1, this formulation can be used to model a data matrix that has missing entries.

This has immediate applications to the so-called matrix completion problem [14], where the existing entries of the matrix are used to predict the unknown entries. One approach to the matrix completion problem is to first develop a low-rank approximation for the available entries, and use the found factor matrices to estimate the unknown entries. This is the basis for recommender systems, which are used by companies such as Netflix to estimate user ratings for films that they have not viewed, based on the ratings the user has selected for other films [43]. As with the NMF problem, this matrix factorization aims to discover latent features (i.e. basis vectors) in the data, and to fill the unknown matrix entries using combinations of these features.

The addition of the weight matrix makes the low-rank factorization problem significantly more difficult. In the non-weighted case, all of the local minima for the loss function are in fact global optima, which directly leads to the easy ability to find a solution through the SVD [67]. However, in the weighted case, this helpful structure for the critical points is lost, greatly complicating the problem. In fact, even in the case where simply a rank-one solution is desired, the weighted problem is NP-hard to solve (and thus, it is NP-hard to solve for any desired rank) [31].

Srebro and Jaakkola discussed using gradient-descent techniques (as described in 2.2.1) to solve (2.12) [67]. They found that, while the algorithm could theoretically converge to unhelpful local minimum points, in practice Gradient Descent tended to converge to the true global minimum. In the same paper, the authors also suggested an Expectation-Maximization (EM) algorithm approach, which alternates between estimating the missing values of V from the current iterates of W and H, and then updating W and H based on the newly completed V [67]. Other approaches, including second-order Newton-type algorithms [13] have been shown to have strong numerical performance. Furthermore, there exist algorithms based on matrix sketching techniques that produce an ϵ-approximate solution with high probability [61].

2.1.4 L_1 Norm Low-Rank Matrix Factorization

In many practical machine-learning and data science applications, the L_2 or Frobenius norm is not the most appropriate loss function for modeling the problem in question [74]. A specific example is compressed sensing, where we seek a solution to a linear system with an additional sparsity constraint (i.e. the solution found must have a low number of non-zero entries) [25]. For this reason, and due to the sensitivity of the L_2 norm to noise [74], matrix factorization using the L_1 norm as the loss function has gained significant popularity [41]. For a matrix A, we have:

$$||A||_1 = \sum_i \sum_j |A(i,j)|. \tag{2.13}$$

We seek matrices W and H to minimize the following:

$$\min \; ||V - WH||_1. \tag{2.14}$$

This formulation can be extended to capture the weighted case (and the nonnegative case, NMF) as well:

$$\min \; ||\Omega \odot (V - WH)||_1. \tag{2.15}$$

In both the weighted and unweighted case, however, the L_1 problem is difficult to solve, as the problem is non-convex in (W, H) and non-differentiable. Indeed, even the unweighted case (2.14) is NP-hard to solve in general [33].

Similar to the alternating least squares approach to the NMF problem, Ke and Kanade proposed an alternating convex minimization approach to solving (2.14), where each iterate is updated as follows) [41] :

$$\begin{aligned} H_k &= \underset{X}{\operatorname{argmin}} \, ||V - W_{k-1}X||_1, \\ W_k &= \underset{X}{\operatorname{argmin}} \, ||V - XH_k||_1. \end{aligned} \tag{2.16}$$

The first sub-problem above can be decomposed into n smaller sub-problems, each of which can be solved using standard convex optimization methods. In particular, each of the sub-problems can be formulated as a linear program, which can be efficiently solved by a variety of algorithms. It is worth noting that, in the specific instance of (2.15) where Ω is a 0-1 matrix, which models missing data from V, it is easy to simply "drop" the constraints for the missing entries of V, leading to a straightforward extension of this approach to the weighted case.

Eriksson and van den Hengel proposed a similar approach to solve (2.15) using the so-called Wiberg method [56], an algorithm for matrix factorization that has regained popularity due to its applications to computer vision problems [27]. The Wiberg method recognizes that, if W and H are respectively fixed, then the optimizers to (2.15), denoted W^* and H^*, can be found as follows:

$$H_j^*(W) = \underset{H_j}{\operatorname{argmin}} ||\Omega \odot V_j - \Omega \odot (I_n \otimes W)H_j||_1, \tag{2.17}$$

$$W_j^*(H) = \underset{W_j}{\operatorname{argmin}} ||\Omega \odot V_j - \Omega \odot (H^T \otimes I_m)W_j||_1. \tag{2.18}$$

By substituting (2.17) into (2.18), we obtain the following equivalent statement for the optimal value of (2.15), parametrized by W [27]:

$$||\Omega V_j - \Omega W H^*(W)||_1 := ||\Omega V_j - \phi(W)||_1. \tag{2.19}$$

Although $\phi(W)$ is a non-linear, non-differentiable term, the gradient can be approximated using a first-order Taylor expansion for each iterate U_k. By repeating this update rule, (2.15) can be solved.

Other approaches to solving (2.14) and (2.15) include statistical models such as a variational Bayesian approach proposed by Zhao et. al [74]; these types of algorithms are not detailed further here.

2.2 First-Order Optimization Methods

2.2.1 Gradient Descent

Gradient Descent is a well-known optimization method, often attributed to Cauchy in 1847 [15]. Gradient Descent is the canonical example of a first-order optimization method, meaning it makes use of information concerning the gradient of the objective function in order to determine an update direction. In its most basic form, Gradient Descent updates an iterate by taking a small step in the opposite direction of the objective function gradient, often referred to as the steepest descent direction [12]:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k). \tag{2.20}$$

This update step can be shown to minimize the first-order Taylor approximation of the objective function [12]. For many objective functions, this first-order approximation is sufficient to ensure that progress is made towards an optimizer. Indeed, as we will note below, there exists a large class of problems for which Gradient Descent is guaranteed to find the optimal solution.

The parameter α_k indicates the step-size (or learning rate) of the problem. Typically, this parameter is chosen to be small, as the accuracy of the first-order Taylor approximation will decrease as the step-size increases. However, choosing a step-size that is very small will force the algorithm to perform more iterations in order to reach a solution. As a result, there exists a trade-off when selecting a step-size. A commonly-used method is to choose a fixed step-size; this is a favored method due to its simplicity, but will often require tuning in order to improve algorithm performance. An alternative approach involves using a line-search to identify a step-size that is guaranteed to decrease the objective function by some fixed amount [55]. This sufficient-decrease condition, also known as the Armijo condition [49], adds an additional computation to the Gradient Descent procedure, but aims to compensate by ensuring each iteration of the algorithm makes significant progress towards the optimizer.

In order to discuss the performance of Gradient Descent for common problem classes, we must first discuss the goals of optimization in general. When solving an optimization problem (we will assume a minimization problem over the space $\mathbb{R}^n$ for simplicity), one seeks points of the following forms:

- (Global Minimizer): x^* such that $f(x^*) \leq f(x)\ \forall x \in \mathbb{R}^n$
- (Local Minimizer): x^* such that, for some positive radius r, $f(x^*) \leq f(x)\ \forall x \in \mathbb{B}_r(x^*)$
- (Stationary Point): x^* such that $\nabla f(x^*) = 0$

While the ultimate goal is to find a global optimizer, this is a goal that is often unrealistic [39]; indeed, local solutions and saddle points will often prevent algorithms from finding true global optimizers, if they even exist. Gradient Descent will terminate when it reaches a stationary point, even though there is no guarantee that such a point is even a local optimizer. With the introduction of additional assumptions on the problem structure, however, we can observe that finding stationary points is often sufficient, or even equivalent, to finding local or global optimizers. To further this discussion, we introduce two concepts: smoothness and convexity.

Definition 2.2.1. A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is called **L-smooth** if $\forall x, y \in \mathbb{R}^n$, $||\nabla f(x) - \nabla f(y)|| \leq L||x - y||$ [12]

One can think of L-smooth functions as functions whose gradient changes no faster than the current iterate changes, up to a proportionality constant. It is straightforward to show that GD performed with a constant step-size inversely proportional to the smoothness parameter L is guaranteed to make progress towards a stationary point on each iteration, and thus will eventually converge to a stationary point [12].

In order to strengthen this result, convexity must be incorporated:

Definition 2.2.2. A set S is called **convex** if, $\forall x, y \in S$ and $\forall \lambda \in [0, 1]$, we have $\lambda x + (1 - \lambda)y \in S$. [53]

We introduce the similar notion of convex functions, which can be thought of as all functions whose epigraph (i.e. the set of all points above the graph) is itself a convex set [11]:

Definition 2.2.3. A continuously differentiable function f is called **convex** if, $\forall x, y \in \mathbb{R}^n$, we have [53]:

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle. \tag{2.21}$$

The inequality in (2.21) is referred to as the subgradient inequality. One further strengthening gives the definition of strong convexity:

Definition 2.2.4. A continuously differentiable function f is called **strongly-convex with modulus l**, or **l-strongly convex**, if, $\forall x, y \in \mathbb{R}^n$, we have [53]:

$$f(y) \geq f(x) + \langle \nabla(x), y - x \rangle + \frac{l}{2}||x - y||^2. \tag{2.22}$$

We observe that this above definition is a strengthening of the subgradient inequality from (2.21).

Convex functions are ubiquitous within optimization, and are of significant use. A useful property of convex functions is that all local optimizers of a convex function $f(x)$ are also global optimizers of $f(x)$ [11]. From this, it is clear that when dealing with convex functions as opposed to general functions, the goal of finding a global optimizer is significantly more tenable.

Armed with the tools of smoothness and convexity, a fundamental result for Gradient Descent can be stated [12]:

Theorem 3. *Let f be a L-smooth convex function on $\mathbb{R}^n$. Then Gradient Descent (as defined in 2.27) with $\alpha = \frac{1}{L}$ satisfies:*

$$f(x_k) - f(x^*) \leq \frac{2L||x_1 - x^*||^2}{k-1},$$

where k represents the number of iterations.

While this result ensures the convergence of Gradient Descent at a guaranteed rate, the rate is in fact quite slow. Indeed, in order to make an improvement of one significant figure of accuracy, one must perform a number of calculations on the same order as all of the computations already completed (each additional digit of accuracy requires an order of magnitude more computations [53]). This is referred to as sublinear convergence, and is a poor convergence rate for most optimization applications.

A more desirable convergence rate is a linear convergence rate, defined as follows [53]:

Definition 2.2.5. An algorithm is said to converge at a **linear rate** if the distance between the current iterate and the optimizer is proportional to an exponential function of the iteration counter, that is:

$$||x_k - x^*|| \leq k_0 c^k \quad \text{where } c \in (0, 1). \tag{2.23}$$

In order to achieve linear convergence for Gradient Descent, one must make use of the concept of strong convexity introduced above. We have the following result [16]:

Theorem 4. *Let f be a L-smooth, l-strongly convex function within a local ball $\mathbb{B}_r(x_0)$. Then Gradient Descent (as defined in 2.12) with $\alpha = \frac{1}{L}$ satisfies:*

$$||x_k - x^*|| \leq \left(1 - \frac{l}{L}\right)^k ||x_0 - x^*||,$$

where k represents the number of iterations.

Despite the convergence guarantees outlined above, Gradient Descent faces many limitations, particularly in the Big-Data regime outlined above. As the number of data points increases, the effort required to calculate the gradient of the objective function increases [10]. With so many samples available, performing a full calculation of the objective function gradient can be viewed as a highly-inefficient method for utilizing the available data [10]. For these reasons, as well as numerous others, a modified approach is preferred in most modern applications.

2.2.2 Stochastic Gradient Descent

Stochastic Gradient Descent (abbreviated as SGD herein) was first introduced as a Markov-chain method by Robbins and Munro in the early 1950s [63] as a method to solve iterative equations with introduced noise. Since its introduction, it has been adapted to be used as a Gradient Descent variation where the introduced noise is instead an estimate of the true gradient. SGD is designed to use partial information from the true gradient of the objective function to made a series of random iterations that, in expectation, should converge towards the optimal solution.

To formally define SGD, we introduce the general model of expected and/or empirical loss. In the expected loss case, we assume that the objective function we aim to minimize is a loss function of the form:

$$f(x) = \mathbb{E}[l(x, \xi)] \quad \text{(Expected Loss)}, \tag{2.24}$$

where ξ is a random variable, with a potentially unknown distribution, representing the data. Here, the optimal point x^* will be the point that minimizes the expected loss [12]. It is assumed that, at each iterate, the algorithm has access to the partial gradient with respect to the current iterate, $\nabla_x l(x_k, \xi)$. Using the empirical loss, we can formulate the general SGD step as follows:

$$x_{k+1} = x_k - \alpha_k \nabla_x l(x_k, \xi). \tag{2.25}$$

In order to minimize the expected loss, one must have access to the probability distribution that underlies the data; however, this is often an unrealistic assumption [10]. When this is the case, optimizers will often instead focus on the expected loss [12]:

$$\begin{aligned} f(x) &= \frac{1}{n} \sum_{i=1}^{n} f_i(x) \quad \text{(Empirical Loss)} \\ \text{where} \quad f_i(x) &= f(x, \xi_i). \end{aligned} \tag{2.26}$$

In this formulation, each ξ_i is a specific realization of the random variable ξ, i.e. a random selection of distinct data points. f_i represents the loss with respect to this random selection.

The expected loss formulation will be most appropriate for the SGD applications we will discuss later. Using the expected loss, we can formulate the general SGD step as follows:

$$x_{k+1} = x_k - \alpha_k \nabla f_{i_k}(x_k), \tag{2.27}$$

where the index i_k corresponds to the randomly selected realization, ξ_{i_k} is the corresponding data, and $\nabla f_{i_k}(x_k)$ represents the gradient of the sample loss function corresponding to the random selection.

The Expected and Empirical loss regimes introduced above are fundamentally different. If we have a finite number of samples m and wish to minimize the expected loss (2.24), we are limited to only m calculations before we should stop. This is because, if we use the same sample twice, we lose any guarantee that we are taking an unbiased sample [12][1]. In the Empirical loss regime (2.26), we have no restrictions, and can iterate over the m examples as many times as required.

We observe that, if the step-size remains constant on each iteration, we have no guarantees that the optimal point will be a stationary point for this algorithm; in fact, it is highly unlikely in general that each of the gradient estimates will all be zero precisely when the full objective function gradient is zero. Thus, for the purposes of developing convergence results, SGD is often analyzed with a decreasing step-size to improve performance as the algorithm approaches the optimizer [10] [63].

As outlined above, SGD is often preferred to full Gradient Descent as it avoids the need to complete a large gradient calculation at each step. However, this trade-off comes at the expense of weakening some of the performance guarantees of Gradient Descent. As a specific example, we have the following [12]:

Theorem 5. *Let f be a γ-strongly convex function, and assume that the stochastic gradient has a bounded second moment, i.e. $\mathbb{E}[||\nabla f_{i_k}(x_k)||^2] \leq B^2$. Then SGD (as defined in (2.27) with $\alpha_k = \frac{2}{l(k+1)}$ satisfies:*

$$\mathbb{E}\left[f\left(\sum_{j=1}^{k} \frac{2j}{k(k+1)} x_j\right)\right] - f(x^*) \leq \frac{2B^2}{\gamma(k+1)}.$$

[1] There exists some more recent work on performance gains found by conducting multiple passes of the data, see [50] as an example, but these are not discussed further here

We observe that, in the strongly-convex and smooth case that produced linear-convergence for Gradient Descent, we can only guarantee sublinear convergence for SGD. In the general case, in order to obtain a linear convergence rate for a stochastic gradient-based algorithm, we need to incorporate additional information. Approaches such as stochastic variance-reduction descent (SVRG) [40] can achieve linear convergence, but require a calculation of the full gradient at regular intervals [10]. An additional class of algorithms aims to improve convergence rates by incorporating second-order derivative information, similar to Newtons method, or by generating an approximate second-order update step using only gradient information [10]. These classes of algorithm are not detailed further here.

2.3 Machine Learning Preliminaries

2.3.1 Learning Through Training and Testing

In statistical machine learning, we aim to develop a prediction model that can correctly predict some property for an unobserved piece of data with a high degree of accuracy. A standard approach is called supervised machine learning, where we are provided with a dataset that is "labelled", meaning we already know the value of the desired property [65]. This is opposed to the process of unsupervised machine learning, where our dataset does not have labels; this is discussed further in 2.3.3.

In supervised machine learning, we use the labelled data to help develop our prediction model through a process known as "training". As a specific example, we will define a training model called Empirical Risk Minimization (ERM). In ERM learning, we make the assumption that a prediction function that performs best on the data we have been provided will also perform well when applied to unobserved data [70]. Assume we are given a dataset of the form $\{(x_1, y_1), ..., (x_n, y_n)\}$, where $\forall i$, x_i represents some important features of the data, and y_i is a label that indicates which class the data belongs in [10]. This can be a binary label (e.g. 1 if the data point belongs in a class and 0 if it does not), or can be generalized to a more complicated labelling scheme. Furthermore, we define a prediction function h that takes data points and maps them to labels. In ERM, we seek a prediction function that minimizes the so-called empirical risk:

$$R_n(h) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}[h(x_i) \neq y_i], \tag{2.28}$$

where $\mathbb{I}$ is an indicator function that outputs 1 if the expression is true, and 0 otherwise [10]. This empirical risk is similar to the empirical loss defined in 2.26

A problem with this regime should be immediately obvious: by minimizing $R_n(h)$ when our goal is really to minimize the error made by h on the unobserved data, we can potentially make a very poor choice for h using an ERM learning scheme. As an example, imagine a function h that outputs y_i for any input x_i that is present in the provided data, and outputs the label 0 otherwise [10]. This "memorization" predictor will have an empirical risk of 0, so is an ideal ERM learner [65]. However, it will perform very poorly on data that it has not observed before; selecting this h as our prediction function completely defeats the point of machine learning in the first place. This is the phenomena known as overfitting, where a predictor performs very well on the data used to train the model, but generalizes to new data very poorly.

To counteract this, we will often require that h be selected from a predetermined class of functions $\mathcal{H}$ (as opposed to all possible function that map from the space of data points to the space of real numbers) [65]. For example, we may require that h be drawn from the set of linear halfspaces, i.e. functions that separate the data space into two distinct sets. By making a judicious choice for this class of potential prediction functions, we can improve the likelihood that an ERM learner will perform well on unobserved data. We note, however, that there is a trade-off here: by choosing a smaller $\mathcal{H}$, we will minimize overfitting but will likely have a larger empirical error. If instead we choose a larger or "richer" $\mathcal{H}$, we will minimize the empirical risk but are more likely to overfit. This trade-off can be formalized by defining the so-called "VC Dimension" for a prediction class $\mathcal{H}$, but we do not discuss this further here (see [10] for a further discussion).

In light of the above, when solving machine learning problems, we will often measure performance on both the labelled data available to the algorithm (the "training set"), and a new set of data that was not used to fit the model (the "test set"). In practice, this is often done by holding back a portion of the labelled data to be used as a test set after the training phase is complete. The performance on the training data helps us understand how well the prediction model fits the data it was provided with, and the performance on the testing data helps us understand how well the model can predict new data. Understanding algorithm performance on both the training set and the testing set is of significant interest to the optimization community.

2.3.2 Machine Learning and SGD

The key process in machine learning is the selection of the prediction function; this is done by solving an optimization problem. As outlined above, a common approach is to minimize the loss over the training set using some optimization algorithm before applying to selected prediction function to the test set. While many optimization algorithms are possible, Gradient Descent, and specifically SGD, are among the most popular methods [37].

The fact that the machine learning process involves solving a specific problem (minimizing loss over the training set) when the ultimate goal is to solve a different problem (minimize losses over the test set) leads to a nearly unavoidable optimality gap between the optimal classifier and the predictor chosen by the learning procedure. This is referred to in the literature as the generalization error, and improving understanding of this error is of significant interest to the optimization and machine learning communities. We can define the generalization error $\mathcal{E}$ as follows:

$$\mathcal{E} := \mathbb{E}[R(\tilde{h}_n) - R(h^*)], \tag{2.29}$$

where:

- R is the expected risk (i.e. the loss over all possible data points, including the testing set)
- $\tilde{h}_n$ is the prediction function chosen during the machine learning procedure
- h^* is the true minimizer of R

Bottou and Bousquet [9] deconstruct this error into three distinct terms to help illustrate where the error accumulates from:

$$\begin{aligned} \mathcal{E} &= \mathbb{E}[R(h^*_H) - R(h^*)] + \mathbb{E}[R(h_n) - R(h^*_H)] + \mathbb{E}[R(\tilde{h}_n) - R(h_n)] \\ &= \mathcal{E}_{approx} + \mathcal{E}_{estim} + \mathcal{E}_{optimization}, \end{aligned} \tag{2.30}$$

where:

- h^*_H is the best possible predictor from our class of possible predictors to choose from, H

- h_n is the predictor that minimizes the empirical loss R_n (see 2.27)

The first error, $\mathcal{E}_{approx}$, is due to the fact that the true optimal classifier may not belong to our class H of possible predictors; this is the error incurred when we shrink the class to avoid overfitting. The second error, $\mathcal{E}_{estim}$, measures the error incurred because we are optimizing the empirical loss R_n instead of the true loss R. The final error, $\mathcal{E}_{optimization}$, is incurred if we do not find a zero-loss solution over the training set, and instead settle for an approximate solution.

Of these errors, the first two represent the overfitting trade-off discussed in 2.3.1. The last error, the optimization error, is puzzling at first; why would we settle for an approximate solution to the empirical loss problem? The answer relates back to the estimation error; that is, the fact that we have no guarantee that the optimizer over the training set will perform particularly well on the testing set. In practice, it may be more desirable from a computational perspective to only perform optimization on the training set to within a set tolerance, say 5%, and then use that predictor to begin testing. This is particularly of use if the optimization algorithm can converge to within the tolerance in a small number of iterations, but finding the true optimizer requires significantly more iterations.

Bottou and Bousquet studied the performance of various descent algorithms as measured by this error decomposition, and they found that, despite its somewhat inferior performance over the training set, SGD outperforms normal Gradient Descent from a generalization standpoint [9]. This phenomenon has been observed in practice [9] and studied in greater detail (see [37]), but remains a highly active research area.

2.3.3 Using Nonnegative Matrix Factorization for Learning

Before discussing how the matrix factorization problems from 2.1.2 are viewed from a machine learning standpoint, we introduce the concept of unsupervised machine learning. While supervised machine learning requires labelled data that indicates how the data should ideally be classified, unsupervised learning does not require labels; instead, unsupervised machine learning seeks to uncover hidden relationships and similarities within the data.

A canonical example of unsupervised machine learning is clustering, where the goal is to divide the data into distinct groupings with similar internal structure. A specific clustering algorithm, known as k-means clustering, achieves this by identifying the centroids of

k distinct clusters, and minimizing the overall distance between each point and the centre of its assigned cluster [65]. k-means is most commonly implemented using an alternating algorithm known as Lloyd's Algorithm [51]: each data point is assigned to a cluster by minimizing the overall Euclidean distance between each cluster point and the centroid, and then the new centroid of each cluster is calculated. This process is repeated until the cluster assignments remain constant for two consecutive iterations.

k-means clustering is an unsupervised method because the data points do not have a "correct" cluster that they should be assigned to that is known before the algorithm begins. As the algorithm progresses, the clustering process will determine what the clusters should be, and which data points should belong to each cluster. As a result, there is no training phase and testing phase in unsupervised machine learning; instead, there is one learning phase over all of the available data. While k-means clustering is a very popular method, there exist many other methods for clustering that identify similarities between data points using different metrics. For example, minimum volume ellipsoid clustering aims to minimize the overall size of the ellipsoids that cover each of the identified clusters [66]. As this method can be formulated as a semidefinite programming problem, there exist guarantees that the solution found is indeed an optimal solution.

The matrix factorization problems of 2.1.2 have many connections with unsupervised machine learning problems. As a specific example, it can be shown that a modified version of the nonnegative matrix factorization problem, with an orthogonality constraint, is equivalent to the k-means clustering problem introduced above [20]. Additional matrix factorization problems, such as standard low-rank matrix factorization and weighted low-rank matrix factorization, can all be related to k-means or other standard clustering algorithms, see [48] for further examples.

Arora et al. [2] discussed the concept of using NMF for topic modelling; this is a formalization of the document classification example discussed in 2.1.2. In topic modelling, algorithms seek to simultaneously identify latent variables ("topics") within the data, and classify each data point as a combination of those topics. Topic modelling had previously been done using the SVD to solve the low-rank matrix factorization problem, but required the strong assumptions that each document was only associated with one topic [58]. By using NMF to solve this problem, this assumption could be relaxed to the "separability assumption" (introduced in 2.1.2). Arora et al. introduce a polynomial time algorithm to recover the topics matrix W.

The work of Arora et al. is an example of machine learning through a generative model, where we assume that the provided data is generated from an unknown distribution with unknown parameters [54]. In such a model, we seek to learn the unknown parameters, thus giving us a complete understanding of the data model. Once these parameters are known, Bayes's Rule can be used to predict the labels for unobserved data in an optimal way [65]. In addition, knowledge of the distribution parameters allows us to generate new data that should maintain the structure of the existing data. Up until this point, we have described machine learning through the perspective of a discriminative model, where we only care about the classification of the data, not the underlying distribution. While the generative model sounds like an unnecessary complication, especially when performance on the test set is the main goal, it can offer more insights into the structure of the problem at hand.

To the best of our knowledge, Arora et al. were the first to use NMF as a tool in the generative framework to both perform topic classification and attempt to learn the parameters for the topic distribution. Anandkumar et al. [1] built upon this work by developing a topic modelling procedure that does not require the separability assumption of Arora et al. and simplifies the procedure for determining the distribution parameters. Their approach assumes a model known as Latent Dirichlet Allocation, which proposes that the unknown distribution is best described by a Dirichlet distribution (see below for more detail). While the Anandkumar et al. approach is more general than just matrix factorization, it demonstrates how matrix factorization, and specifically NMF, can be used to learn model parameters with minimal assumptions about the data.

Latent Dirichlet Allocation was introduced by Blei et al. [7] as an example of a larger class of generative models known as probabilistic topic models. In a probabilistic topic model, a given document is assumed to be a mixture of given topics, with specific statistical assumptions that indicate how a document can be constructed by sampling these topics [68]. Specifically, for a given word w_j, we assume that:

$$P(w_j) = \sum_{i=1}^{T} P(w_i|v_i = j)P(v_i = j), \tag{2.31}$$

where $P(w_j)$ indicates the distribution of words in a document, $P(w_j|v_j = v)$ indicates the probability of a word given the chosen topic, and $P(v_i = j)$ indicates the probability that the topic was chosen. In LDA, a Dirichlet prior is assumed for the distribution $P(w|z = j)$, with a preselected set of input parameters $\beta_1, ..., \beta_r$. In general, lower values of β_i cause the topics to be more distinct from each other, while higher values of β_i lead to more mixing of topics.

While both the Arora et al. and Anandkumar et al. works detail the usefulness of NMF in the generative model framework, neither makes specific mention of algorithm performance over an unobserved test set. In particular, given how machine learning algorithms are ultimately evaluated on their performance over unobserved data, it is of interest how a model such as NMF would perform if the latent topics found using an approach such as the two above were then applied to unobserved data. This is the subject of the numerical experiments conducted in Chapter 5

In the existing literature, research is generally focused on the performance of NMF on training data, and very little work has been done to quantify the performance of NMF on unobserved test data. To the best of our knowledge, there exist no theoretical guarantees for the performance of NMF as a machine learning algorithm when generalizing to unobserved test data.

Erichson et al. [26] studied the performance of a specific randomized NMF algorithm by using the uncovered latent variables to classify unobserved test data. Their numerical results suggest that the variables uncovered by NMF perform similarly for classification as the variables uncovered by factoring the data matrix using the SVD. For classification, Erichson et al. use the k-nearest-neighbours algorithm [65], although it is unclear exactly how this is implemented. One approach could be to compare all test points to all training points to find nearest neighbours, which is an expensive process for large data sets. A potential simplification would be to use only a random sample of training points for classification; this could lead to decreased performance.

A final example we discuss briefly is online NMF [35], where a new sample is received at each iteration, and the solution matrices are updated after projecting the new sample onto the current basis. This approach is particularly applicable for very large datasets, where new samples are readily available and can be provided to the algorithm on an ongoing basis. This approach can be thought of in a similar vein to supervised machine learning, although there is no specific training and testing phase. This approach is not explored any further in this thesis.

Chapter 3

Problem Formulation

3.1 Defining the Problem

In this thesis, we explore the problem of solving the nonnegative matrix factorization problem using a stochastic gradient descent approach. We define our main problem below:

$$\begin{aligned} &\min_{W,H} \frac{1}{2}||V - WH||_F^2 \\ &s.t.\ W \geq 0,\ \ H \geq 0, \end{aligned} \tag{3.1}$$

where $V \in \mathbb{R}^{m \times n}$, $W \in \mathbb{R}^{m \times r}$, $H \in \mathbb{R}^{r \times n}$. More specifically, we have:

- V is a data matrix with n data points, represented with m features
- W represents the m features over r latent variables
- H represents the n data points as combinations of the r latent variables
- r is a design parameter selected in advance.

We note that by adding a factor of $\frac{1}{2}$ to the objective function and squaring the norm, we will not change the optimal solutions for (3.1). This adjustment is made to make finding gradients of this function easier.

While analysis of the nonnegative version (3.1) is the ultimate goal, the results that we prove fall short of incorporating the non-negative projections required to solve the NMF problem. Instead, our results focus on the relaxed version of (3.1) without the nonnegativity constraints, which we state below:

$$\min_{W,H} \frac{1}{2}||V - WH||_F^2, \tag{3.2}$$

where V, W, and H are defined as above.

To develop a formulation for solving (3.1) using SGD, we must answer the following questions:

- How do we determine a stochastic estimate of the gradient?
- How do we select a step-size?
- How do we ensure each iterate remains nonnegative?

We address each of these issues individually.

3.2 Developing an SGD Algorithm

3.2.1 Selecting a Stochastic Gradient Estimate

In order to select a stochastic gradient estimate at each step, we must make a random selection of the data on which to base our estimate. There are many ways to make such a selection. As a specific example, a seemingly naive choice would be to, at each iteration, randomly select an entry of the data matrix V, and update both W and H based on this selection. This would mean updating the row of W and column of H that correspond to this entry. This is the approach taken by Koren et al. in their Netflix Prize submission [43], see 1.2 for more detail.

We propose iterating over the data in a slightly different method, motivated in part by the separability assumption of 2.1.2; specifically, the separability assumption motivates a re-examining of how the data should be partitioned for random selection. Indeed, if the W matrix is updated in a row-wise fashion as suggested above, we should not expect the algorithm to recover full columns of V in the W matrix. Instead, we should consider a random selection scheme that would aim to preserve the separable nature of the data, should it exist. In addition, iterating over columns of the data fits well with most machine learning applications, where we would consider each column of the data matrix as a single item (document, image, signal, etc.). It is a natural choice to perform gradient updates based on all features for each of these items, not just a single one.

This motivates an asymmetric update rule as follows: instead of selecting entries of V at random, we select a column of V at random. The H matrix is updated in a column-wise fashion, according to the column of V selected. For the W matrix, however, we update the entire matrix on each step based on the selected data point.

We observe that we can reformulate the objective function as follows:

$$\begin{aligned}\frac{1}{2}||V - WH||_F^2 &= \frac{1}{2}\sum_j \sum_i (V(i,j) - W(i,:)H(:,j))^2 \\ &= \frac{1}{2}\sum_j (\sum_i (V(i,j) - W(i,:)H(:,j))^2) \\ &= \frac{1}{2}\sum_j ||V(:,j) - WH(:,j)||_F^2 \quad (3.3) \\ &:= \sum_j F_j(W,H). \quad (3.4)\end{aligned}$$

Using (3.2), we can determine the gradient of the objective function with respect to columns of H. We have:

$$\frac{\partial F_j}{\partial W} = -(V(:,j) - WH(:,j))H(:,j)^T, \quad (3.5)$$

$$\frac{\partial F_j}{\partial H_i} = -W^T(V(:,j) - WH(:,j)), \forall i = j; \quad \frac{\partial F_j}{\partial H_i} = 0, \forall i \neq j. \quad (3.6)$$

We observe that (3.5) is the same size as W and (3.6) is the same size as a column of H, as desired.

3.2.2 Step-Size Selection

As outlined in 2.2.1, there are many different approaches in the literature for how to select a step-size for a Gradient Descent algorithm. The most straightforward, and the method chosen for our formulation, is to use a fixed step-size. This approach has the benefit of being very straightforward to implement, and adds no additional computations to each step. In addition, since the step-size is effectively a tuning parameter for the algorithm performance, we minimize the amount of tuning required by simply selecting a fixed value.

The potential drawback of choosing a fixed step-size is that, as the algorithm approaches a stationary point, it will continue to oscillate in a neighbourhood around the stationary point instead of terminating. This is because while the gradient will vanish at the stationary point, each possible stochastic gradient estimate is not likely to vanish at the same point, so we expect the algorithm to eventually iterate away from the point.

As discussed in 2.2.1, this behaviour can be tempered by using a step-size that reduces gradually over time; we refer to a further discussion on this topic by Bottou et al. [10]. For our analysis, we avoid these complications by using a fixed step-size, and accepting that the algorithm will be affected by noise around the optimizers.

3.2.3 Maintaining Nonnegativity of Iterates

It is clear that, in general, we should not expect a gradient update to maintain nonnegativity over each iterate. A natural choice, motivated by the alternating least squares method discussed in 2.1.2, is to perform the gradient update step, and then project any negative entries of the resulting iterate onto the nonnegative orthant. This is the method employed in our formulation.

We note that, provided the step-size is small, iterate entries that do become negative will only slightly become negative, so they are well-approximated by being set to zero. If the step-size is too large, we would expect a very large number of the iterate entries to be set to zero. This can potentially be an asset, as sparse solutions to (3.1) are often desirable in real-world applications, but the sparsity can also slow down the algorithm and potentially cause it to stall at a large distance from the optimizer. This illustrates why selection and tuning of the step-size is vital to observing good algorithm performance.

We introduce one additional projection scheme that was considered, which was also motivated by the separable NMF case introduced above. In this case, we expect to recover r identity columns in the final H matrix (i.e. r columns that have all 0 entries except for a single non-zero entry of value 1). This motivates a projection scheme where instead of projecting the columns of H onto the nonnegative orthant, we project the columns of H onto the r-dimensional simplex:

Definition 3.2.1. The r-dimensional simplex is defined as [11]:

$$\{x \in \mathbb{R}^{r+1} : x_0 + \cdots + x_r = 1,\ x_i \geq 0\ \forall i \in [0:r]\}.$$

We observe that this projection scheme is particularly conducive to helping recover the identity columns of H. This projection operator can be computed efficiently using a "soft thresholding" algorithm [59]. While this approach is likely useful for data that are known to be separable, it is not considered further here.

3.2.4 An SGD Algorithm for Nonnegative Matrix Factorization

With these gradients, step-size, and projection schemes identified above, we can adapt the general SGD update rule (2.27) to solve (3.1). Specifically, we have:

Algorithm 1: Stochastic Gradient Descent for NMF

```
for $k$=1,2,... do
    Choose $j$ from $[1:n]$ uniformly at random;
    $W_{k+1} = max\{W_k + \alpha(V(:,j) - W_k H_k(:,j))H_k(:,j)^T, 0\}$;
    $H_{k+1}(:,j) = max\{H_k(:,j) + \alpha W_k^T(V(:,j) - W_k H_k(:,j)), 0\}$;
end
```

where the maximum operator above indicates an element-wise maximum, and $0 \in R^m$. Note that we do not explicitly state a termination condition here; instead, we run the algorithm for a sufficiently long number of iterations, and observe where the algorithm has settled (understanding that there will be noise).

We observe that this is "projected subgradient descent" algorithm discussed in [12]. We also observe that this is distinct from the "gradient-projection" method discussed in Chapter 16 of [55]. In this method, a line-search is conducted along the direction of steepest descent, but when a bound is reached (i.e. we leave the nonnegative orthant), the search direction is projected onto this boundary; as a result, the nonnegativity constraint is maintained.

As outlined in 3.1, our results use the same algorithm as above without the projection steps for W and H to solve the relaxed problem 3.2; in effect, this is SGD performed on the low-rank matrix factorization problem. With this relaxation, we have the following modified algorithm:

Algorithm 2: Stochastic Gradient Descent for Low-Rank Matrix Factorization

```
for $k$=1,2,... do
    Choose $j$ from $[1 : n]$ uniformly at random;
    $W_{k+1} = W_k + \alpha(V(:,j) - W_k H_k(:,j)) H_k(:,j)^T$;
    $H_{k+1}(:,j) = H_k(:,j) + \alpha W_k^T (V(:,j) - W_k H_k(:,j))$;
end
```

3.2.5 A Preliminary Approach: Gradient Descent

In order to study the performance of Algorithm 1 and Algorithm 2 on the matrix factorization problems, we first analyze the case of full Gradient Descent (i.e. instead of choosing a stochastic gradient estimate at each step, we use the full gradient information from the objective function). At first glance, this seems like a strange case to analyze, given that we can efficiently find exact solutions to the matrix factorization problem using the Singular Value Decomposition (see 2.1.1). By analysing this simplified case, we aim to observe how the algorithm performs, and use these insights to aid our analysis of how the more complicated problems (i.e. the NMF case) can be solved. Given that our main problem is non-convex, we have no guarantee that Gradient Descent will avoid local minimizers and converge. Despite these challenges, we will show that some meaningful results can still be observed.

It is straightforward to observe that, in the full gradient case, we have:

$$\nabla_W F(W, H) = -(V - WH)H^T, \tag{3.7}$$

$$\nabla_H F(W, H) = -W^T(V - WH). \tag{3.8}$$

We can now state a straightforward algorithm for solving the low-rank matrix factorization problem with Gradient Descent (note: for simplicity, we continue to use a fixed step-size α):

Algorithm 3: Gradient Descent for Low-Rank Matrix Factorization

```
for $k$=1,2,... do
    Choose $j$ from $[1:n]$ uniformly at random;
    $W_{k+1} = W_k + \alpha(V - W_k H_k)H_k^T$;
    $H_{k+1} = H_k + \alpha W_k^T(V - W_k H_k)$;
end
```

We will develop our first results based on this algorithm.

Chapter 4

Analytic Results

4.1 Gradient Descent for Low-Rank Matrix Factorization

As outlined in Chapter 3, we begin our analysis with Algorithm 3: the non-projected, non-stochastic algorithm. Here, we have two cases: the case where a zero-loss low-rank factorization is known to exist, and the general case where no exact low-rank factorization exists. We develop our first result below.

4.1.1 Gradient Descent with a Known Zero-Loss Solution

For the first case, we assume that our problem instance has a zero-loss solution $V = W^*H^*$. Our goal is to show that the update rule from Algorithm 3 causes W_kH_k to converge to V, or equivalently, drives the residual term $V - W_kH_k$ to 0.

Despite the broad literature relating to the performance of Gradient Descent, we are unaware of any existing theorems that satisfy the unique restrictions of this problem. Specifically, the matrix factorization problem is a non-convex problem without a unique minimizer; furthermore, we require a global minimizer, not just a local minimizer. These properties prevent the application of a standard Gradient Descent result here.

To aid the analysis, we express W_k and H_k in the following form:

$$W_k = W^* R_k + L_k, \tag{4.1}$$
$$H_k = S_k H^* + M_k. \tag{4.2}$$

In the above decomposition, R_k is the least-squares solution to $W_k = W^* R_k$ (i.e. $||L_k||$ is minimized) and similar for S_k. From this decomposition, we have that each column of L_k is orthogonal to each column of W^* and each row of M_k is orthogonal to each row of H_k (i.e. $W^{*T} L_k = 0$ and $M_k H^{*T} = 0$).

We observe that, provided L_k and M_k are initially small, we have that the residual $V - W_k H_k$ is well-approximated by $W^*(I - R_k S_k) H^*$. Indeed, in our analysis, we will use $W^*(I - R_k S_k) H^*$ as a proxy for our target residual. By initializing the algorithm with $||L_0||$, $||M_0||$, and $||I - R_0 S_0||$ sufficiently small, we can control how small the initial residual $V - W_k H_k$ is.

We have the following result:

Theorem 6. *Let $V \in \mathbb{R}^{m \times n}$, and let $W^* \in \mathbb{R}^{m \times r}$, $H^* \in \mathbb{R}^{r \times n}$ be rank-r matrices that satisfy $V = W^* H^*$. Let R_k, S_k, L_k, M_k be defined as above, and let:*

$$\begin{aligned}
r_0 &= ||W^* R_0||, \\
s_0 &= ||S_0 H^*||, \\
l_1 &= ||L_0||, \\
m_1 &= ||M_0||, \\
q_0 &= ||(R_0^T W^{*T} W^* R_0)^{-1} R_0^T W^{*T}||, \\
t_0 &= ||H^{*T} S_0^T (S_0 H^* H^{*T} S_0^T)^{-1}, ||
\end{aligned}$$

where we assume for simplicity that $m_1 \geq l_1$[1]. Let m_1, l_1, and $||I - R_0 S_0||$ be sufficiently small such that p_1 can be selected to satisfy the following statements simultaneously:

[1]If $l_1 > m_1$, we can conduct an identical analysis, with the roles of q_0 and t_0 exchanged.

$$p_1 > (m_1^2 + l_1^2)(r_0+1)(s_0+1)(q_0+1)^2, \tag{a}$$
$$p_1 \geq ||W^*(I - R_0 S_0)H^*||, \tag{b}$$
$$p_1 \leq \frac{\frac{1}{(q_0+1)^2} - \frac{1}{(q_0+1)^4} - m_1(q_0+1)}{(q_0+1)^2(s_0+1)}, \tag{c}$$
$$p_1 \leq \frac{\frac{1}{(t_0+1)^2} - \frac{1}{(t_0+1)^2(q_0+1)^2} - m_1(t_0+1)}{(t_0+1)^2(r_0+1)}, \tag{d}$$
$$p_1 < \frac{1}{(s_0+1)(q_0+1)^2}, \tag{e}$$

and α is chosen such that:

$$\begin{aligned} \alpha((s_0+1)^2 + m_1^2) &< 1, \\ \alpha((r_0+1)^2 + l_1^2) &< 1, \\ \alpha &\leq \frac{1}{2}, \end{aligned}$$

Then Algorithm 3 starting from the specified initial point will produce a sequence of iterates (W_k, H_k) where the residual $V - W_k H_k$ tends 0 at a linear rate as $k \to \infty$.

Proof. We begin by deriving recursive forms of the updates for all of these terms, starting with L_k. We have $L_k = (I - W^*(W^{*T}W^*)^{-1}W^{*T})W_k$, and thus:

$$\begin{aligned} L_{k+1} &= (I - W^*(W^{*T}W^*)^{-1}W^{*T})(W_k + \alpha(V - W_k H_k)H_k^T) \\ &= L_k + \alpha(I - W^*(W^{*T}W^*)^{-1}W^{*T})(V - W_k H_k)H_k^T \\ &= L_k + \alpha(I - W^*(W^{*T}W^*)^{-1}W^{*T})W^*H^*H_k^T - \alpha(I - W^*(W^{*T}W^*)^{-1}W^{*T})W_k H_k H_k^T \\ &= L_k - \alpha L_k H_k H_k^T \\ &= L_k(I - \alpha H_k H_k^T), \end{aligned}$$

where the fourth line follows from recognizing that the middle term is exactly equal to zero after cancellation, and a factor of L_k appears in the final term. Thus, we have:

$$L_{k+1} = L_k(I - \alpha H_k H_k^T). \tag{4.3}$$

We observe that, due to the symmetry between the forms of W and H, the calculations that we do for W can be adapted to show the same result for H; this will be used to simplify the proof. An analogous calculation to the one for L_{k+1} can be used to show:

$$M_{k+1} = (I - \alpha W_k^T W_k) M_k. \tag{4.4}$$

From the above equations, we can also derive recursive forms for R_{k+1} and S_{k+1}, or more specifically $W^* R_{k+1}$ and $S_{k+1} H^*$. We have

$$\begin{aligned}
W^* R_{k+1} &= W_{k+1} - L_{k+1} \\
&= W_k + \alpha(V - W_k H_k) H_k^T - L_k(I - \alpha H_k H_k^T) \\
&= W_k(I - \alpha H_k H_k^T) + \alpha V H_k^T - L_k(I - \alpha H_k H_k^T) \\
&= W^* R_k (I - \alpha H_k H_k^T) + \alpha V H_k^T \\
&= W^* R_k (I - \alpha(S_k H^* + M_k)(H^{*T} S_k^T + M_k^T)) + \alpha W^* H^* (H^{*T} S_k^T + M_k^T) \\
&= W^* R_k (I - \alpha S_k H^* H^{*T} S_k^T - \alpha M_k M_k^T) + \alpha W^* H^* H^{*T} S_k^T \quad (\text{since } H^* M_k^T = 0) \\
&= W^* R_k (I - \alpha M_k M_k^T) + \alpha W^* (I - R_k S_k) H^* H^{*T} S_k^T.
\end{aligned}$$

A similar calculation for the $S_{k+1} H^*$ term gives the equations:

$$W^* R_{k+1} = W^* R_k (I - \alpha M_k M_k^T) + \alpha W^* (I - R_k S_k) H^* H^{*T} S_k^T, \tag{4.5}$$

$$S_{k+1} H^* = (I - \alpha L_k^T L_k) S_k H^* + \alpha R_k^T W^{*T} W^* (I - R_k S_k) H^*. \tag{4.6}$$

As outlined above, we can use $W^*(I - R_k S_k) H^*$ as a proxy for the residual $V - W_k H_k$. We can use (4.5) and (4.6) to develop the following recursion for $W^*(I - R_k S_k) H^*$. Dropping all $O(\alpha^2)$ terms[2], we obtain:

$$\begin{aligned}
W^*(I - R_{k+1} S_{k+1}) H^* &= W^* H^* - W^* R_{k+1} S_{k+1} H^* \\
&= (I - \alpha W^* R_k R_k^T W^{*T})(W^*(I - R_k S_k) H^*)(I - \alpha H^{*T} S_k^T S_k H^*) \\
&\quad + \alpha W^* R_k (M_k M_k^T + L_k^T L_k) S_k H^*.
\end{aligned} \tag{4.7}$$

[2]In this analysis, we ignore all terms that are $O(\alpha^2)$ or of higher degree; the analysis could be repeated to include these terms with a slight strengthening of the constraints presented.

We aim to prove bounds which show that (4.3), (4.4), and (4.7) converge to zero geometrically, while proving upper and lower bounds on the singular values for W^*R_k and S_kH^* using (4.5) and (4.6). That is, we aim to find constants that satisfy the following inequalities simultaneously, which we will show through induction on k

$$||L_k|| \leq l_1 l_2^k \quad \text{where } l_2 < 1, \tag{4.8}$$
$$||M_k|| \leq m_1 m_2^k \quad \text{where } m_2 < 1, \tag{4.9}$$
$$||W^*R_k|| \leq r_0 + r_1 \sum_{i=1}^{k} r_2^i \quad \text{where } r_2 < 1, \tag{4.10}$$
$$||S_kH^*|| \leq s_0 + s_1 \sum_{i=1}^{k} s_2^i \quad \text{where } s_2 < 1, \tag{4.11}$$
$$||W^*(I - R_kS_k)H^*|| \leq p_1 p_2^k \quad \text{where } p_2 < 1, \tag{4.12}$$
$$\sigma_{min}(W^*R_k) \geq \frac{1}{q_0 + q_1 \sum_{i=1}^{k} q_2^i}, \tag{4.13}$$
$$\sigma_{min}(S_kH^*) \geq \frac{1}{t_0 + t_1 \sum_{i=1}^{k} t_2^i}. \tag{4.14}$$

The base of the induction for (4.8) through (4.14) follows directly from the constants in the hypothesis of the theorem. To show the induction step for each, we select the following constant values (the choice of which will be motivated later):

$$\begin{aligned}
r_2 &= s_2 = q_2 = t_2 = p_2,\\
r_1 &= \alpha(s_0 + 1)p_1,\\
s_1 &= \alpha(r_0 + 1)p_1,\\
q_1 &= \frac{\alpha}{(q_0 + 1)^2},\\
t_1 &= \frac{\alpha}{(t_0 + 1)^2},\\
l_2 &= 1 - \frac{\alpha}{(t_0 + 1)^2},\\
m_2 &= 1 - \frac{\alpha}{(q_0 + 1)^2},\\
p_2 &= 1 - \frac{2\alpha}{(q_0 + 1)^2} + \alpha\frac{(r_0 + 1)(s_0 + 1)}{p_1}(m_1^2 + l_1^2).
\end{aligned} \tag{4.15}$$

We define the maximum values of the RHS of (4.10) and (4.11) as r_∞ and s_∞ respectively. In addition, we make the following inductive assumptions on the growth of these bounds:

$$\begin{aligned} r_1 \sum_{i=1}^{k} r_2^i &< 1, \\ s_1 \sum_{i=1}^{k} s_2^i &< 1, \\ q_1 \sum_{i=1}^{k} q_2^i &< 1, \\ t_1 \sum_{i=1}^{k} t_2^i &< 1, \end{aligned} \tag{4.16}$$

We begin by using Statement (a) from the theorem to show that $p_2 < 1$. From (a), we have:

$$\begin{aligned} \frac{\alpha(m_1^2 + l_1^2)(r_0+1)(s_0+1)}{p_1} &< \frac{\alpha}{(q_0+1)^2} \\ &< \frac{2\alpha}{(q_0+1)^2}. \end{aligned}$$

Thus, we clearly have:

$$0 < \frac{2\alpha}{(q_0+1)^2} - \frac{\alpha(m_1^2 + l_1^2)(r_0+1)(s_0+1)}{p_1},$$

or

$$1 - \frac{2\alpha}{(q_0+1)^2} + \frac{\alpha(m_1^2 + l_1^2)(r_0+1)(s_0+1)}{p_1} < 1.$$

The LHS of the expression above is exactly the p_2 formulated in (4.15). Thus, we have shown that $p_2 < 1$.

Now we examine (4.10) and (4.11). We will show now that the choices of constants in (4.15), along with r_0 and s_0 defined in the Theorem, establish (4.10) and (4.11). Consider Statement (e) from the theorem. By rearranging and multiplying both sides by α, we have:

$$\begin{aligned}(e) \;\Rightarrow\; \alpha(s_0+1)p_1 &< \frac{\alpha}{(q_0+1)^2} \\ &= \frac{2\alpha}{(q_0+1)^2} - \frac{\alpha}{(q_0+1)^2}.\end{aligned}$$

We now use Statement (a) to conclude that:

$$\frac{m_1^2+l_1^2}{p_1}(r_0+1)(s_0+1) < \frac{1}{(q_0+1)^2}.$$

We can substitute this in above to obtain:

$$\begin{aligned}\alpha(s_0+1)p_1 &< \frac{2\alpha}{(q_0+1)^2} - \alpha(r_0+1)(s_0+1)\frac{m_1^2+l_1^2}{p_1} \\ &= 1-p_2 \quad \text{(from (4.15))},\end{aligned}$$

and thus:

$$\begin{aligned}\frac{\alpha(s_0+1)p_1}{1-p_2} &< 1 \\ \Rightarrow \alpha(s_0+1)p_1\sum_{i=1}^{\infty} p_2^i &< 1,\end{aligned} \tag{4.17}$$

where the last line follows from $p_2 < 1$ (and using the form for an infinite geometric sum). We observe that (4.17) validates our assumption $r_1\sum_{i=1}^{k} r_2^i < 1$ from (4.16).

We can also use the above to validate the assumption $q_1\sum_{i=1}^{k} q_2^i < 1$ from (4.16). We begin with the reformulation of Statement (a):

$$\frac{m_1^2+l_1^2}{p_1}(r_0+1)(s_0+1) < \frac{1}{(q_0+1)^2}.$$

By multiplying by a factor of α, this can be rewritten as:

$$\frac{\alpha(m_1^2 + l_1^2)(r_0 + 1)(s_0 + 1)}{p_1} < \frac{2\alpha}{(q_0 + 1)^2} - \frac{\alpha}{(q_0 + 1)^2}.$$

Rearranging, we can show:

$$\begin{aligned} \frac{\alpha}{(q_0+1)^2} &< 1 - p_2 \\ \Rightarrow \frac{\alpha}{(q_0+1)^2(1-p_2)} &< 1. \end{aligned}$$

The last statement above gives $q_1 \sum_{i=1}^{\infty} q_2^i < 1$, as required.

Now we observe that, taking norms of (4.5) and applying the induction hypotheses (4.11) and (4.12), we have that:

$$||W^* R_{k+1}|| \leq ||W^* R_k|| + \alpha p_1 p_2^k s_\infty,$$

provided that $\alpha m_1^2 \leq 1$, which is certainly implied by the stated conditions on α. More generally, we can expand this recursion to say that:

$$\begin{aligned} ||W^* R_{k+1}|| &\leq ||W^* R_0|| + \alpha p_1 s_\infty \sum_{i=1}^{k+1} p_2^i \\ &\leq ||W^* R_0|| + \alpha p_1 (s_0 + 1) \sum_{i=1}^{k+1} p_2^i \quad \text{(by (4.16))}. \end{aligned}$$

We observe that, with the choices of r_0, r_1, and r_2 specified above, (4.17) certainly implies that (4.10) holds for $k+1$.

An analogous argument can be used to show that (4.11) will hold for $k+1$ with s_0, s_1, and s_2 as defined above, while also validating the two remaining assumptions from (4.16).

We continue by analyzing (4.13) and (4.14). We observe that we can express the minimum singular value of W^*R_k as:

$$\sigma_{min}(W^*R_k) = \frac{1}{||(R_k^T W^{*T} W^* R_k)^{-1} R_k^T W^{*T}||}.$$

Thus, we need to compute an expression for $(R_{k+1}^T W^{*T} W^* R_{k+1})^{-1} R_{k+1}^T W^{*T}$. We use the identity $(X+E)^{-1} = X^{-1} - X^{-1}EX^{-1}$ for small E[3]. For simplicity, we denote $\tilde{W}_k = R_k^T W^{*T} W^* R_k$. We have:

$$\begin{aligned} R_{k+1}^T W^{*T} W^* R_{k+1} &= \tilde{W}_k - \alpha M_k M_k^T R_k^T W^{*T} W^* R_k + \alpha S_k H^* H^{*T} (I - S_k^T R_k^T) W^{*T} W^* R_k \\ &\quad - \alpha R_k^T W^{*T} W^* R_k M_k M_k^T + \alpha R_k^T W^{*T} W^* (I - R_k S_k) H^* H^{*T} S_k^T \\ &= \tilde{W}_k + E_k, \end{aligned}$$

where

$$\begin{aligned} E_k = &-\alpha M_k M_k^T R_k^T W^{*T} W^* R_k + \alpha S_k H^* H^{*T} (I - S_k^T R_k^T) W^{*T} W^* R_k \\ &- \alpha R_k^T W^{*T} W^* R_k M_k M_k^T + \alpha R_k^T W^{*T} W^* (I - R_k S_k) H^* H^{*T} S_k^T. \end{aligned}$$

Thus, we can use the inverse identity above to conclude:

$$\begin{aligned} (R_{k+1}^T W^{*T} W^* R_{k+1})^{-1} R_{k+1}^T W^* &= (\tilde{W}_k^{-1} - \tilde{W}_k^{-1} E_k \tilde{W}_k^{-1})((I - \alpha M_k M_k^T) R_k^T W^{*T} \\ &\quad + \alpha S_k H^* H^{*T} (I - S_k^T R_k^T) W^{*T}) \\ &= (I + \alpha M_k M_k^T - \alpha \tilde{W}_k^{-1} S_k H^* H^{*T} (I - S_k^T R_k^T) W^{*T} W^* R_k \\ &\quad - \alpha \tilde{W}_k^{-1} R_k^T W^{*T} W^* (I - R_k S_k) H^* H^{*T} S_k^T) \tilde{W}_k^{-1} R_k^T W^{*T} \\ &\quad + \alpha \tilde{W}_k^{-1} S_k H^* H^{*T} (I - S_k^T R_k^T) W^{*T}. \end{aligned}$$

If R_k is non-singular, which is implied by (4.13), we observe that $W^{*T} W^* R_k \tilde{W}_k^{-1} R_k^T = I$, which allows the above to simplify to:

[3]This identity holds provided that E is $O(\alpha)$, and the $O(\alpha^2)$ terms have been neglected.

$$(R_{k+1}^T W^{*T} W^* R_{k+1})^{-1} R_{k+1}^T W^* = (I + \alpha M_k M_k^T \\ - \alpha \tilde{W}^{-1} R_k^T W^{*T} W^* (I - R_k S_k) H^* H^{*T} S_k^T) \tilde{W}^{-1} R_k^T W^{*T}.$$

To simplify once more, we denote $X_k = \tilde{W}^{-1} R_k^T W^{*T} = (R_k^T W^{*T} W^* R_k)^{-1} R_k^T W^{*T}$ to conclude that:

$$X_{k+1} = X_k + \alpha M_k M_k^T X_k - \alpha X_k W^* (I - R_k S_k) H^* H^{*T} S_k^T X_k. \tag{4.18}$$

Taking norms, we obtain:

$$||X_{k+1}|| \leq ||X_k|| + \alpha ||M_k||^2 ||X_k|| + \alpha ||W^*(I - R_k S_k) H^*|| \cdot ||S_k H^*|| \cdot ||X_k||^2.$$

By using the induction hypotheses 4.9, 4.11, 4.12, and 4.16, we obtain:

$$||X_{k+1}|| \leq ||X_k|| + \alpha m_1^2 m_2^{2k} \left(q_0 + q_1 \sum_{i=1}^{k} q_2^i \right) + \alpha \left(q_0 + q_1 \sum_{i=1}^{k} q_2^i \right)^2 p_1 p_2^k \left(s_0 + s_1 \sum_{i=1}^{k} s_2^i \right). \tag{4.19}$$

An analogous argument gives the following expressions for Y_{k+1} and $||Y_{k+1}||$:

$$Y_{k+1} = Y_k + \alpha L_k^T L_k Y_k - \alpha Y_k R_k^T W^{*T} W^* (I - R_k S_k) H^* Y_k, \tag{4.20}$$

$$||Y_{k+1}|| \leq ||Y_k|| + \alpha l_1^2 l_2^{2k} \left(t_0 + t_1 \sum_{i=1}^{k} t_2^i \right) + \alpha \left(t_0 + t_1 \sum_{i=1}^{k} t_2^i \right)^2 p_1 p_2^k \left(r_0 + r_1 \sum_{i=1}^{k} r_2^i \right). \tag{4.21}$$

We address (4.19). Consider Statement (c) from the Theorem. Since $\alpha \leq \frac{1}{2}$, we can express (c) as:

$$p_1 \leq \frac{\frac{1}{(q_0+1)^2} - \frac{2\alpha}{(q_0+1)^4} - m_1(q_0+1)}{(q_0+1)^2 (s_0+1)}.$$

By rearranging and multiplying both sides by α, we obtain:

$$\begin{aligned}\alpha m_1(q_0+1)+\alpha(q_0+1)^2(s_0+1)p_1 &\leq \frac{\alpha}{(q_0+1)^2}\left(1-\frac{2\alpha}{(q_0+1)^2}\right)\\ &= q_1\left(1-\frac{2\alpha}{(q_0+1)^2}\right) \quad \text{(by (4.15))}\\ &\leq q_1\left(1-\frac{2\alpha}{(q_0+1)^2}+\frac{\alpha(r_0+1)(s_0+1)(m_1^2+l_1^2)}{p_1}\right)\\ &= q_1p_2.\end{aligned}$$

Since $m_2^2 \leq p_2$, the above implies:

$$\alpha m_1\frac{m_2^{2k}}{p_2^k}(q_0+1)+\alpha(q_0+1)^2p_1(s_0+1)\leq q_1p_2,$$

and multiplying by p_2^k, we obtain:

$$\begin{aligned}\alpha m_1 m_2^{2k}(q_0+1)+\alpha(q_0+1)^2p_1p_2^k(s_0+1) &\leq q_1p_2^{k+1}\\ &= q_1q_2^{k+1}.\end{aligned}$$

Thus, by applying our induction statements from (4.16) and substituting into (4.19), we have shown that:

$$||X_{k+1}|| \leq ||X_k|| + q_1q_2^{k+1}.$$

By induction, we obtain:

$$||X_{k+1}|| \leq q_0 + q_1\sum_{i=1}^{k+1} q_2^i,$$

and therefore

$$\begin{aligned}\frac{1}{q_0 + q_1 \sum_{i=1}^{k+1} q_2^i} &\leq \frac{1}{||X_{k+1}||} \\ &= \sigma_{min}(W^* R_{k+1}).\end{aligned}$$

This verifies (4.13). An analogous argument can be used to use Theorem Statement (d) and (4.21) to show:

$$t_0 + t_1 \sum_{i=1}^{k+1} t_2^i \geq ||Y_{k+1}||,$$

and therefore

$$\begin{aligned}\frac{1}{t_0 + t_1 \sum_{i=1}^{k+1} t_2^i} &\leq \frac{1}{||Y_{k+1}||} \\ &= \sigma_{min}(S_{k+1} H^*),\end{aligned}$$

which verifies (4.14).

Now we consider (4.8) and (4.9). By taking norms of (4.3), we have:

$$\begin{aligned}||L_{k+1}|| &\leq ||L_k|| \cdot ||I - \alpha H_k H_k^T|| \\ &= ||L_k|| \cdot ||(I - \alpha S_k H^* H^{*T} S_k^T - \alpha M_k M_k^T)|| \\ &\leq ||L_k||(1 - \alpha \lambda_{min}(S_k H^* H^{*T} S_k^T)) \\ &\leq ||L_k|| \left(1 - \frac{\alpha}{(t_0 + t_1 \sum_{i=1}^{\infty} t_2^i)^2}\right) \\ &\leq ||L_k|| \left(1 - \frac{\alpha}{(t_0 + 1)^2}\right) \text{ (by (4.16))} \\ &\leq l_1 l_2^k \left(1 - \frac{\alpha}{(t_0 + 1)^2}\right) \text{ (by induction)} \\ &= l_1 l_2^{k+1}.\end{aligned}$$

This completes the induction proof for (4.8). An analogous argument shows that we also have:

$$\begin{aligned}||M_{k+1}|| &\leq ||M_k|| \left(1 - \frac{\alpha}{(q_0+1)^2}\right)^k \\ &= m_1 m_2^{k+1}.\end{aligned}$$

This completes the induction proof for (4.9).

We conclude our analysis by establishing (4.12). Taking norms of (4.7), we obtain:

$$\begin{aligned}||W^*(I - R_{k+1}S_{k+1})H^*|| \leq &\left(1 - \frac{\alpha}{(t_0 + t_1 \sum_{i=1}^{\infty} t_2^i)^2}\right)\left(1 - \frac{\alpha}{(q_0 + q_1 \sum_{i=1}^{\infty} q_2^i)^2}\right) ||W^*(I - R_k S_k)H^*|| \\ &+ \alpha r_\infty s_\infty (m_1^2 + l_1^2) m_2^{2k}.\end{aligned}$$

Since we assume $m_2 > l_2$, we can simplify to:

$$||W^*(I - R_{k+1}S_{k+1})H^*|| \leq m_2^2 ||W^*(I - R_k S_k)H^*|| + \alpha(r_0+1)(s_0+1)(m_1^2 + l_1^2) m_2^{2k}. \tag{4.22}$$

Recall our definition of p_2 from (13). Neglecting $O(\alpha^2)$ terms, we can complete the square for the definition of p_2, and obtain:

$$\begin{aligned}p_2 &= (1 - \frac{\alpha}{(q_0+1)^2})^2 + \frac{\alpha(m_1^2 + l_1^2)(r_0+1)(s_0+1)}{p_1} \\ &= m_2^2 + \frac{\alpha(m_1^2 + l_1^2)(r_0+1)(s_0+1)}{p_1} \\ &\geq m_2^2 + \frac{\alpha(m_1^2 + l_1^2)(r_0+1)(s_0+1)}{p_1} \frac{m_2^{2k}}{p_2^k},\end{aligned}$$

where the last line follows given $p_2 > m_2^2$, which is certainly implied by the second line above. Multiplying each term through by $p_1 p_2^k$, we obtain:

$$\begin{aligned} p_1 p_2^{k+1} &\geq m_2^2 p_1 p_2^k + \alpha(m_1^2 + l_1^2)(r_0+1)(s_0+1)m_2^{2k} \\ &\geq ||W^*(I - R_{k+1}S_{k+1})H^*|| \ \text{ (from (17))}. \end{aligned}$$

This final statement shows that, by induction, we will always have that (4.7) holds. As a final condition, in order for the base case for the induction to be true, we must enforce that $p_1 \geq ||W^*(I - R_k S_k)H^*||$, which is given in Statement b) of the Theorem. This completes the proof for (4.7), thus completing the proof of the theorem. □

Discussion of Theorem 6

As outlined in the proof, since we have that L_k, M_k, and $W^*(I - R_k S_k)H^*$ all converge to zero, we can guarantee the algorithm produces a sequence with limit points that satisfy $W_k H_k = W^* H^* = V$. Thus, we have that $V - W_k H_k$ will converge to zero; furthermore, as this residual is a multiplicative factor in both terms of the objective function gradient, we observe that we obtain a zero-gradient limit point.

The proof for Theorem 6 demonstrates that this convergence will occur at a linear rate; however, since the parameters l_2, m_2, and p_2 are often quite close to 1, this convergence can still be "slow". In addition, in order to satisfy the conditions of the theorem, Algorithm 3 must be initialized quite close to the the true solution (W^*, H^*). While this constraint is quite restrictive, in practice the algorithm will converge at this linear rate from any randomly initialized point (see 5.1.1 for a numerical example of this).

A final observation we make is that, since L_k and M_k tend towards zero, the iterates (W_k, H_k) tend towards the span of the zero-loss solution (W^*, H^*). This is a nice property for the iterates to have, and we will show that, even in the case where no zero-loss solution exists, this property will in fact continue to hold.

4.1.2 Gradient Descent: Extending to Non Zero-Loss Solution Case

To extend the above result, we weaken the assumption that a zero-loss solution exists (i.e. we remove the assumption $V = W^* H^*$). Instead, we assume that our problem instance is "close" to a zero-loss solution, i.e. $V = W^* H^* + \Delta$, where Δ represents a small error term. We note that, by changing the matrix V and the target low-rank r, we have some measure of control over the magnitude $\delta := ||\Delta||$. We assume that W^* and H^* represent the best-possible rank-r approximation of V, found through the SVD (see 2.1.1). This implies that we must have $W^{*T}\Delta = 0$ and $\Delta H^{*T} = 0$.

We state and prove the following result:

Theorem 7. *Let all of the assumptions of Theorem 6 hold, with the exception that $V = W^*H^* + \Delta$, where W^* and H^* represent the best rank-r approximation of V. In addition to the Theorem 6 assumptions, let δ be sufficiently small such that:*

$$\delta < \frac{1}{(t_0+1)^2}. \tag{f}$$

Then Algorithm 3 starting from the specified initial point will generate a sequence of iterates (W_k, H_k) where W_kH_k tends to W^*H^* as $k \to \infty$.

Proof. We claim that the same induction strategy that was used to establish Theorem 6 can be used to prove Theorem 7, with minor modifications. In Theorem 6, there were 7 different quantities that needed to be controlled. We list them below, and the corresponding recursive form that outlined how each quantity is updated:

- L_{k+1} (4.3)
- M_{k+1} (4.4)
- W^*R_{k+1} (4.5)
- $S_{k+1}H^*$ (4.6)
- $W^*(I - R_{k+1}S_{k+1})H^*$ (4.7)
- $(R_{k+1}^TW^{*T}W^*R_{k+1})^{-1}R_{k+1}^TW^{*T}$ (4.18)
- $H^{*T}S_{k+1}^T(S_{k+1}H^*H^{*T}S_{k+1}^T)^{-1}$ (4.20)

We claim that, of the above recursions, only the rules for L_{k+1} and M_{k+1} change with the introduction of a Δ.

We begin with the L_{k+1} and (4.3). We have:

$$\begin{aligned}
L_{k+1} &= (I - W^*(W^{*T}W^*)^{-1}W^{*T})(W_k + \alpha(V - W_kH_k)H_k^T) \\
&= L_k + \alpha(I - W^*(W^{*T}W^*)^{-1}W^{*T})(V - W_kH_k)H_k^T \\
&= L_k + \alpha(I - W^*(W^{*T}W^*)^{-1}W^{*T})(W^*H^* + \Delta - W_kH_k)H_k^T \\
&= L_k + \alpha(I - W^*(W^{*T}W^*)^{-1}W^{*T})W^*H^*H_k^T + \alpha(I - W^*(W^{*T}W^*)^{-1}W^{*T})\Delta H_k^T \\
&\quad - \alpha(I - W^*(W^{*T}W^*)^{-1}W^{*T})W_kH_kH_k^T \\
&= L_k + \alpha\Delta H_k^T - \alpha(I - W^*(W^{*T}W^*)^{-1}W^{*T})W_kH_kH_k^T \\
&= L_k(I - \alpha H_kH_k^T) + \alpha\Delta(H^{*T}S_k^T + M_k^T) \\
&= L_k(I - \alpha H_kH_k^T) + \alpha\Delta M_k^T.
\end{aligned}$$

Thus, we have:

$$L_{k+1} = L_k(I - \alpha H_kH_k^T) + \alpha\Delta M_k^T. \tag{4.23}$$

By an analogous argument for M_{k+1}, we obtain:

$$M_{k+1} = (I - \alpha W_k^TW_k)M_k + \alpha L_k^T\Delta. \tag{4.24}$$

In the original proof, (4.3) and (4.4) were used to derive the recursive update forms for W^*R_{k+1} (4.5) and $S_{k+1}H^*$ (4.6) respectively. We will show that, even when (4.23) is used in place of (4.3), (4.5) does not change. We have:

$$\begin{aligned}
W^*R_{k+1} &= W_{k+1} - L_{k+1} \\
&= W_k(I - \alpha H_kH_k^T) + \alpha VH_k^T - L_k(I - \alpha H_kH_k^T) - \alpha\Delta M_k^T \\
&= W^*R_k(I - \alpha H_kH_k^T) + \alpha(W^*H^* + \Delta)H_k^T - \alpha\Delta M_k^T \\
&= W^*R_k(I - \alpha(S_kH^* + M_k)(H^{*T}S_k^T + M_k^T)) + \alpha W^*H^*(H^{*T}S_k^T + M_k^T) \\
&\quad + \alpha\Delta(H^{*T}S_k^T + M_k^T) - \alpha\Delta M_k^T \\
&= W^*R_k(I - \alpha S_kH^*H^{*T}S_k^T - \alpha M_kM_k^T) + \alpha W^*H^*H^{*T}S_k^T \\
&= W^*R_k(I - \alpha M_kM_k^T) + \alpha W^*(I - R_kS_k)H^*H^{*T}S_k^T,
\end{aligned}$$

which is exactly the same as (4.5). Thus, the new assumptions do not change the recursive update (4.5). A similar argument can be used to show that (4.6) does not change.

Furthermore, since the recursive updates (4.7), (4.18), and (4.20) are derived entirely from (4.5) and (4.6), we can conclude that none of the other recursive updates change under the new assumptions..

Due to the changes in (4.23) and (4.24), the induction assumptions (4.8) and (4.9) from Theorem 6 need to be slightly adjusted and re-verified. We state the new assumptions required:

$$||L_k|| \leq m_1 m_2^k \quad \text{where } m_2 < 1, \tag{4.25}$$

$$||M_k|| \leq m_1 m_2^k \quad \text{where } m_2 < 1. \tag{4.26}$$

Note that we have utilized the assumption that $m_2 \geq l_2$ (from the Theorem 6 statement) to weaken the assumptions of (4.8) somewhat. We define:

$$l_2 = m_2 = 1 - \frac{\alpha}{(t_0+1)^2} + \alpha\delta.$$

We note that our new assumption (f) guarantees that l_2 and m_2 will be < 1. By taking norms of (4.23), we have:

$$\begin{aligned}
||L_{k+1}|| &\leq ||L_k|| \cdot ||I - \alpha H_k H_k^T|| + ||\alpha \Delta M_k^T|| \\
&= ||L_k|| \cdot ||(I - \alpha S_k H^* H^{*T} S_k^T - \alpha M_k M_k^T)|| + \alpha\delta ||M_k^T|| \\
&\leq ||L_k||(1 - \alpha \lambda_{min}(S_k H^* H^{*T} S_k^T)) + \alpha\delta ||M_k^T|| \\
&\leq ||L_k|| \left(1 - \frac{\alpha}{(t_0 + t_1 \sum_{i=1}^{\infty} t_2^i)^2}\right) + \alpha\delta ||M_k^T|| \\
&\leq ||L_k|| \left(1 - \frac{\alpha}{(t_0+1)^2}\right) + \alpha\delta ||M_k^T|| \\
&\leq m_1 m_2^k \left(1 - \frac{\alpha}{(t_0+1)^2}\right) + \alpha\delta m_1 m_2^k \\
&= m_1 m_2 \left(1 - \frac{\alpha}{(t_0+1)^2} + \alpha\delta\right) \\
&= m_1 m_2^{k+1}.
\end{aligned}$$

Thus, we verify that the induction hypothesis holds, and (4.25) is verified. A similar argument can be used to show that (4.26) will hold. Since none of the other induction arguments rely on (4.3) or (4.4), or the definition of l_2 or m_2, we can apply the same proof as was used in Theorem 6 immediately to show that L_k, M_k, and $W^*(I - R_k S_k)H^*$ all go to zero. Thus, we have that $W_k H_k$ goes to $W^* H^*$, proving Theorem 7. □

Discussion of Theorem 7

From the above proof, we know that L_k, M_k, and $W^*(I - R_kS_k)H^*$ all go to zero; however, we can no longer guarantee that $V - W_kH_k$ goes to zero. In fact, from the problem assumptions regarding W^* and H^*, the minimum value that the objective function $\frac{1}{2}||V - W_kH_k||_F^2$ can attain is $\frac{1}{2}||\Delta||_F^2$; this is due to the fact that (W^*, H^*) was assumed to be a minimum loss solution, and the value of the objective function at (W^*, H^*) is $\frac{1}{2}||\Delta||^2$).

From the result above, we have shown that the value W_kH_k tends to W^*H^* as $k \to \infty$. From this, we have that:

$$\frac{1}{2}||V - W_kH_k||_F^2 = \frac{1}{2}||W^*H^* + \Delta - W^*H^*||_F^2 = \frac{1}{2}||\Delta||_F^2.$$

Thus, the solutions found from Algorithm 3 attain the minimum possible objective function value.

As above, this theorem requires an initialization very close to these matrices W^* and H^* (as we require δ to be small), but in practice Algorithm 3 was observed to converge from a randomly initialized starting point.

4.2 SGD for Low-Rank Matrix Factorization

4.2.1 SGD with a Known Zero-Loss Solution

We now move on from the simplified Gradient Descent case to analyzing Algorithm 2: SGD for low-rank matrix factorization. For this result, we again first assume that we have known rank-r zero-loss solution matrices, W^* and H^*, such that $V = W^*H^*$. We have the following result:

Theorem 8. *Let all of the assumptions of Theorem 6 hold. In all instances where α appears, replace it with the decreased step-size $\frac{\alpha}{n}$. Then stochastic gradient descent (Algorithm 2) starting from the specified initial point will generate a sequence of iterates (W_k, H_k) that satisfy the following equations simultaneously:*

$$\mathbb{E}[||L_{k+1}||_F^2 | L_k] = \left\| L_k \left(I - \frac{\alpha}{n} H_k H_k^T \right) \right\|_F^2, \tag{4.27}$$

$$\mathbb{E}[||M_{k+1}||_F^2|M_k] = \left\| \left(I - \frac{\alpha}{n} W_k^T W_k \right) M_k \right\|_F^2, \tag{4.28}$$

$$\begin{aligned}\mathbb{E}[||K_{k+1}||_F^2|K_k] = || \left(I - \frac{\alpha}{n} W^* R_k R_k^T W^{*T} \right) K_k \left(I - \frac{\alpha}{n} H^{*T} S_k^T S_k H^* \right) \\ + \frac{\alpha}{n} W^* R_k (M_k M_k^T + L_k^T L_k) S_k H^* ||_F^2\end{aligned} \tag{4.29}$$

where $K_k := W^*(I - R_k S_k) H^*$.

Proof. We begin by noting that the recursive updates for both W_k and H_k in Algorithm 2 can be written compactly using unit vector notation, where e_j represents the vector of size $n \times 1$ (where $V \in \mathbb{R}^{m \times n}$) with all entries zero except for the entry j, which is 1:

$$W_{k+1} = W_k + \alpha(V - W_k H_k) e_j e_j^T H_k^T, \tag{4.30}$$

$$\begin{aligned} H_{k+1} &= H_k + \alpha W^T (V(:, j) - W_k H_k(:, j)) e_j^T \\ &= H_k + \alpha W^T (V - W_k H_k) e_j e_j^T \end{aligned}. \tag{4.31}$$

We begin with the recursive update for L_{k+1}. Recall that in the deterministic case, this recursive update is given by (4.3). We consider a single stochastic update of L_{k_1}:

$$\begin{aligned} L_{k+1} &= (I - W^*(W^{*T} W^*)^{-1} W^{*T}) W_{k+1} \\ &= (I - W^*(W^{*T} W^*)^{-1} W^{*T})(W_k + \alpha(V - W_k H_k) e_j e_j^T H_k^T \\ &= L_k + \alpha(I - W^*(W^{*T} W^*)^{-1} W^{*T})(W^* H^*) e_j e_j^T H_k^T \\ &\quad - \alpha(I - W^*(W^{*T} W^*)^{-1} W^{*T}) W_k H_k e_j e_j^T H_k^T \\ &= L_k - \alpha L_k H_k e_j e_j^T H_k^T \\ &= L_k(I - \alpha H_k(:, j) H_k(:, j)^T). \end{aligned}$$

This recursive update form looks similar to the recursive update for L_{k+1} found in Theorem 6, with the exception that only a single column j of the matrix H_k is used. If we take expectations of this, we obtain:

$$\begin{aligned} \mathbb{E}[L_{k+1}|L_k] &= \frac{1}{n}(L_k(I - \alpha H_k(:, 1) H_k(:, 1)^T) + ... + \frac{1}{n}(L_k(I - \alpha H_k(:, n) H_k(:, n)^T) \\ &= L_k \left(I - \frac{\alpha}{n} H_k H_k^T \right). \end{aligned}$$

We observe that this is the recursive update (4.3) found for the full Gradient Descent case, with the step-size reduced by a factor of $\frac{1}{n}$.

For M_{k+1}, where the deterministic recursive update is given by (4.4), we have:

$$\begin{aligned}
M_{k+1} &= H_{k+1}(I - H^{*T}(H^* H^{*T})^{-1})H^*) \\
&= (H_k + \alpha W_k^T (V - W_k H_k) e_j e_j^T)(I - H^{*T}(H^* H^{*T})^{-1})H^*) \\
&= M_k + \alpha W_k^T (W^* H^* - W_k H_k) e_j e_j^T - \alpha W_k^T (W^* H^* - W_k H_k) e_j e_j^T (H^{*T}(H^* H^{*T})^{-1} H^*).
\end{aligned}$$

Note that due to the placement of the e_j vectors, we cannot express this recursive update in a more simplified form. By taking expectations, we observe that, as with the L_{k+1} case, this update reduces to a nice form:

$$\begin{aligned}
\mathbb{E}[M_{k+1} | M_k] &= M_k + \frac{\alpha}{n} W_k^T (W^* H^* - W_k H_k)(I - H^{*T}(H^* H^{*T})^{-1})H^*) \\
&= M_k - \frac{\alpha}{n} W_k^T W_k M_k \\
&= \left(I - \frac{\alpha}{n} W_k^T W_k\right) M_k.
\end{aligned}$$

Again, we observe that this is exactly the recursive update (4.4) found for the full Gradient Descent case, with the step-size reduced by a factor of $\frac{1}{n}$.

To show (4.27), we will make use of the identity $||A||_F^2 = trace(A^T A)$. We have:

$$\begin{aligned}
||L_{k+1}||_F^2 &= ||L_k(I - \alpha H_k(:,j) H_k(:,j)^T)||_F^2 \\
&= trace((L_k - \alpha L_k H_k(:,j) H_k(:,j)^T)^T (L_k - \alpha L_k H_k(:,j) H_k(:,j)^T)) \\
&= trace(L_k^T L_k) - trace(\alpha L_k^T L_k H_k(:,j) H_k(:,j)^T) \\
&\quad - trace(\alpha H_k(:,j) H_k(:,j)^T L_k^T L_k) + O(\alpha^2),
\end{aligned}$$

and thus, by taking expectations, we obtain:

$$\begin{aligned}\mathbb{E}[||L_{k+1}||_F^2|L_k] &= \mathbb{E}[trace(L_k^T L_k) - trace(\alpha L_k^T L_k H_k(:,j)H_k(:,j)^T) \\ &\quad - trace(\alpha H_k(:,j)H_k(:,j)^T L_k^T L_k) + O(\alpha^2)|L_k] \\ &= trace(L_k^T L_k) - trace\left(\frac{\alpha}{n} L_k^T L_k H_k H_k^T\right) - trace\left(\frac{\alpha}{n} H_k H_k^T L_k^T L_k\right) + O(\alpha^2) \\ &= trace\left(\left(L_k - \frac{\alpha}{n} L_k H_k H_k^T\right)^T \left(L_k - \frac{\alpha}{n} L_k H_k H_k^T\right) O(\alpha^2)\right) \\ &= \left\| L_k \left(I - \frac{\alpha}{n} H_k H_K^T\right) + O(\alpha^2)\right\|_F^2 .\end{aligned}$$

Neglecting the $O(\alpha^2)$ term, this is the desired result.

To show (4.28), the calculation is completed using the same method. Omitting some details, and denoting $\bar{H} := (H^* H^{*T})^{-1} H^*)$, we have:

$$\begin{aligned}\mathbb{E}[||M_{k+1}||_F^2|M_k] &= \mathbb{E}[||M_k + \alpha W_k^T (W^* H^* - W_k H_k) e_j e_j^T (I - \bar{H})||_F^2 |M_k] \\ &= \mathbb{E}[trace((M_k^T M_k) + trace(\alpha M_k^T W_k^T (W^* H^* - W_k H_k) e_j e_j^T (I - \bar{H})) \\ &\quad + trace((I - \bar{H}) e_j e_j^T (H^{*T} W^{*T} - H_k^T W_k^T) W_k M_k) + O(\alpha^2)|M_k] \\ &= trace\left(\left(M_k - \frac{\alpha}{n} W_k^T W_k M_k\right)^T \left(M_k - \frac{\alpha}{n} W_k^T W_k M_k\right) + O(\alpha^2)\right) \\ &= \left\| \left(I - \frac{\alpha}{n} W_k^T W_K\right) M_k + O(\alpha^2)\right\|_F^2 ,\end{aligned}$$

again, as desired (neglecting the $O(\alpha^2)$ term).

To show (4.29), we first require an expression for a stochastic update of $W^*(I - R_{k+1} S_{k+1}) H^*$. We first find the intermediate quantities $W^* R_{k+1}$ and $S_{k+1} H^*$:

$$\begin{aligned}W^* R_{k+1} &= W_{k+1} - L_{k+1} \\ &= W_k + \alpha(V - W_k H_k) e_j e_j^T H_k^T - L_k (I - \alpha H_k e_j e_j^T H_k^T) \\ &= W^* R_k (I - \alpha H_k e_j e_j^T H_k^T) + \alpha W^* H^* e_k e_k^T H_k^T \\ &= W^* R_k (I - \alpha M_k e_j e_j^T M_k - \alpha S_k H^* e_j e_j^T M_k^T - \alpha M_k e_j e_j^T H^* S_k^T) \\ &\quad + \alpha W^* (I - R_k S_k) H^* e_j e_j^T H^{*T} S_k^T + \alpha W^* H^* e_j e_J^T M_k^T ,\end{aligned}$$

$$\begin{aligned}
S_{k+1}H^* &= H_{k+1} - M_{k+1} \\
&= H_k + \alpha W_k^T(W^*H^* - W_kH_k)e_je_j^T - M_k - \alpha W_k^T(W^*H^* - W_kH_k)e_je_j^T(I - \bar{H}) \\
&= S_kH^* + \alpha W_k^T(W^*H^* - W_kH_k)e_je_j^T\bar{H} \\
&= S_kH^* + \alpha(R_k^TW^{*T}W^*(I - R_kS_k)H^* - R_k^TW^{*T}W^*R_kM_k \\
&\quad - L_k^TL_kS_kH^* - L_k^TL_kM_k)e_je_k^T\bar{H}.
\end{aligned}$$

With these quantities, we can calculate K_{k+1}:

$$\begin{aligned}
K_{k+1} &= W^*(I - R_{k+1}S_{k+1})H^* \\
&= W^*H^* - (W^*R_{k+1})(S_{k+1}H^*) \\
&= W^*H^* - W^*R_kS_kH^* \\
&\quad - \alpha W^*R_k(M_ke_je_j^TM_k + S_kH^*e_je_j^TM_k^T + M_ke_je_j^TH^*S_k^T)S_kH^* \\
&\quad - \alpha W^*(I - R_kS_k)H^*e_je_j^TH^{*T}S_k^TS_kH^* - \alpha W^*H^*e_je_j^TM_k^TS_kH^* \\
&\quad + \alpha W^*R_k(R_k^TW^{*T}W^*(I - R_kS_k)H^* - R_k^TW^{*T}W^*R_kM_k \\
&\quad - L_k^TL_kS_kH^* - L_k^TL_kM_k)e_je_k^T\bar{H} + O(\alpha^2).
\end{aligned}$$

To calculate $trace(K_{k+1}^TK_{k+1})$, the same process as above can be applied. Since each expanded term of $trace(K_{k+1}^TK_{k+1})$ will have at most one $e_je_j^T$ term in it, the expected value calculation will be the same as above, and each $O(\alpha)$ term will gain a factor of $\frac{1}{n}$ (due to the size of the expanded $O(\alpha)$ terms, the details have been omitted). We have:

$$\begin{aligned}
\mathbb{E}[||K_{k+1}||_F^2|K_k] &= \mathbb{E}[trace(K_{k+1}^TK_{k+1})|K_k] \\
&= ||\left(I - \frac{\alpha}{n}W^*R_kR_k^TW^{*T}\right)K_k\left(I - \frac{\alpha}{n}H^{*T}S_k^TS_kH^*\right) \\
&\quad + \frac{\alpha}{n}W^*R_k(M_kM_k^T + L_k^TL_k)S_kH^* + O(\alpha^2)||_F^2,
\end{aligned}$$

which, neglecting the $O(\alpha^2)$ term, completes the proof [4].

□

[4](4.27), (4.28), and (4.29) together establish the foundation for proving the convergence of Algorithm 2 using a supermartingale argument (see [22] for the required martingale theory, and [5] for an example of how such an argument would work). This argument is not considered further here, but is a natural direction for future work.

Discussion of Theorem 8

We observe that (4.27), (4.28), and (4.29) closely resemble the recursive updates (4.3), (4.4), and (4.7) from Theorem 6, with the main difference being that the step-size is reduced by a factor of $\frac{1}{n}$. This suggests that an induction proof similar to the method used in Theorem 6 could be used to show that (4.27), (4.28), and (4.29) will all tend to zero at a linear rate. As with Theorem 6, this convergence would still be very slow, due to both the reduced step-size and the high values of l_2, m_2, and p_2 required.

In the following section, we will discuss generalizing the above result to include the case where no zero-loss solution exists. While the hope is that we can generalize the result in the same way Theorem 7 generalized Theorem 6, the theoretical guarantees we can obtain in this case are more limited.

4.2.2 SGD: Extending to Non Zero-Loss Solution Case

We weaken the assumption that a zero-loss solution exists, and instead we assume that $V = W^*H^* + \Delta$. We assume that W^* and H^* represent the best-possible rank-r approximation of V, implying that $W^{*T}\Delta = 0$ and $\Delta H^{*T} = 0$.

As with Theorems 7 and 8, we will attempt to show how the recursive updates for the key quantities in the proof change based on the new assumptions, and try to apply the same proof structure where possible.

We begin with the recursive update for L_{k+1}, (4.3). We consider a single stochastic update of L_{k_1}:

$$\begin{aligned}
L_{k+1} &= (I - W^*(W^{*T}W^*)^{-1}W^{*T})W_{k+1} \\
&= (I - W^*(W^{*T}W^*)^{-1}W^{*T})(W_k + \alpha(V - W_kH_k)e_je_j^TH_k^T \\
&= L_k + \alpha(I - W^*(W^{*T}W^*)^{-1}W^{*T})(W^*H^* + \Delta)e_je_j^TH_k^T \\
&\quad - \alpha(I - W^*(W^{*T}W^*)^{-1}W^{*T})W_kH_ke_je_j^TH_k^T \\
&= L_k + \alpha\Delta e_je_j^TH_k^T - \alpha L_kH_ke_je_j^TH_k^T \\
&= L_k(I - \alpha H_k(:,j)H_k(:,j)^T) + \alpha\Delta(:,j)H_k(:,j)^T.
\end{aligned}$$

This recursive update form looks similar to the recursive update for L_{k+1} found in Theorem 7, with the expected $\frac{1}{n}$ added for the learning rate. As with Theorem 8, we can take norms (neglecting $O(\alpha^2)$ terms) to show that:

$$
\begin{aligned}
\mathbb{E}[||L_{k+1}||_F^2|L_k] &= \mathbb{E}[||L_k(I-\alpha H_k(:,j)H_k(:,j)^T)+\alpha\Delta(:,j)H_k(:,j)^T||_F^2|L_k] \\
&= \mathbb{E}[trace(L_k^TL_k)-trace(\alpha L_k^T(L_kH_k(:,j)-\Delta(:,j))H_k(:,j)^T) \\
&\quad -trace(\alpha H_k(:,j)(H_k(:,j)^TL_k^T-\Delta(:,j)^T)L_k)+O(\alpha^2)|L_k] \\
&= trace(L_k^TL_k)-trace\left(\frac{\alpha}{n}L_k^T(L_kH_k-\Delta)H_k^T\right) \\
&\quad -trace\left(\frac{\alpha}{n}H_k(H_k^TL_k^T-\Delta^T)L_k+O(\alpha^2)\right) \\
&= trace\left(\left(L_k-\frac{\alpha}{n}(L_kH_k-\Delta)H_k^T\right)^T\left(L_k-\frac{\alpha}{n}(L_kH_k-\Delta)H_k^T\right)+O(\alpha^2)\right) \\
&= \left\|L_k\left(I-\frac{\alpha}{n}H_kH_K^T\right)+\frac{\alpha}{n}\Delta H_k^T+O(\alpha^2)\right\|_F^2. \qquad (4.32)
\end{aligned}
$$

Similarly, on the M_{k+1} side, we have:

$$
\begin{aligned}
M_{k+1} &= H_{k+1}(I-H^{*T}(H^*H^{*T})^{-1})H^*) \\
&= (H_k+\alpha W_k^T(V-W_kH_k)e_je_j^T)(I-H^{*T}(H^*H^{*T})^{-1})H^*) \\
&= M_k+\alpha W_k^T(W^*H^*+\Delta-W_kH_k)e_je_j^T-\alpha W_k^T(W^*H^* \\
&\quad +\Delta-W_kH_k)e_je_j^T(H^{*T}(H^*H^{*T})^{-1}H^*),
\end{aligned}
$$

and we can take expectations and norms to show that:

$$
\mathbb{E}[||M_{k+1}||_F^2|M_k] = \left\|\left(I-\frac{\alpha}{n}W_k^TW_K\right)M_k+\frac{\alpha}{n}W_k^T\Delta+O(\alpha^2)\right\|_F^2. \qquad (4.33)
$$

Since the recursive updates for L_{k+1} and M_{k+1} are similar to (4.23) and (4.24) in expectation, we would perhaps expect that we could apply the same proof as for Theorem 7 here, and conclude that both L_{k+1} and M_{k+1} will converge to zero. However, as we will explain below, we do not have this guarantee.

For the recursive updates (4.32) and (4.33), even though the expectations of the stochastic updates appear that they could converge to zero, this does not provide any guarantee that the stochastic updates themselves will converge. Instead, we would expect the additive Δ terms to introduce a residual error. In addition, in the stochastic gradient realm, the choice to neglect the $O(\alpha^2)$ terms is more problematic than in the deterministic realm. During

Gradient Descent, we would expect the higher order terms of the Taylor expansion to go to zero at a quicker rate than the first order terms. For SGD, however, we do not have this guarantee, due to the fact that the stationary points for each stochastic gradient estimate are not guaranteed to be stationary points of the true objective function gradient. These observations help to explain why SGD with no zero-loss solution does not enjoy the same performance guarantees as shown in Theorems 7 and 8.

A Heuristic for the Limiting Value of $V - W_k H_k$

Consider the true residual $V - W_k H_k$. As discussed in 4.1.2, under Algorithm 3 we obtain the minimum possible loss value $\frac{1}{2}||\Delta||_F^2$. We would expect Algorithm 2 to behave almost as well as Algorithm 3, with some potential noise around the solution point due to the noise inherent in the stochastic algorithm.

We now present a heuristic for how close Algorithm 2 gets to the known minimal loss solution. Pflug [60] studied the oscillations of stochastic minimization with a fixed step-size around a stationary distribution (in our case, a stationary point in Algorithm 2). Pflug restated a result from Kushner and Hai Huang that the distribution $\alpha^{-\frac{1}{2}}(x_k - x^*)$ will converge to a normal distribution with zero mean and covariance proportional to the covariance of the stochastic gradient estimate (in our case, Δ^2). This suggests that we would expect the distance between iterates, $||x_k - x^*||_F$, to be roughly $||\alpha^{\frac{1}{2}}\Delta||_F$, or the distance between the function values $f(x_k) - f(x^*)$ to be roughly $\alpha||\Delta||_F^2$.

This quantity, $\alpha||\Delta||_F^2$, is the consistently observed difference between the loss value obtained by Gradient Descent (Algorithm 2) and the loss value obtained by SGD (Algorithm 3) (see 5.1.1). This is not a new observation: Dieuleveut et al. [19] cited Pflug to explain why iterates of SGD with a fixed step-size will oscillate with an average magnitude of $O(\alpha^{\frac{1}{2}})$, and why SGD will stop converging with a loss proportional to α. The results from Pflug and Dieuleveut are only proved in the strongly convex regime (see 2.2.1) and thus are not directly applicable here; as discussed by Zhu et al., the non-uniqueness of the optimal solutions to the matrix factorization problem ensure that matrix factorization problem is not strongly convex [75]. Despite this, the Pflug result provide a helpful heuristic to explain the expected performance gap between Gradient Descent and SGD.

We note that the Pflug heuristic does not contradict the conclusions of Theorem 8, where a known zero-loss solution exists. In this situation, the stationary points for each stochastic gradient estimate are also stationary points for the objective function. This suggests that the covariance of the stochastic gradient estimate is itself zero, and thus the covariance of the limiting normal distribution will also be zero. As a result, the value $(f(x_k) - f(x^*)) \approx \alpha||\Delta||_F^2$, should tend to zero as $k \to \infty$, as expected.

Chapter 5

Experiments

5.1 Observing Convergence Using Synthetic Data

5.1.1 Verifying Convergence Results from Chapter 4

As a first experiment, we verify that the results from Chapter 4 can be applied more broadly in experiments; specifically, we observe that the results from Chapter 4, which only applied when Algorithms 2 and 3 were initialized close to a minimizer, will hold when the algorithm is initialized at a random point.

In the following figures, we show the results of two such experiments. Figures 5.1 and 5.3 show two test runs, with varying α parameter. Each figure shows the trend in objective value when Algorithm 2 and Algorithm 3 are run from the same randomly initialized data point[1], and each figure displays two cases: a large Δ case[2] where $||\Delta||_F \approx 10^{-3}$, and a small Δ case where $||\Delta||_F \approx 10^{-5}$. All experiments in this Chapter are performed in MATLAB R2018a on a laptop with a 3.1 GHz Intel Core i5 and 8 GB RAM.

We observe that, as $k \to \infty$, the difference in the objective value attained via SGD (Algorithm 2) and the objective value attained via Gradient Descent (Algorithm 3) is well-

[1]The columns of the initial points W_0 and H_0 are initialized on the appropriately-sized simplex using a Dirichlet distribution

[2]To generate a case with a specific Δ, we first generate a random rank-r W and H, and a random matrix D of the desired magnitude. We let $V = WH * D$, and use the SVD of find the best rank-r factors of V, which we call W^* and H^*. We then let $\Delta = V - W^*H^*$.

approximated on average by $\frac{1}{2}\alpha||\Delta||_F^2$, which is the value suggested by the Pflug heuristic discussed in 4.2.2.

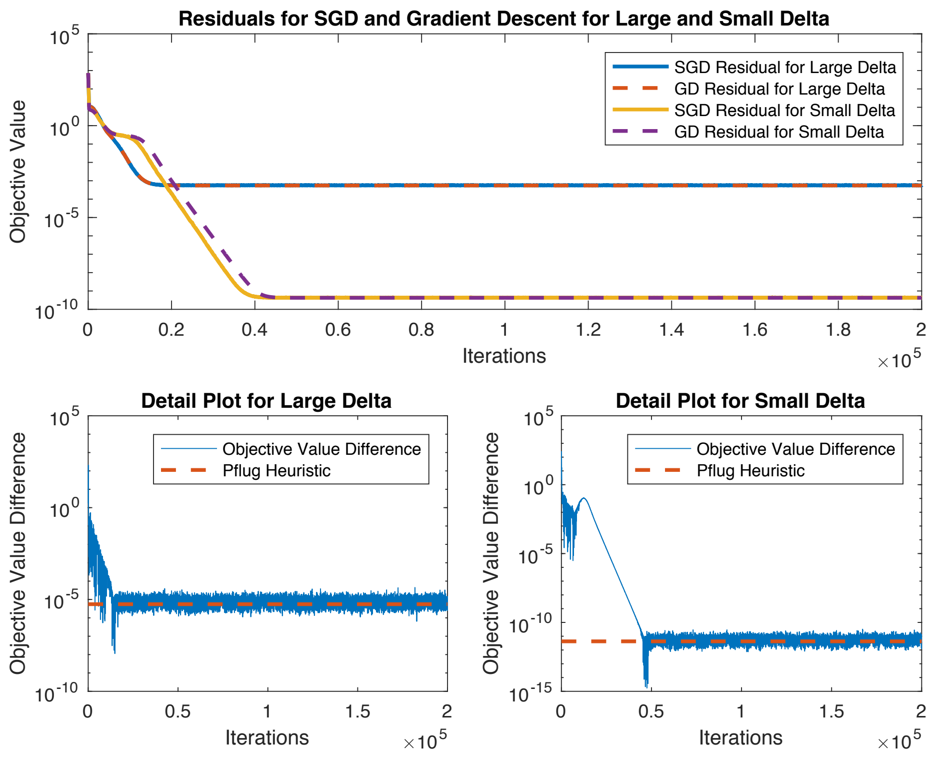

Figure 5.1: Convergence of SGD and Gradient Descent for $\alpha = 0.01$

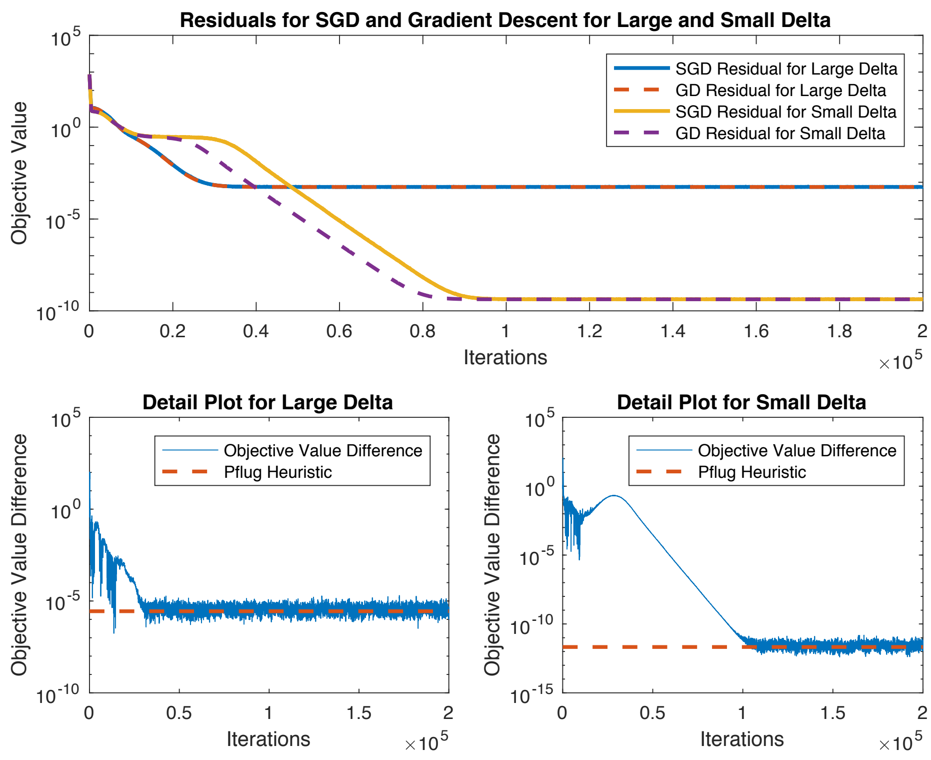

Figure 5.2: Convergence of SGD and Gradient Descent for $\alpha = 0.005$

5.1.2 Experiments with Projection

Here, we show that, even with the projection, the results from Chapter 4 appear to hold, suggesting that the results from Chapter 4 can be extended to the NMF case and Algorithm 1. We show a $\Delta > 0$ result.

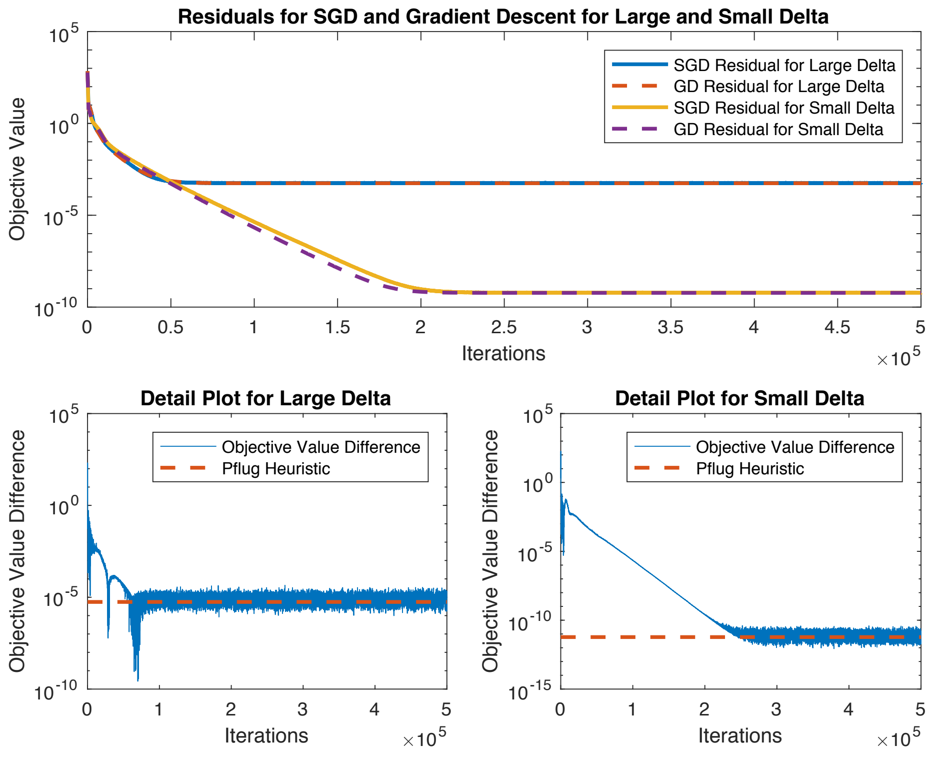

Figure 5.3: Convergence of SGD and Gradient Descent with Projection for $\alpha = 0.01$

We observe again that the Pflug heuristic does a good job of estimating the difference between the optimal value achieved by Gradient Descent and the optimal value achieved by SGD, even in the projected case. While we only show a single example here, this consistency with the non-projected results holds for both higher dimensional test cases and a variety of α and $||\Delta||$ values. This suggests that, with a more complicated analysis, the results from 4.1.2 and 4.2.2 could be extended to include the projected case.

5.2 Using Matrix Factorization for Classification: MNIST Dataset

5.2.1 MNIST Dataset Description

The MNIST dataset is a large dataset of labelled handwritten digits that can be used for supervised machine learning experiments [45]. The example digits have been centred and resized to be displayed as 28×28 pixel images. The training set, which we will denote as V, consists of $60,000$ examples, and the test set, denoted T, has an additional $10,000$ examples. We utilize a series of helper functions [52] to reformat each image as a 784×1 vector; thus, the matrix factorization problem using $60,000$ images is equivalent to factoring a 784×60000 matrix. Figure 5.4 below shows the first 100 examples from the dataset:

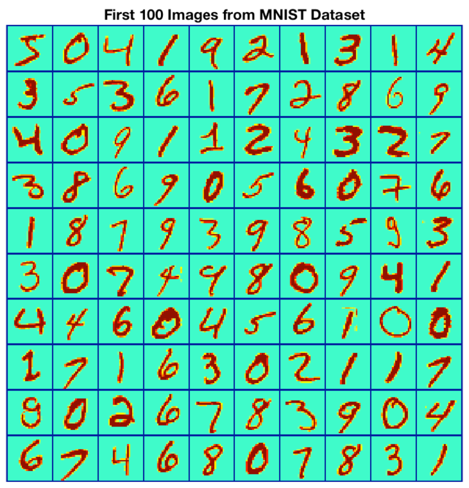

Figure 5.4: First 100 MNIST Digits

5.2.2 Supervised Machine Learning Through Matrix Factorization for MNIST

Motivated by the work of Erichson et al. [26] (see 2.3.3), we evaluate the ability of SGD and matrix factorization to be used as a tool for classification. We consider two different ways that SGD can be applied to solve this problem.

First Approach: Direct Application of Algorithms 2 and 3

For our first approach, we use Algorithms 1 and 2 to perform matrix factorization on the training set matrix V, with a fixed target rank of $r = 16$, and use the found W_k to project the training and test sets onto the low-dimensional space. Specifically, for each test point $t_i := T(:, i)$, we determine its low-dimensional representation $\hat{t}_i$ in the non-projected case (Algorithm 2) by solving:

$$\hat{t}_i = \underset{x}{\operatorname{argmin}}\{Wx - t_i\}. \tag{5.1}$$

We repeat this for each $\hat{t}_i$ to form the low-dimensional representation of the test set $\hat{T}$. In the projected case (Algorithm 1), we find the low-dimensional representation of each test point by solving the nonnegative version of (5.1):

$$\hat{t}_i = \underset{x \geq 0}{\operatorname{argmin}}\{Wx - t_i\}. \tag{5.2}$$

In both the unconstrained and nonnegative case, we apply the same process to find the low-dimensional representation of V, defined as $\hat{V}$. To perform classification, we select a random sample of points from $\hat{V}$ to use for 3-nearest neighbour ($3NN$) classification; we denote this set $\hat{V}_a$ where $|\hat{V}_a| = a$. The $3NN$ classification algorithm determine the 3 points of $\hat{V}_a$ that are closest to each projected training point $\hat{t}_i$, and classifies the training point based on the most common label among these 3 points (in the case of ties, the closest projected training point determines the label). We define the error as the percentage of total test points that are labelled incorrectly.

The following figures show the results of a test run of Algorithm 2 with $a = 40000$: Figure 5.5 shows the first 100 digits of the low-rank approximation $W_k H_k$ of V, and Figure 5.6 shows the first 100 digits of the test set before and after projection onto W_k.

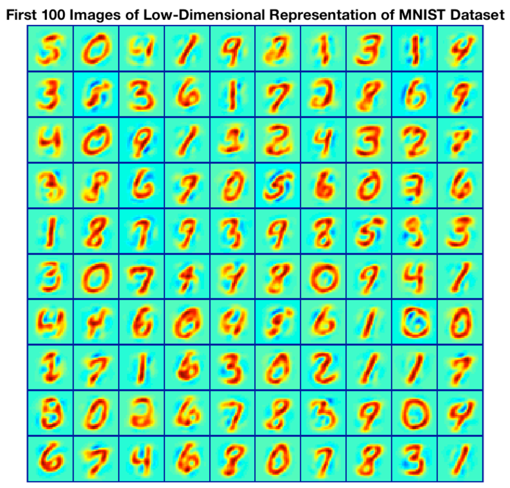
First 100 Images of Low-Dimensional Representation of MNIST Dataset

Figure 5.5: First 100 MNIST Digits of $W_k H_k$ After Factorization by Algorithm 2

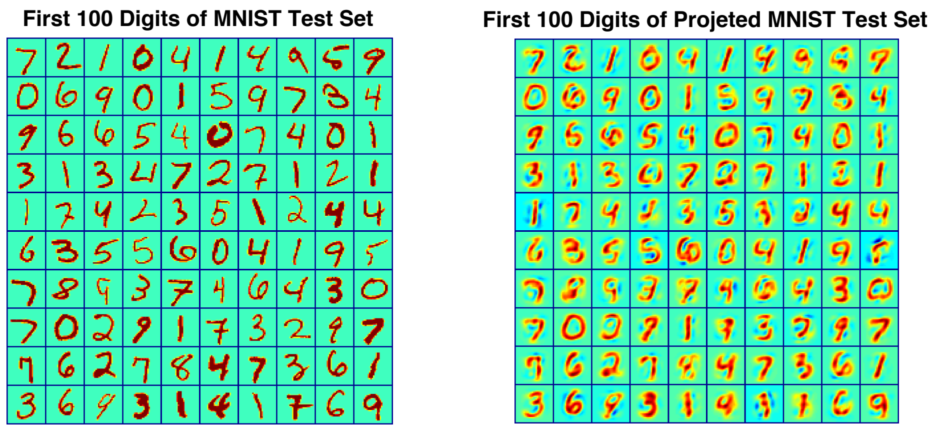
First 100 Digits of MNIST Test Set
First 100 Digits of Projeted MNIST Test Set

Figure 5.6: First 100 Digits of MNIST Test Set Before/After Low-Dimensional Projection

Second Approach: Limiting the Dimension of H

We observe that, by directly applying Algorithms 1 and 2 to the training set V, we will maintain a matrix H at each step with one column per data point. We propose an alternative algorithm that limits the column dimension of H: instead of having one column per data point, we let $H \in \mathbb{R}^{r\times 10}$, and designate one column of H for each possible label $0, 1, ..., 9$. On the $k-th$ iteration, when a column of $V(:, j)$ with label l is selected by the algorithm, we update the full matrix W_k as before, but instead of updating the column of $H_k(:, j)$, we update the column $H_k(:, l + 1)$[3]. In this way, each column $H_k(:, l + 1)$ should represent an average of all of the examples labelled l as more and more iterations occur. Specifically, we define the following modified versions of Algorithms 1 and 2, which we will call Algorithms 4 and 5:

Algorithm 4: Modified Version of Algorithm 1: SGD for NMF

```
for $k$=1,2,... do
    Choose $j$ from $[1, n]$ uniformly at random;
    Denote $l$ as the label of $j$, $l \in [0, 9]$;
    $W_{k+1} = max\{W_k - \alpha(V(:, j) + W_kH_k(:, l + 1))H_k(:, l + 1)^T, 0\}$;
    $H_{k+1}(:, j) = max\{H_k(:, l + 1) - \alpha W_k^T(V(:, j) + W_kH_k(:, l + 1)), 0\}$;
end
```

Algorithm 5: Modified Version of Algorithm 2: Matrix Factorization for NMF

```
for $k$=1,2,... do
    Choose $j$ from $[1, n]$ uniformly at random;
    Denote $l$ as the label of $j$, $l \in [0, 9]$;
    $W_{k+1} = W_k - \alpha(V(:, j) + W_kH_k(:, l + 1))H_k(:, l + 1)^T$;
    $H_{k+1}(:, j) = H_k(:, l + 1) - \alpha W_k^T(V(:, j) + W_kH_k(:, l + 1))$;
end
```

The following figures provide some insight into this approach. With $r = 16$, we apply Algorithm 4. Figure 5.7 shows two images: the first is the columns of W_k found by the algorithm (i.e. the basis vectors of the low-dimensional space that was found), and the second is the product W_kH_k (i.e. an "average" representation of each digit in the low-dimensional space).

[3]MATLAB does not use the zero-indexing convention for counting columns, so in order to represent the first column as the label 0 column, we must use the index 1, and so on, hence the index $l + 1$.

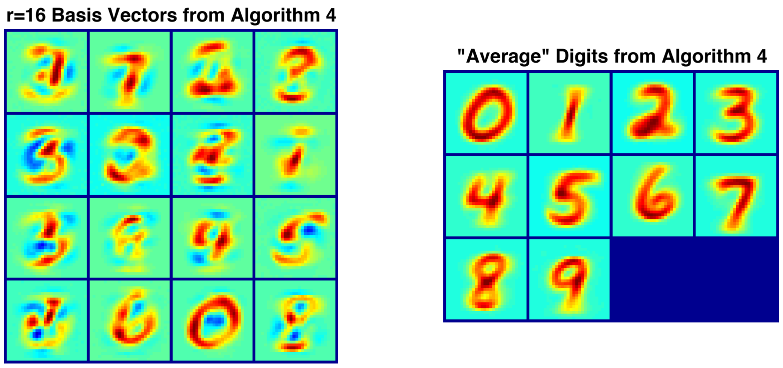
r=16 Basis Vectors from Algorithm 4
"Average" Digits from Algorithm 4

Figure 5.7: r Basis Vectors Found by the Modified Algorithm 1, and the Representation of Each “Average” Digit in this Basis

Erichson et al. [26] measure the performance of the Hierarchical Alternating Least Squares (HALS) NMF algorithm (see 2.1.2 or [17] for further detail) against a randomized version of HALS, and compares their classification error. We compare both the training and test error from our test runs to their results in Table 5.1[4][5]:

Training and Test Error for Various Matrix Factorization Methods		
Erichson et al. Results		
Method	Training Error	Test Error
Deterministic HALS	0.03	0.05
Randomized HALS	0.03	0.05
SGD: First Approach		
Algorithm 1	0.06	0.08
Algorithm 2	0.03	0.05
SGD: Second Approach		
Algorithm 4	0.10	0.12

Table 5.1: Comparison of Classification Error for Various Matrix Factorization Algorithms

[4] In [26], the error is broken down as two specific quantities, precision and recall, which are quantities that indicate the occurrence of false positives and negatives in binary classification. For the MNIST dataset, precision, recall, and training/test error are in fact all equivalent.

[5] For Algorithms 1, 2, and 4, we initialize the columns of W and H with vectors selected from the appropriately-sized unit simplex; this selection tended to produce favourable results.

We observe that, in the first approach where H is formulated with $n = 60000$ columns, Algorithm 2 is competitive with the results from Erichson et al. in terms of both training error and test error, and Algorithm 1 produces similar, but slightly weaker, results. We note that, as Algorithms 1 and 2 will only update W_k in a limited, rank-1 manner each time, we require significantly more iterations of Algorithm 2 than either the deterministic or randomized HALS algorithm to obtain the results.

We also observe that Algorithm 4, where H is formulated with $n = 10$ columns, performs reasonably well, with higher training and test error. We note that, since the parameter n is fixed at a low value no matter how large our dataset is, this approach will scale much better with very large datasets; for example, in the Big Data regime where n is massive, the trade-offs in decreased accuracy may be desirable in order to improve runtime.

We omit the results for Algorithm 5 as it performs very poorly as a classifier. Upon studying this test case further, we observed that, when projected onto the low-dimensional space spanned by W_k, the corresponding test set $\hat{T}$ contained vectors with very large negative components. These test vectors were very difficult to classify correctly using the $3NN$ algorithm. This is in contrast with Algorithm 2, where the projected test set vectors had entries that were either nonnegative, or very slightly negative (and this were well-approximated by zero).

5.2.3 Effects of Varying α on Test Error for MNIST Classification

In the following experiments, we aim to understand the impact that varying the step-size α can have on test error performance. As discussed in 2.3.1, improving the training error will not always correspond to an improvement in the test error; we can evaluate this using the MNIST dataset.

Erichson et al. observed that, after a certain number of optimization steps, further optimization over the training set does not improve the test set classification error [26]. We observe this same behaviour for the SGD algorithm; after initial improvement, the test stabilizes around a consistent value for further iterations. We demonstrate this behaviour in Figure 5.8 below, over a smaller test case with $n = 10000$ and $a = 8000$, by showing how the test error decreases roughly proportionally to the norm of the objective function gradient.

We evaluate this behaviour further by considering the effects of varying the step-size. As suggested by the Pflug heuristic, decreasing the step-size should allow Algorithm 2 to reach

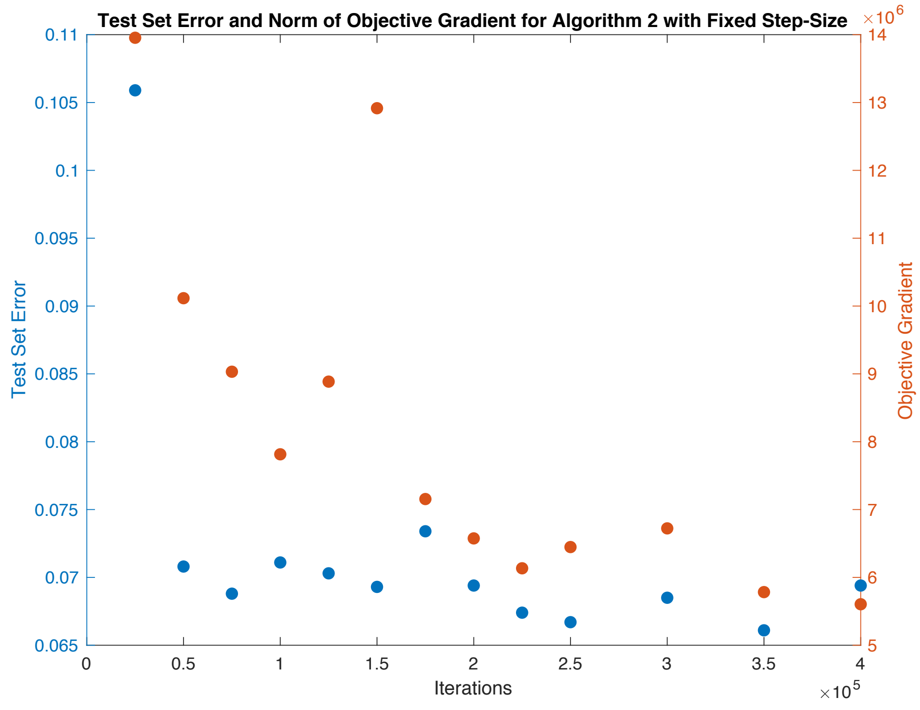

Figure 5.8: Test Set Error and Objective Gradient for Algorithm 2 with $\alpha = 0.02$

a smaller objective value (and thus, likely, a smaller training error). This improvement, however, will come at the cost of a slower convergence due to the decrease in the step-size. We wish to understand whether the decrease in objective value from using a smaller α corresponds to a decrease in the test error, or if the test error plateaus.

In Table 5.2, we show the results of a number of test runs of Algorithm 2 with varying α. To ensure each test run can be compared with one another, the product αT is kept fixed (i.e. as the step-size decreases, the number of iterations is increased proportionally). Table 5.2 shows that, as α is decreased, we do observe a corresponding decrease in the objective value, although the decrease is small relative to the objective value magnitude. We observe that, despite the decreases in α, the test error remains within an $\sim 0.3\%$ range; thus, the additional iterations required to improve the objective value have little to no impact on the test error.

Objective Value and Test Error for Varying α and T with Constant αT			
α	T	Test Error	Obj. Value ($\times 10^5$)
2×10^{-2}	5×10^5	6.78%	1.0633
1×10^{-2}	1×10^6	6.65%	1.0574
8×10^{-3}	1.25×10^6	6.91%	1.5610
6×10^{-3}	1.67×10^6	6.55%	1.0555
5×10^{-3}	2×10^6	6.71%	1.0552
4×10^{-3}	2.5×10^6	6.79%	1.0548
2.5×10^{-3}	4×10^6	6.66%	1.0546
2×10^{-3}	5×10^6	6.84%	1.0545
1×10^{-3}	1×10^7	6.88%	1.0543
9×10^{-4}	1.1×10^7	6.78%	1.0543
7×10^{-4}	1.43×10^7	6.74%	1.0542
5×10^{-4}	2×10^7	7.04%	1.0542
1×10^{-4}	1×10^8	6.62%	1.0541

Table 5.2: Objective Value and Test Error for Algorithm 2 with Varying α and T, and Constant αT

Figure 5.8 and Table 5.2 suggest a specific strategy for minimizing test error for a given test case. Matrix factorization should be run at a large α value (perhaps as large as possible without causing the algorithm to fail) to minimize the runtime and test error jointly. Any additional optimization, while potentially improving the training error (if this is of interest), does not appear to have a substantial effect on the test set error.

5.2.4 Unsupervised Machine Learning Through NMF for MNIST

As discussed in 2.3.3, using NMF as a tool for unsupervised machine learning has been explored extensively in the literature [48]. In the following, we discuss a specific example of how NMF can be used as a dimensionality reduction tool, using the MNIST digits.

Given the 60000 training examples in the MNIST dataset, we can use an algorithm such as k-means clustering to try to group the training examples into clusters. For the digits example, it is natural to attempt to group the digits into one cluster per digit, or 10 clusters total. Since each example has many features (specifically, each digit v is in $\mathbb{R}^{784}$), this can be an expensive procedure.

In the following experiment, we use NMF to find a lower-dimensional basis W, and project each digit onto this lower-dimensional set. We then compare the performance of the k-means algorithm on both the original training examples and the lower-dimensional projected examples. To do this, we use the following experimental procedure:

1. Select 10 random digits, one of each label, to initialize as the k-means algorithm centroids
2. Run the k-means algorithm to termination on the full, high-dimension training data
3. Label each of the 10 clusters based on the most common label of all of the digits in that cluster (in event of a tie, select the larger label)
4. Calculate the classification error: $\frac{\text{Number of points with a label that does not match its cluster label}}{\text{Total number of data points}}$
5. Check to see if each of the clusters is labelled with a unique digit from 0 to 9. If so, we call this a "good" clustering, and if not we call it a "bad" clustering.
6. Run Algorithm 4 for a predetermined number of iterations t_1
7. Project the training digits onto the low-dimensional set spanned by W_{t_1}
8. Initialize the k-means algorithm using the projected version of the 10 digits from Step 1
9. Run the k-means algorithm to termination on the projected data data
10. Repeat the labelling procedure from Steps 3 and 4 to determine the low-dimensional error rate
11. Repeat Steps 6 to 10 for various numbers of iterations $t_2, ..., t_i$.

We run this experiment on three different test cases (A, B, C), representing a different random initialization of centroids in Step 1. We summarize the results in Figure 5.9. In the figure, each circle in the plot represents the classification error rate for the indicated case after the indicated number of iterations, with the filled circles representing "good" clusterings and the empty circles representing "bad" clusterings. Note that the high-dimensional initial case is shown on the vertical axis at Iteration 0, and is marked by crosses.

We observe that, in all three test cases, the k-means clustering algorithm fails to identify 10 clusters with distinct labels when conducted on the full, high-dimensional data. After

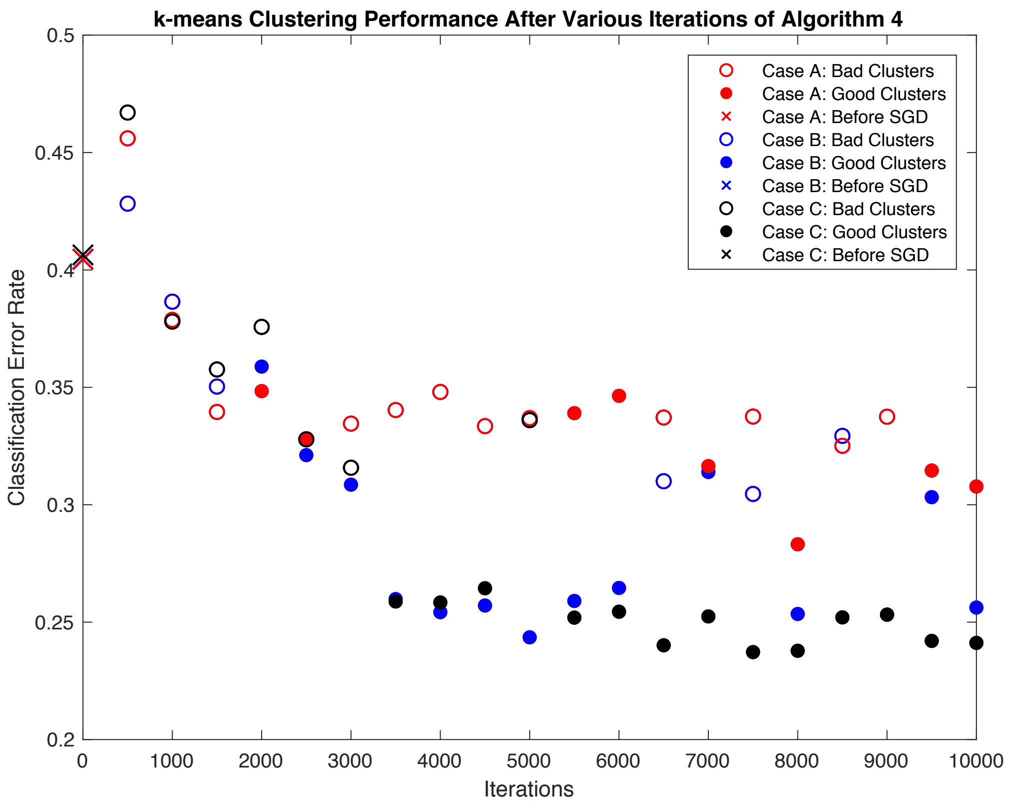

Figure 5.9: k-means Clustering for 3 Cases After Varying Number of Algorithm 4 Iterations

running Algorithm 4 for some iterations, all three cases show an improvement in overall classification error, and successfully find a clustering solution where each cluster is uniquely labelled by majority vote. These results support the idea that NMF can be used not only as a dimensionality reduction tool to simplify data science and machine learning problems, but also can be used to improve the performance of classification problems.

5.3 Topic Classification with NMF for the Latent Dirichlet Allocation Model

5.3.1 Latent Dirichlet Allocation and the NMF Problem

As a final set of experiments, we apply Algorithm 1 to the Latent Dirichlet Allocation topic model example, as introduced in 2.3.3. Specifically, we assume that there exists an unknown word-to-topics matrix W^*, and the topics-to-documents matrix is generated by a Dirichlet prior with unknown parameters. The goal is to use Algorithm 3 to recover the unknown matrix W^* as accurately as possible. For the following experiments, we generate the W^* matrix with random values concentrated near the main diagonal of the matrix; this is somewhat similar to the separability assumption introduced in 2.1.2[6].

In order to apply Algorithm 1, we require a data matrix V^*. In our experiments, there are 4 parameters that need to be selected in advance: the step-size α, the r-vector of Dirichlet parameters β, the number of documents n, and the number of words in each document N. We set up the problem as follows:

1. Sample the Dirichlet distribution to obtain an r-vector probability vector (i.e. all entries are nonnegative and sum to 1)

2. Sample the probability vector with a categorical distribution N times; this can be though of as generating a column $H^*(:, i)$, where H^* is a "true" topics-to-documents matrix.

3. Multiply W^* by $H^*(:, i)$ to obtain $V^*(:, i)$, which can be thought of as a new document.

4. Repeat the above steps n times to obtain a set of n documents (n should be chosen large enough to ensure each of the r topics are sampled at least once)

This process gives us a set of "true" documents that represent the underlying Dirichlet distribution. We can certainly perform matrix factorization on the matrix V^* and try to recover W^*; a more interesting problem is to use V^* to generate new documents, and use

[6]Specifically, given $W^* \in R^{m\times r}$, a row index i, and a column index j, if $\lfloor \frac{ir}{m} \rfloor - 2 \leq j \leq \lfloor \frac{ir}{m} \rfloor + 2$, then W^*_{ij} is assigned a random value sampled uniformly between 0 and 1, otherwise W^*_{ij} is set to 0; in this manner, we construct a data matrix that has values concentrated along the main diagonal.

these documents to try and recover W^*. Note that, in both of these cases, we are not specifically interested in the H matrix that the matrix factorization procedure will output.

As discussed in 2.2.2, there are two different regimes in which SGD can operate: one where new samples can be generated from an underlying distribution, and one where the same data sample is iterated through over multiple passes. The LDA model developed above allows us to simulate both of these regimes and compare the results. In the second case, which we will refer to as fixed-sample SGD, we sample p new documents, generating each one using a categorical distribution over the corresponding column of V^* (again, p should be chosen large enough to ensure each of the "true" documents is sampled at least once). This gives us a sample matrix $\tilde{V}$ which we can use as the input to Algorithm 1.

To simulate the first regime, where new data points can be generated on demand, we use a procedure we will refer to as generated-sample SGD. Here, at each iteration of Algorithm 1, we generate a brand new document $\tilde{V_i}$ by selecting one of the columns of V^* at random, and sampling using a categorical distribution over that column of V^* to form a new document. We then use this new sample to update W_k and the corresponding column $H_k(:, j)$ using Algorithm 1. This allows us to simulate how SGD will perform if it is provided with a new, unobserved document at each iteration.

5.3.2 Comparison of Two SGD Regimes for LDA

As outlined above, our goal is to understand how well SGD can recover the true words-to-topics matrix W^*. To this end, we consider a new objective function of the following form:

$$f(W_k) = ||\tilde{W}_k - W^*||_F, \tag{5.3}$$

where $\tilde{W}_k$ is a (possibly) scaled and permuted version of the matrix W_k to best match up with the target matrix W^*. Due to the non-uniqueness of the NMF problem, it is likely that the matrix W_k found by the algorithm will only resemble the true matrix W^* after permuting the columns and scaling appropriately. Given a W_k, we calculate $\tilde{W}_k$ as follows:

1. Denote $\tilde{W}_{k_1}, ..., \tilde{W}_{k_{r!}}$ as the $r!$ matrices formed by all possible column permutations of W_k
2. For a given permuted matrix $\tilde{W}_{k_a}$, scale each column j of $\tilde{W}_{k_a}$ by the factor $\frac{||W^*(:,j)||_1}{||\tilde{W}_{k_a}(:,j)||_1}$

3. Select
$$\tilde{W}_k = \underset{a}{\operatorname{argmin}}\{||\tilde{W}_{k_a} - W^*||,\ a \in [1, r!]\}.$$

In the following experiments[7], we vary three input parameters (Dirichlet parameters β, number of “true” documents n, and number of sampled documents p), with a fixed step-size α and a fixed total number of iterations T, to evaluate the effects that these parameters have on $f(W_T)$ for both SGD regimes. The results are summarized in Table 5.3.[8]

$f(W_T)$ **for Varying Input Parameters for 2 SGD Models**					
β**: Uniform** ($\beta_i = 0.5$, $\forall i \in [1, r]$)					
	$p = 500$	$p = 1000$	$p = 2000$	$p = 5000$	Generated-Sample
$n = 20$	1.12	1.07	1.00	1.00	1.02
$n = 50$	0.65	0.60	0.61	0.59	0.59
$n = 100$	0.59	1.03	1.00	0.54	0.98
$n = 200$	0.44	0.39	0.36	0.34	0.32
$n = 500$	0.75	0.75	0.74	0.68	0.72
β**: Mixed** ($\beta_i = [0.1, 0.2, 0.3, 0.2, 0.05, 0.05, 0.6, 0.4]$)					
	$p = 500$	$p = 1000$	$p = 2000$	$p = 5000$	Generated-Sample
$n = 20$	1.55	2.02	1.49	2.50	1.79
$n = 50$	1.23	0.98	1.09	0.75	1.11
$n = 100$	0.45	0.46	0.54	0.48	0.47
$n = 200$	0.48	0.35	0.37	0.33	0.32
$n = 500$	0.52	0.49	0.43	0.45	0.42
β**: Small** ($\beta_i = 0.05$, $\forall i \in [1, r]$)					
	$p = 500$	$p = 1000$	$p = 2000$	$p = 5000$	Generated Sample
$n = 20$	0.34	0.33	0.32	0.31	0.30
$n = 50$	0.34	0.32	0.32	0.32	0.31
$n = 100$	0.30	0.30	0.29	0.29	0.29
$n = 200$	0.55	0.55	0.54	0.53	0.53
$n = 500$	0.27	0.25	0.25	0.24	0.24

Table 5.3: $f(W_T)$ with varying β, n, p, and fixed $\alpha = 1 \times 10^{-4}$, $T = 10^6$, $m = 100$, $r = 8$

[7]We utilize code from [68] to generate Dirichlet random samples.

[8]Note that, since the number of samples p does not affect the generated-sample SGD model, we only list the generated-sample results once for each n and β. The results in each p column are for fixed-sample SGD.

We observe that, in general, the objective function $f(W_T)$ shrinks as n grows. This is not surprising, as an increase in n corresponds to an increase in available "true" data for the algorithm, which should lead to an increase in accuracy. We note that, as p increases, there are usually marginal improvements in $f(W_T)$. Again, this is not too surprising, as we would expect the algorithm to perform better with more available data; however, since the data is generated from the same n "true" data points, we would expect there to be a threshold where increasing p no longer leads to an improvement in the objective value. We also observe that the algorithm achieves slightly better accuracy results in the case where the β parameters are smaller. In the LDA model, smaller β parameters are associated with data that is drawn more from the extreme points of the underlying topic distribution simplex as opposed to the center; thus, it seems reasonable that lower β values lead to topics that are easier to discriminate between, and higher accuracy.

We also observe that the performance between the fixed-sample and generated-sample SGD models is very similar. This suggests that, over the same number of iterations, fixed-sample and generated-sample SGD both reach a similar level of accuracy. This is reasonable, as both fixed-sample and generated-sample SGD perform the same underlying update step at each iteration. This suggests that the decision of whether to use fixed-sample or generated-sample SGD can be made based on which application makes the most sense for the problem at hand, without having to worry about limiting the potential performance based on which approach is chosen.

Chapter 6

Conclusion

In this thesis, we studied the use of stochastic gradient descent to solve matrix factorization problems, with a specific interest in the nonnegative matrix factorization problem. We proved three results for the low-rank matrix factorization problem: two results on the convergence of standard Gradient descent, and an additional result for the stochastic case (where a zero-loss solution was known to exist). In all three cases, the algorithm required careful initialization near the minimum-loss solution in order to guarantee these results.

In addition to the above proved results, we developed a heuristic for how the SGD algorithm should behave where no zero-loss solution exists. This heuristic was drawn from known results for SGD for strongly-convex objective functions (which the matrix factorization problem is not); however, numerical experiments demonstrated that this heuristic predicted the performance of SGD quite well.

While all of these results were for the unconstrained low-rank matrix factorization problem, further numerical experiments suggested that the same algorithm with an additional projection step can be applied to solve the NMF problem, with very similar results. This suggests that, by extending the results from this thesis to the projected case, there may be provable performance guarantees for applying SGD to the NMF problem that can be found.

Additional numerical experiments were conducted to study the application of SGD and matrix factorization application to machine learning problems, with a specific interest in how matrix factorization can be used to classify unobserved data. These experiments show that SGD without projection can attain a classification accuracy that is competitive with that of existing matrix factorization algorithms in both the unconstrained and the nonnegative case. Additional experiments suggested that this approach can be used as a dimensionality reduction tool that can improve the performance of unsupervised clustering algorithms over large data sets.

A final set of experiments conducted matrix factorization of the NMF problem assuming the Latent Dirichlet Allocation model for topic classification, where a Dirichlet prior distribution was assumed for generating new documents. This set-up allowed SGD to be applied in two separate regimes: one with a large fixed sample of training documents, and one where new documents were generated at each training iteration. The results suggest that SGD is an effective tool for recovering the hidden words-to-topics matrix in both of these regimes.

The results from this thesis could be extended through future work in a number of ways. The three proved results can all be extended to weaken the condition on initializing close to the minimizer, and the heuristic provided for the performance of SGD in the non-zero-loss solution case should be developed further to provide some provable guarantees. This, combined with the addition of the projection step into the analysis, will help complete the picture for understanding the performance of SGD on the NMF problem.

An additional avenue of future work is to expand on the numerical results evaluating how the NMF solutions found by SGD can be used as machine learning classifiers. Specifically, further work should seek to understand where using SGD for this problem can provide advantages that other algorithms may not. This could include opportunities for parallelization, or utilizing SGD specifically to handle newly generated training data (as in the LDA model).

Overall, this work suggests that the unique properties of SGD make it an intriguing approach for solving large scale matrix factorization problems. Given the effectiveness of SGD as a training algorithm in many other data science or deep learning problems, the application of this algorithm to other challenging optimization problems continues to merit further study.

References

[1] Anima Anandkumar, Dean P Foster, Daniel J Hsu, Sham M Kakade, and Yi-Kai Liu. A spectral algorithm for latent dirichlet allocation. In *Advances in Neural Information Processing Systems*, pages 917–925, 2012.

[2] Sanjeev Arora, Rong Ge, and Ankur Moitra. Learning topic models–going beyond svd. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 1–10. IEEE, 2012.

[3] A. Berman and R.J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1994.

[4] Michael W Berry, Murray Browne, Amy N Langville, V Paul Pauca, and Robert J Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational statistics & data analysis*, 52(1):155–173, 2007.

[5] Dimitri P Bertsekas and John N Tsitsiklis. Gradient convergence in gradient methods with errors. *SIAM Journal on Optimization*, 10(3):627–642, 2000.

[6] José M Bioucas-Dias, Antonio Plaza, Nicolas Dobigeon, Mario Parente, Qian Du, Paul Gader, and Jocelyn Chanussot. Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches. *IEEE journal of selected topics in applied earth observations and remote sensing*, 5(2):354–379, 2012.

[7] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[8] Avrim Blum, John Hopcroft, and Ravi Kannan. *Foundations of Data Science*. June 2017.

[9] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168, 2008.

[10] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[11] Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge university press, 2004.

[12] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[13] Aeron M Buchanan and Andrew W Fitzgibbon. Damped newton algorithms for matrix factorization with missing data. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 316–322. IEEE, 2005.

[14] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717, 2009.

[15] Augustin Cauchy. Méthode générale pour la résolution des systemes déquations simultanées. 1847.

[16] Yuejie Chi, Yue M Lu, and Yuxin Chen. Nonconvex optimization meets low-rank matrix factorization: An overview. *arXiv preprint arXiv:1809.09573*, 2018.

[17] Andrzej Cichocki, Rafal Zdunek, Anh Huy Phan, and Shun-ichi Amari. *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation.* John Wiley & Sons, 2009.

[18] Damek Davis and Dmitriy Drusvyatskiy. Stochastic subgradient method converges at the rate $o(k^{-1/4})$ on weakly convex functions. *arXiv preprint arXiv:1802.02988*, 2018.

[19] Aymeric Dieuleveut, Alain Durmus, and Francis Bach. Bridging the gap between constant step size stochastic gradient descent and markov chains. *arXiv preprint arXiv:1707.06386*, 2017.

[20] Chris Ding, Xiaofeng He, and Horst D Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 606–610. SIAM, 2005.

[21] David Donoho and Victoria Stodden. When does non-negative matrix factorization give a correct decomposition into parts? In *Advances in neural information processing systems*, pages 1141–1148, 2004.

[22] J. L. Doob. *Stochastic Processes*. New York: Wiley, 1953.

[23] Dmitriy Drusvyatskiy and Courtney Paquette. Efficiency of minimizing compositions of convex functions and smooth maps. *Mathematical Programming*, pages 1–56, 2018.

[24] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, Sep 1936.

[25] Yonina C Eldar and Gitta Kutyniok. *Compressed sensing: theory and applications*. Cambridge University Press, 2012.

[26] N Benjamin Erichson, Ariana Mendible, Sophie Wihlborn, and J Nathan Kutz. Randomized nonnegative matrix factorization. *Pattern Recognition Letters*, 104:1–7, 2018.

[27] A. Eriksson and A. van den Hengel. Efficient computation of robust low-rank matrix approximations in the presence of missing data using the l1 norm. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 771–778, June 2010.

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

[29] Nicolas Gillis. Sparse and unique nonnegative matrix factorization through data preprocessing. *Journal of Machine Learning Research*, 13(Nov):3349–3386, 2012.

[30] Nicolas Gillis. The Why and How of Nonnegative Matrix Factorization. *arXiv e-prints*, page arXiv:1401.5226, Jan 2014.

[31] Nicolas Gillis and François Glineur. Low-rank matrix approximation with weights or missing data is np-hard. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1149–1165, 2011.

[32] Nicolas Gillis and Stephen A. Vavasis. Fast and Robust Recursive Algorithms for Separable Nonnegative Matrix Factorization. *arXiv e-prints*, page arXiv:1208.1237, Aug 2012.

[33] Nicolas Gillis and Stephen A. Vavasis. On the Complexity of Robust PCA and L-1-norm Low-Rank Matrix Approximation. *arXiv e-prints*, page arXiv:1509.09236, Sep 2015.

[34] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, third edition, 1996.

[35] N. Guan, D. Tao, Z. Luo, and B. Yuan. Online nonnegative matrix factorization with robust stochastic approximation. *IEEE Transactions on Neural Networks and Learning Systems*, 23(7):1087–1099, July 2012.

[36] David Guillamet and Jordi Vitrià. Non-negative matrix factorization for face recognition. In *Catalonian Conference on Artificial Intelligence*, pages 336–344. Springer, 2002.

[37] Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. *arXiv preprint arXiv:1509.01240*, 2015.

[38] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations.* Chapman & Hall/CRC, 2015.

[39] Prateek Jain and Purushottam Kar. Non-convex optimization for machine learning. *Foundations and Trends® in Machine Learning*, 10(3-4):142–336, 2017.

[40] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[41] Qifa Ke and Takeo Kanade. Robust l/sub 1/norm factorization in the presence of outliers and missing data by alternative convex programming. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 739–746. IEEE, 2005.

[42] Hyunsoo Kim and Haesun Park. Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. *SIAM journal on matrix analysis and applications*, 30(2):713–730, 2008.

[43] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[44] Hans Laurberg, Mads Græsbøll Christensen, Mark D Plumbley, Lars Kai Hansen, and Søren Holdt Jensen. Theorems on positive data: On the uniqueness of nmf. *Computational intelligence and neuroscience*, 2008, 2008.

[45] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[46] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788, 1999.

[47] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2001.

[48] Tao Li and Cha-charis Ding. Nonnegative matrix factorizations for clustering: A survey. In *Data Clustering*, pages 149–176. Chapman and Hall/CRC, 2013.

[49] Chih-Jen Lin. Projected gradient methods for nonnegative matrix factorization. *Neural computation*, 19(10):2756–2779, 2007.

[50] Junhong Lin and Lorenzo Rosasco. Optimal learning for multi-pass stochastic gradient methods. In *Advances in Neural Information Processing Systems*, pages 4556–4564, 2016.

[51] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[52] Andrew Mass and Sameep Tandon. Stanford unsupervised feature learning and deep learning tutorial. https://github.com/amaas/stanford_dl_ex, 2013.

[53] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[54] Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848, 2002.

[55] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[56] Takayuki Okatani and Koichiro Deguchi. On the wiberg algorithm for matrix factorization in the presence of missing components. *International Journal of Computer Vision*, 72(3):329–337, May 2007.

[57] Pentti Paatero and Unto Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994.

[58] Christos H Papadimitriou, Prabhakar Raghavan, Hisao Tamaki, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. *Journal of Computer and System Sciences*, 61(2):217–235, 2000.

[59] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[60] Georg Ch Pflug. Stochastic minimization with constant step-size: asymptotic laws. *SIAM Journal on Control and Optimization*, 24(4):655–666, 1986.

[61] Ilya Razenshteyn, Zhao Song, and David P Woodruff. Weighted low rank approximations with provable guarantees. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 250–263. ACM, 2016.

[62] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2011.

[63] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[64] Farial Shahnaz, Michael W Berry, V Paul Pauca, and Robert J Plemmons. Document clustering using nonnegative matrix factorization. *Information Processing & Management*, 42(2):373–386, 2006.

[65] Shai Shalev-Schwartz and Shai Ben-David. *Understanding Machine Learning.* Cambridge University Press, 2014.

[66] Romy Shioda and Levent Tunçel. Clustering via minimum volume ellipsoids. *Comput. Optim. Appl.*, 37(3):247–295, July 2007.

[67] Nathan Srebro and Tommi Jaakkola. Weighted low-rank approximations. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 720–727, 2003.

[68] Mark Steyvers and Tom Griffiths. Probabilistic topic models. *Handbook of latent semantic analysis*, 427(7):424–440, 2007.

[69] L. Thomas. Rank factorization of nonnegative matrices (a. berman). *SIAM Review*, 16(3):393–394, 1974.

[70] Vladimir Vapnik. Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838, 1992.

[71] Stephen A Vavasis. On the complexity of nonnegative matrix factorization. *SIAM Journal on Optimization*, 20(3):1364–1377, 2009.

[72] Jim Jing-Yan Wang, Xiaolei Wang, and Xin Gao. Non-negative matrix factorization by maximizing correntropy for cancer clustering. *BMC Bioinformatics*, 14(1):107, Mar 2013.

[73] Ming Yuan, Ali Ekici, Zhaosong Lu, and Renato Monteiro. Dimension reduction and coefficient estimation in multivariate linear regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(3):329–346, 2007.

[74] Qian Zhao, Deyu Meng, Zongben Xu, Wangmeng Zuo, and Yan Yan. $l\_\{1\}$-norm low-rank matrix factorization by variational bayesian method. *IEEE transactions on neural networks and learning systems*, 26(4):825–839, 2015.

[75] Zhihui Zhu, Qiuwei Li, Gongguo Tang, and Michael B Wakin. The global optimization geometry of low-rank matrix optimization. *arXiv preprint arXiv:1703.01256*, 2017.

[76] Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. *Journal of computational and graphical statistics*, 15(2):265–286, 2006.