OPTIMIZATION PROBLEMS WITH LOW-DIMENSIONAL TENSOR STRUCTURE

By

SOO MIN KWON

A thesis submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Anand D. Sarwate

and approved by

_____

_____

_____

New Brunswick, New Jersey

May, 2022

ABSTRACT OF THE THESIS

Optimization Problems with Low-Dimensional Tensor Structure

By Soo Min Kwon

Thesis Director:

Anand D. Sarwate

In this thesis, we consider optimization problems that involve statistically estimating signals from tensor data. We observe that modelling the signals as tensors using tensor factorizations increases the accuracy of more closely estimating the true signal. We empirically show that this is a result of two reasons: (i) it reduces the number of parameters that need to be estimated, hinting at a decrease in sample complexity, and (ii) it allows us to take advantage of the low-rank property that many high-dimensional tensor data samples possess. We also show that while there exists a tradeoff between the accuracy of the reconstructed signal and the ranks according to the tensor decomposition, there often exists a rank that improves performance in sample-starved settings. We discuss these tradeoffs and develop algorithms for two of these applications, classification and phase retrieval, and demonstrate the effectiveness of our algorithms under several different settings and performance metrics.

Acknowledgements

Foremost, I would like to thank my parents and grandparents for supporting me throughout my college journey. There is no way I can ever repay my parents for the countless sacrifices they have made for me. They have always been supportive in all of my decisions, and I cannot thank them enough for that. I would also like to thank my partner Suyeon for her love, patience and encouragement throughout the past four years. She was the first person I ever told that I wanted pursue graduate school, and she has been nothing but supportive.

I am also thankful for my excellent committee. I asked Professor Shirin Jalali and Professor Yuqian Zhang very last minute to be a part of my committee, and they have been very patient and responsive. They asked intriguing questions during my defense that helped me gain a new perspective on how I can further my research. I will try my best to keep these questions in mind as I pursue more research during my PhD. I would also like to extend my gratitude to Professor Waheed U. Bajwa and Professor Narayan Mandayam for writing me recommendation letters for graduate school.

I have also been very fortunate to have interacted with many talented PhD students: Sijie Xiong and Hafiz Imtiaz. My first ever research project was with Hafiz, and he was very patient with me while he taught me the basics of differential privacy. I was also not a very good programmer at the time, but he was very patient with me as he taught me the basics. During my undergraduate research, Sijie also helped me a lot with programming. I would often send her code that would not work and asked her to debug it for me. She patiently taught me how to debug and write code, and my first project would not have been completed without her help.

Lastly, I would like to thank my role model and advisor Professor Anand D. Sarwate. He has taught me everything I know about research, all the way from reading a paper to

writing a paper. He has taught me the value of mathematical rigor, how to effectively present research and how to think about research problems. Without his help, I doubt I would be where I am today. I will always be thankful for his patience, encouragement, and guidance.

Dedication

To my parents and grandparents.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Many fundamental problems in machine learning and signal processing are formulated as either convex or non-convex optimization problems, examples including dictionary learning [1], phase retrieval [2, 3] and blind deconvolution [4]. The objective of these optimization problems is to estimate or recover signals given data or functions of the data. One of the most common properties that these optimization problems share is that they are generally formulated for vector-valued data. While problems with structure arise in many applications, it may not be appropriate for data that intrinsically have many dimensions. To use vector-based formulations for multidimensional data samples, which we call tensors, we need to vectorize the data. However, the vectorization of tensors destroys their spatial and temporal structure, a structure that many tensors possess in applications such as medical and hyperspectral imaging [5, 6]. Additionally, for high-dimensional tensors, vectorization yields a problem that is either ill-posed or requires significantly more samples, making the recovery of the signal (or signals) more difficult.

In this thesis, we tackle these challenges in two applications, statistical machine learning and phase retrieval, by imposing a tensor factorization on the parameters of the optimization problem. We observe that imposing this structure leverages the low-rank property that many tensor data samples have, which increases performance across various metrics. We also empirically show that this structure decreases the sample complexity needed for accurate recovery of the signals by decreasing the number of parameters that need to be estimated.

1.2 Contributions and Organization

Contributions. The main contributions of this thesis are the empirical results that demonstrate the effectiveness of imposing a tensor factorization on the parameters of two optimization problems: statistical machine learning and low-rank phase retrieval. We extensively show with several experiments that our algorithms outperforms existing methods by leveraging the low-rank property that many high-dimensional tensor data samples have. For each of our proposed algorithms, we provide pseudo-code and details on its implementation. The code to reproduce the results presented in this thesis is available at

`https://github.com/soominkwon`.

Organization. The rest of this thesis is organized as follows. In Chapter 2, we introduce the notation that we will use throughout this thesis and discuss some relevant tensor algebra. Those who are interested in the algorithms and results can skip to Chapters 3 and 4. In Chapter 3, we present our first contribution [7]. We modify two popular machine learning classification algorithms, Logistic Regression and Linear Support Vector Machine, for tensor data and discuss them in this chapter. This chapter also briefly reviews the main ideas behind these two algorithms, which may be beneficial in understanding the algorithms modified to fit tensor data. In Chapter 4, we present our second contribution, which is changing the modelling assumption of data to tensors rather than as matrices as previously done in the literature for low-rank phase retrieval [8]. We show that modelling the signals as tensors more accurately recovers the signals in both the under and over-determined settings when the values of the ranks corresponding to the tensor factorization are chosen appropriately.

Chapter 2

Notation and Preliminaries

In this chapter, we begin by specifying the notation that we will be using throughout this thesis. We then review some preliminaries regarding some vector and matrix products as well as some tensor algebra.

2.1 Notation

We denote scalars with lowercase letters (e.g. x), vectors with bold lowercase letters (e.g. $\mathbf{x}$), matrices with bold uppercase letters (e.g. $\mathbf{X}$), and tensors with underlined, bold uppercase letters (e.g. $\underline{\mathbf{X}}$). We denote the n-th column of the matrix $\mathbf{X}$ as $\mathbf{x}_n$. Similarly, we denote the n-th frontal slice of the tensor $\underline{\mathbf{X}}$ as $\mathbf{X}_n$. We denote the identity matrix as $\mathbf{I_n}$, where n is the dimensions of the matrix. We use $\|\cdot\|_2$ and $\|\cdot\|_\mathsf{F}$ for the Euclidean (or ℓ_2) norm of a vector or spectral norm of a matrix and the Frobenius norm, respectively. We use $\mathbf{x}^*$ to denote the conjugate (or Hermitian) transpose of the vector $\mathbf{x}$. Lastly, we denote the inner-product between two vectors $\mathbf{a}$ and $\mathbf{b}$ as $\langle \mathbf{a}, \mathbf{b} \rangle$.

2.2 Vector and Matrix Products

Kronecker Product. Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{j \times k}$ be two matrices with entries

$$\mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} b_{11} & \dots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{j1} & \dots & b_{jk} \end{bmatrix}.$$

The Kronecker product of $\mathbf{A}$ and $\mathbf{B}$, denoted as $\mathbf{A} \otimes \mathbf{B}$, is computed by multiplying every element in $\mathbf{A}$ by the matrix $\mathbf{B}$. Hence, $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{mj \times nk}$ is a matrix with entries

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \dots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \dots & a_{mn}\mathbf{B} \end{bmatrix}.$$

Khatri-Rao Product. Let $\mathbf{C} \in \mathbb{R}^{m \times n}$ and $\mathbf{D} \in \mathbb{R}^{p \times n}$ be two matrices with columns

$$\mathbf{C} = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_n \end{bmatrix}, \ \mathbf{D} = \begin{bmatrix} \mathbf{d}_1 & \mathbf{d}_2 & \dots & \mathbf{d}_n \end{bmatrix}.$$

The Khatri-Rao product of matrices $\mathbf{C}$ and $\mathbf{D}$, denoted by $\mathbf{C} \odot \mathbf{D}$, is computed by taking the column-wise Kronecker product. Hence, $\mathbf{C} \odot \mathbf{D} \in \mathbb{R}^{mp \times n}$ is a matrix with columns

$$\mathbf{C} \odot \mathbf{D} = \begin{bmatrix} \mathbf{c}_1 \otimes \mathbf{d}_1 & \mathbf{c}_2 \otimes \mathbf{d}_2 & \dots & \mathbf{c}_n \otimes \mathbf{d}_n \end{bmatrix}.$$

Note that if $\mathbf{C}$ and $\mathbf{D}$ were instead column vectors (i.e. $n = 1$), then the Khatri-Rao product is equivalent to the Kronecker product (i.e. $\mathbf{c} \odot \mathbf{d} = \mathbf{c} \otimes \mathbf{d}$).

2.3 A Primer on Tensor Algebra

For a comprehensive review on tensor algebra, we refer the reader to the survey paper by Kolda and Bader [9]. We first review two ways in which we can reorder tensors and then introduce two ways in which we can decompose tensors that we will use later in this thesis.

2.3.1 Tensor Reorderings

To better explain and visualize tensor reorderings, we will consider a third-order tensor of dimensions $\underline{\mathbf{X}} \in \mathbb{R}^{3 \times 3 \times 2}$, with the two frontal slices having entries

$$\mathbf{X}_1 = \begin{bmatrix} 2 & 8 & 14 \\ 4 & 10 & 16 \\ 6 & 12 & 18 \end{bmatrix}, \ \mathbf{X}_2 = \begin{bmatrix} 1 & 7 & 13 \\ 3 & 9 & 15 \\ 5 & 11 & 17 \end{bmatrix}.$$

Vectorization. We can create a vector from any matrix or tensor by stacking the row or column elements into a row or column vector, respectively. For example, vectorizing the tensor $\underline{\mathbf{X}}$ by its columns would yield the column vector

$$\text{vec}(\underline{\mathbf{X}}) = \begin{bmatrix} 2 \\ 4 \\ \vdots \\ 15 \\ 17 \end{bmatrix},$$

where we stack the columns from the first frontal slice, $\mathbf{X}_1$, and then the second frontal slice, $\mathbf{X}_2$. The dimensions of the resulting vector would be $\mathbf{x} \in \mathbb{R}^{18}$. Throughout this thesis, we will use $\text{vec}(\cdot)$ to denote vectorization as an operator and will assume that this operator performs column-wise stacking unless stated otherwise.

A Property of the Vec Operator. One important property of the $\text{vec}(\cdot)$ operator is that given three matrices $\mathbf{A} \in \mathbb{R}^{q \times n_1}$, $\mathbf{B} \in \mathbb{R}^{n_1 \times n_2}$, and $\mathbf{C} \in \mathbb{R}^{n_2 \times r}$, the vectorization of the product of these three matrices yield

$$\text{vec}(\mathbf{ABC}) = (\mathbf{C}^\top \otimes \mathbf{A})\,\text{vec}(\mathbf{B}). \tag{2.1}$$

This property is useful for problems where one needs to isolate one of the matrices from a product of matrices. Note that if you needed to isolate the matrix $\mathbf{A}$ instead of the matrix $\mathbf{B}$, one could multiply the left-hand side of the product with an identity matrix and then use this property to obtain

$$\text{vec}(\mathbf{I_q ABC}) = ((\mathbf{BC})^\top \otimes \mathbf{I_q})\,\text{vec}(\mathbf{A}). \tag{2.2}$$

Matricization. We can also create a matrix from a tensor by stacking the columns from the n-th mode of the tensor. For example, the three matricizations of the tensor $\underline{\mathbf{X}}$ would be

$$\mathcal{M}_1(\underline{\mathbf{X}}) = \begin{bmatrix} 2 & 8 & 14 & 1 & 7 & 13 \\ 4 & 10 & 16 & 3 & 9 & 15 \\ 6 & 12 & 18 & 5 & 11 & 17 \end{bmatrix},$$

$$\mathcal{M}_2(\underline{\mathbf{X}}) = \begin{bmatrix} 2 & 4 & 6 & 1 & 3 & 5 \\ 8 & 10 & 12 & 7 & 9 & 11 \\ 14 & 16 & 18 & 13 & 15 & 17 \end{bmatrix},$$

$$\mathcal{M}_3(\underline{\mathbf{X}}) = \begin{bmatrix} 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 & 18 \\ 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 \end{bmatrix},$$

where $\mathcal{M}_n(\cdot)$ denotes the n-th mode matricization operation. Note that since the tensor $\underline{\mathbf{X}}$ is a third-order tensor, there are three ways in which we can matricize $\underline{\mathbf{X}}$, one for each mode.

2.3.2 Tensor Decompositions

Many data matrices that arise pervasively throughout data science exhibit an approximate (or exact) low-rank structure [10]. Mathematically, this means that we can decompose a matrix $\mathbf{Y}$ into a bi-linear form such that the error defined as

$$\min_{\mathbf{U},\mathbf{V}} \|\mathbf{Y} - \mathbf{U}\mathbf{V}^\top\|_{\mathsf{F}}^2 \tag{2.3}$$

is small. Tensors also have low-rank structures, but the decomposition for tensors is mathematically defined a bit differently. We discuss the two ways in which we can decompose tensors in this section.

CANDECOMP/PARAFAC (CP) Decomposition. The CP (or Polyadic) decomposition is a straightforward extension of matrix factorization to tensors. The objective of the CP decomposition is to express a tensor as the sum of component rank-one tensors, i.e. vectors, as depicted in Figure 2.1. For example, consider a third-order tensor $\underline{\mathbf{X}} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$. The CP decomposition of the tensor $\underline{\mathbf{X}}$ would yield

$$\underline{\mathbf{X}} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r, \tag{2.4}$$

where $\mathbf{a}_r \in \mathbb{R}^{n_1}$, $\mathbf{b}_r \in \mathbb{R}^{n_2}$, $\mathbf{c}_r \in \mathbb{R}^{n_3}$, R is the rank of the tensor, and $\mathbf{a}_r \circ \mathbf{b}_r$ represents the outer product of vectors $\mathbf{a}_r$ and $\mathbf{b}_r$. The CP decomposition as an optimization problem would mean finding the best $\hat{\underline{\mathbf{X}}}$ given $\underline{\mathbf{X}}$ such that the error

$$\min_{\hat{\underline{\mathbf{X}}}} \|\underline{\mathbf{X}} - \hat{\underline{\mathbf{X}}}\|, \quad \text{subject to} \quad \hat{\underline{\mathbf{X}}} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r, \tag{2.5}$$

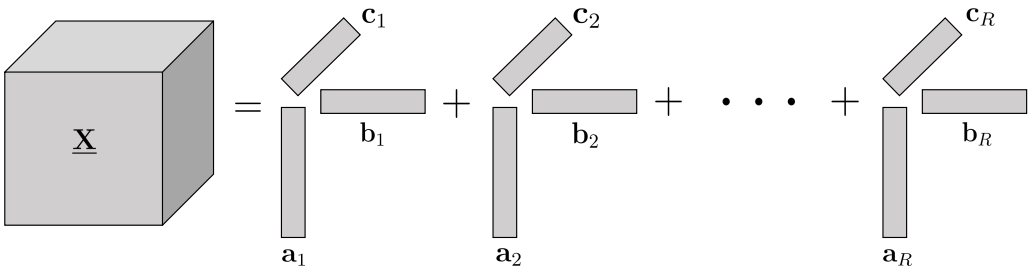


Figure 2.1: Depiction of the CANDECOMP/PARAFAC (CP) decomposition. The CP decomposition factorizes a tensor into a sum of rank-one components.

is small. We will often refer to the collection of the rank-one tensors as the factor matrices of the tensor. For example, the factor matrix $\mathbf{A} \in \mathbb{R}^{n_1 \times R}$ would be

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_R \end{bmatrix}.$$

A Property of the CP Decomposition. Using the definition of the factor matrices, we can formulate some useful properties of the CP decomposition. Given the CP decomposition of a third-order tensor $\underline{\mathbf{X}}$ with factor matrices $\mathbf{A} \in \mathbb{R}^{n_1 \times R}$, $\mathbf{B} \in \mathbb{R}^{n_2 \times R}$, and $\mathbf{C} \in \mathbb{R}^{n_3 \times R}$, we have the following relations:

$$\mathcal{M}_1(\underline{\mathbf{X}}) = (\mathbf{C} \odot \mathbf{B})\mathbf{A}^\top \tag{2.6}$$

$$\mathcal{M}_2(\underline{\mathbf{X}}) = (\mathbf{C} \odot \mathbf{A})\mathbf{B}^\top \tag{2.7}$$

$$\mathcal{M}_3(\underline{\mathbf{X}}) = (\mathbf{B} \odot \mathbf{A})\mathbf{C}^\top. \tag{2.8}$$

We would like to note that these relationships easily generalize to n-dimensional tensors [9].

Tucker Decomposition. The Tucker decomposition can be viewed as a higher-order analogue of the Singular Value Decomposition (SVD) [9]. It decomposes a tensor into a core tensor, multiplied by factor matrices along each mode. For example, given a third-order tensor $\underline{\mathbf{X}} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, the Tucker decomposition of $\underline{\mathbf{X}}$ would yield

$$\underline{\mathbf{X}} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr}\, \mathbf{a}_p \circ \mathbf{b}_q \circ \mathbf{c}_r \tag{2.9}$$

$$= \underline{\mathbf{G}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}, \tag{2.10}$$

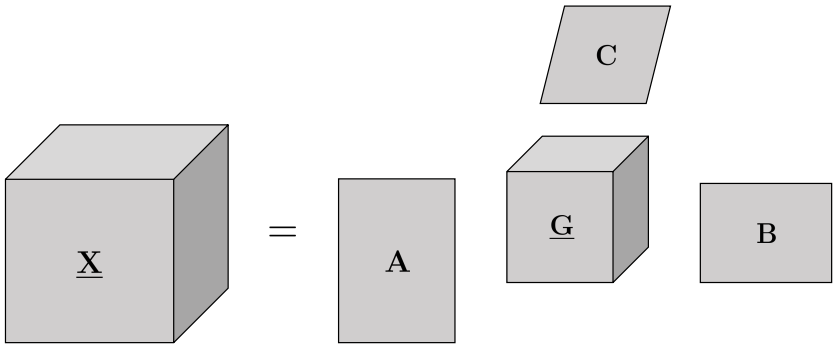


Figure 2.2: Depiction of the Tucker decomposition. The Tucker decomposition factorizes a tensor into a product of a smaller tensor and factor matrices in each mode.

where $\underline{\mathbf{G}} \in \mathbb{R}^{P \times Q \times R}$ is the core tensor, and $\mathbf{A} \in \mathbb{R}^{n_1 \times P}$, $\mathbf{B} \in \mathbb{R}^{n_2 \times Q}$, and $\mathbf{C} \in \mathbb{R}^{n_3 \times R}$ are the factor matrices. One interesting (and also quite useful) fact about the Tucker decomposition is that each factor matrix can have different ranks, i.e. $P \neq Q \neq R$, whereas the factor matrices for the CP decomposition must have the same ranks. This flexibility often makes the Tucker decomposition a more useful factorization, which we will see later in this thesis.

A Property of the Tucker Decomposition. Similar to the CP decomposition, we can formulate some useful relationships of the Tucker decomposition in terms of the matricized forms of the tensor. Given the Tucker decomposition of a third-order tensor $\underline{\mathbf{X}}$ with core tensor $\underline{\mathbf{G}} \in \mathbb{R}^{P \times Q \times R}$ and factor matrices $\mathbf{A} \in \mathbb{R}^{n_1 \times P}$, $\mathbf{B} \in \mathbb{R}^{n_2 \times Q}$, and $\mathbf{C} \in \mathbb{R}^{n_3 \times R}$, we have the following relations:

$$\mathcal{M}_1(\underline{\mathbf{X}}) = \mathbf{A}\,\mathcal{M}_1(\underline{\mathbf{G}})(\mathbf{C} \otimes \mathbf{B})^\top \tag{2.11}$$

$$\mathcal{M}_2(\underline{\mathbf{X}}) = \mathbf{B}\,\mathcal{M}_2(\underline{\mathbf{G}})(\mathbf{C} \otimes \mathbf{A})^\top \tag{2.12}$$

$$\mathcal{M}_3(\underline{\mathbf{X}}) = \mathbf{C}\,\mathcal{M}_3(\underline{\mathbf{G}})(\mathbf{B} \otimes \mathbf{A})^\top. \tag{2.13}$$

We would like to note again that these relationships easily generalize to n-dimensional tensors [9].

Chapter 3

Machine Learning with Tensor Factorizations

3.1 Introduction

Machine learning refers to a set of statistical algorithms that we use to predict, classify, and cluster data. These algorithms are widely used in many applications, examples including disease prediction [11, 12], speech recognition [13], and product recommendation [14, 15]. The objective of many of these statistical methods is to estimate coefficients (or predictors) that best capture the relationship amongst the data samples. This relationship is quantified by a loss function, where each algorithm (generally) has its own unique function. When the loss function involves some linear combination of the weights and data points, we refer to the algorithm as a *linear* machine learning algorithm. The linearity of these functions implicitly require each data sample to be a vector, and otherwise need the data sample to be vectorized to fit these algorithms. In the literature, there have been a lot of advancements that study the behavior of these linear algorithms [16], as well as its sample complexity [17], and its performance.

Many applications of machine learning involve analyzing the relationship of data samples that have intrinsically many dimensions [5, 18–20]. The objective of these analyses is the same as that of traditional machine learning – we want to establish a close association between the multidimensional (tensor) data samples and their outcomes. However, as previously mentioned, most machine learning algorithms (if not all) are formulated for vector-valued data. To fit these algorithms for tensor data, we need to vectorize the data, which makes studying tensor data more challenging for several reasons. Firstly, tensor-valued data in many real-world domains are usually very high-dimensional. For example, in medical imaging, it is common to see tensor data samples that are of dimensions $128 \times 128 \times 128$ or even greater. If we vectorize these samples to fit

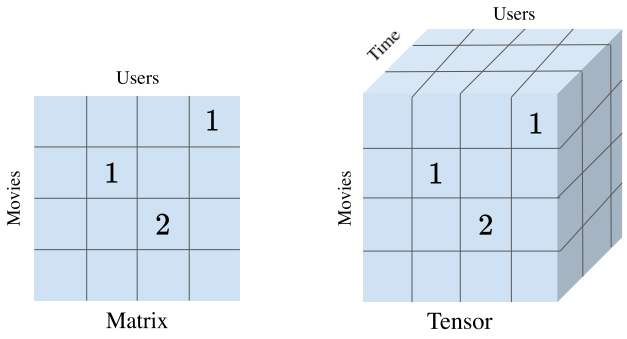


Figure 3.1: Example of modeling observations: left – matrix, right – tensor

the traditional algorithms, this would mean estimating $128^3 = 2097152$ parameters! As the number of dimensions increases, we not only need significantly more data samples, but computation also becomes intractable. Secondly, and perhaps more importantly, the vectorization of tensor data destroys its spatial structure, which can (and should) be leveraged for accurate analysis. For example, consider a tensor as shown in Figure 3.1, where users' ratings of different movies are measured over time. If we wanted to use machine learning to make movie recommendations to other users using this data, we would need to vectorize the data, which reorders the data in such a way that a specific user's movie ratings over time no longer matches. Clearly, this incorrect reordering could result in poor recommendations, making vectorization an inefficient modelling scheme for tensor data.

To solve these challenges, we propose two algorithms that impose a tensor structure on the weights of two machine learning algorithms used for classification. More specifically, we assume that the weights of Logistic Regression and Support Vector Machine (SVM) admit a CANDECOMP/PARAFAC (CP) decomposition, which allows us to reduce the number of parameters that need to be estimated and exploits the structure of the data. Unlike their traditional counterparts that use vanilla gradient descent for optimization, these proposed algorithms, CP-Logistic Regression and CP-SVM, use an alternating minimization method to update the factor matrices (weights). Under several synthetic settings, we demonstrate the effectiveness of our algorithms for increasing sample sizes and ranks. Since imposing this CP structure on the weights assume that

the weights (and consequently, the data) are low-rank, we simulate an exact and an approximate low-rank setting and record their performances. In all of these settings, we observe that CP-Logistic Regression and CP-SVM outperforms the traditional machine learning algorithms for several different metrics.

3.2 Related Work

In the literature, there are several related works that exploits the low-rank structure of the data by imposing some factorization technique [21–24]. We were heavily inspired by the results of Zhou et al. [21], where the authors proposed a family of tensor regression models that also imposed a CP decomposition on the weights. Their approach included a mixture of maximum likelihood and alternating minimization to estimate the parameters of the density of the exponential family and the factor matrices, respectively. Motivated by their results on medical data, we pursued in adopting their ideas for classification models with tensor data.

The work most closely related to the ideas presented in this chapter is the work by Tan et al [25]. Tan et al. proposed a logistic regression model that imposed a CP decomposition on the weights of this model. Their algorithm involved using alternating minimization with an ℓ_1 regularization term on the objective function to solve for sparse factor matrices. We were unaware of this work before we dove into looking at classification problems with tensor structure; we routed around this issue by answering questions that the authors did not consider, such as the behavior of these algorithms under approximate low-rank settings. We also showed empirical results using an ℓ_2 regularizer, and made our code publicly available on Github. Lastly, we also proposed an tensor-based model for Support Vector Machine (SVM), which Tan et al. does not consider in their work.

3.3 Unstructured Machine Learning

In this section, we give a quick primer on two linear machine learning classification algorithms: Logistic Regression and Support Vector Machines. We interchangeably refer to these algorithms as traditional, unstructured, or vectorized algorithms, as these methods do not assume any structure and vectorizes the data. We provide some intuition behind these machine learning algorithms and show how these algorithms are posed as optimization problems.

3.3.1 Logistic Regression

Logistic Regression is very similar to Linear Regression [26], where we consider a linear model of the form

$$y_i = \mathbf{w}^\top \mathbf{x}_i + b,\ i = 1, \ldots, n. \tag{3.1}$$

In Linear Regression, our objective is to find the weights, $\mathbf{w}$, and bias, b, such that we can accurately predict the dependent variables y_i given independent variables $\mathbf{x}_i$. The dependent and independent variables can take on any continuous value. However, Logistic Regression is a binary classification algorithm, where the dependent variable y_i can only take on discrete values $y_i \in \{0, 1\}$. We need to modify the linear model in such a way that we can predict discrete y_i values instead of continuous ones.

Instead of directly predicting discrete values $y_i \in \{0, 1\}$, Logistic Regression predicts the *probabilities* that a data sample $\mathbf{x}_i$ belongs to either 0 or 1. We do this by "regressing" on the log-odds of the probabilities rather than y_i itself. Formally, the linear model now becomes

$$\log\left(\frac{\mathbb{P}(y_i = 1)}{\mathbb{P}(y_i = 0)}\right) = \mathbf{w}^\top \mathbf{x}_i + b,\ i = 1, \ldots, n. \tag{3.2}$$

Rearranging equation (3.2) for $\mathbb{P}(y_i = 1)$ and $\mathbb{P}(y_i = 0)$, we obtain

$$\mathbb{P}(y_i = 1) = \sigma(\mathbf{w}^\top \mathbf{x_i} + b) \tag{3.3}$$

$$= \frac{1}{1 + \exp(-(\mathbf{w}^\top \mathbf{x_i} + b))} \tag{3.4}$$

$$\mathbb{P}(y_i = 0) = 1 - \mathbb{P}(y_i = 1) \tag{3.5}$$

$$= \frac{1}{1 + \exp(\mathbf{w}^\top \mathbf{x_i} + b)}, \tag{3.6}$$

where $\sigma(\cdot)$ is the sigmoid function. The objective of Logistic Regression is to estimate the predictors (or weights) $\mathbf{w}$ and bias b that maximizes these two probabilities. Once we obtain the weights and bias terms, we can make predictions by plugging in $\mathbf{x}_i$ into equation (3.3) and deciding $y_i = 1$ if $\mathbb{P}(y_i = 1) \geq \tau$ or $y_i = 0$ if $\mathbb{P}(y_i = 1) < \tau$, for some threshold τ.

The main challenge now is to find a way to estimate the predictors that maximizes the probabilities. We can do this by finding the weights that minimizes the *logistic loss* function

$$\ell(y_i, \mathbf{x}_i, \mathbf{w}, b) = \sum_{i=1}^{n} -y_i \log(\hat{y}_i) - (1 - y_i) \log(1 - \hat{y}_i), \tag{3.7}$$

where $\hat{y}_i = \sigma(\mathbf{w}^\top \mathbf{x_i} + b)$. Note that we can rewrite $\ell(\cdot)$ into a more compact form, as

$$\ell(y_i, \mathbf{x}_i, \mathbf{w}, b) = \sum_{i=1}^{n} \log(1 + \exp\left(-y_i(\mathbf{w}^\top \mathbf{x_i} + b)\right)). \tag{3.8}$$

Now, using equation (3.8) and given data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, we can formulate Logistic Regression as the optimization problem

$$\hat{\mathbf{w}}, \hat{b} = \underset{\mathbf{w}, b}{\operatorname{argmin}} \sum_{i=1}^{n} \log(1 + \exp\left(-y_i(\mathbf{w}^\top \mathbf{x_i} + b)\right)) + \lambda \|\mathbf{w}\|_2. \tag{3.9}$$

Since there is no closed-form solution for this optimization problem, we need to use techniques such as gradient descent to estimate these parameters. Note that the term $\lambda \|\mathbf{w}\|_2$ in the objective function is a regularizer that induces sparsity in $\mathbf{w}$.

3.3.2 Support Vector Machines

Support Vector Machine (SVM) is both a linear and non-linear classification algorithm that predicts whether a data sample $\mathbf{x}_i$ belongs to $y_i \in \{-1, +1\}$. In this chapter,

we will focus on linear SVM, where we again consider the hyperplane $y_i = \mathbf{w}^\top \mathbf{x}_i + b$. Recall that in Logistic Regression, we estimated the predictors $\mathbf{w}$ and b by minimizing the logistic loss function. SVM minimizes a loss function called the *hinge loss* function, where we penalize according to the distance of the data sample $\mathbf{x}_i$ to the hyperplane. Mathematically, the hinge loss function for linear SVM is defined as

$$\ell(y_i, \mathbf{x}_i, \mathbf{w}, b) = \max[0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)] \tag{3.10}$$

The intuition behind this loss function is that we want to find the predictors that best separates the two classes of data, while making minimal violations. Note that the hinge loss function will return a value of 0 – which corresponds to making no violations – if $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1$. Also note that the distance from a point $\mathbf{x}_i$ to a hyperplane is measured by the equation $\frac{y_i(\mathbf{w}^\top \mathbf{x}_i + b)}{\|\mathbf{w}\|_2}$. If we assume without loss of generality that $\|\mathbf{w}\|_2 = 1$, then the hinge loss function corresponding to linear SVM only penalizes when the distance from the data point and the hyperplane is less than 1. Hence, minimizing this loss function equates to finding the coefficients such that the distance between the hyperplane and the data sample $\mathbf{x}_i$ is as far away as possible. The data samples that are closest to the boundary (or hyperplane) are called the "support vectors", since they primarily determine the boundary that yields the smallest amount of mis-classifications. Formulating this idea mathematically, we have the following optimization problem:

$$\hat{\mathbf{w}}, \hat{b} = \underset{\mathbf{w}, b}{\operatorname{argmin}} \sum_{i=1}^{n} \max[0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)] + \lambda \|\mathbf{w}\|_2. \tag{3.11}$$

Similar to Logistic Regression, we need to use optimization techniques such as gradient descent to estimate these parameters. Once we estimate these parameters, we can simply look at which side of the hyperplane the data sample lies on, i.e. $\mathbf{w}^\top \mathbf{x}_i + b \geq 0$ or $\mathbf{w}^\top \mathbf{x}_i + b < 0$, to determine its label.

3.4 Tensor-Structured Machine Learning

In this section, we show how we can reformulate the optimization problem for both Logistic Regression and SVM to exploit the structure of tensor data. To do this, we impose a CP decomposition on the weights that we solve for an alternating minimization method. We discuss these techniques in detail in the following sections.

3.4.1 CP-Structured Logistic Regression

Recall that the objective of Logistic Regression is to estimate the predictors $\mathbf{w}$ and b such that the error given by

$$\min_{\mathbf{w},b} \sum_{i=1}^{n} \log(1 + \exp(-y_i(\mathbf{w}^\top \mathbf{x_i} + b))) + \lambda \|\mathbf{w}\|_2 \tag{3.12}$$

is small. This formulation implicitly requires the data samples $\mathbf{x}_i$ (and consequently $\mathbf{w}$) to be vectorized. Now, consider a tensor dataset $\{(\underline{\mathbf{X}}_i, y_i)\}_{i=1}^n$, where $\underline{\mathbf{X}}_i \in \mathbb{R}^{D_1 \times \ldots \times D_N}$ and $y_i \in \{0, 1\}$. Instead of vectorizing the tensor data sample $\underline{\mathbf{X}}_i$ to fit Equation 3.12, we can reformulate this equation by imposing the CP decomposition on $\mathbf{w}$ as follows:

$$\min_{\mathbf{w}_1,\ldots,\mathbf{w}_N} \sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i \left\langle \sum_{r=1}^{R} \mathbf{W}_{1,r} \circ \ldots \circ \mathbf{W}_{N,r}, \underline{\mathbf{X}}_i \right\rangle\right)\right). \tag{3.13}$$

This new optimization problem, which we call CP-Logistic Regression, involves estimating N factor matrices, one for each dimension. On the face of this problem, it may seem like we are estimating even more parameters than before. To see why this is not true, consider each tensor data sample $\underline{\mathbf{X}}_i$ to be of dimensions $\underline{\mathbf{X}}_i \in \mathbb{R}^{100 \times 100 \times 100}$. If we choose the rank to be 5, i.e. $R = 5$, then CP-Logistic Regression involves estimating a total of 1500 parameters across all factor matrices, whereas the traditional Logistic Regression algorithm estimates a total of 1000000 parameters! This is a significant reduction in the number parameters, which empirically reduces the sample complexity and also exploits the low-rank structure of the data. The remaining challenge lies in how we estimate these N factor matrices. As previously mentioned, we use a technique called alternating minimization, which we discuss in Section 3.4.3.

3.4.2 CP-Structured Support Vector Machines

Similar to Logistic Regression, recall that the objective of linear SVM is to solve for the predictors that minimizes the function

$$\min_{\mathbf{w},b} \sum_{i=1}^{n} \max[0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)] + \lambda \|\mathbf{w}\|_2. \tag{3.14}$$

By imposing the CP decomposition on this loss function, we have the following reformulation:

$$\min_{\mathbf{w}_1,\ldots,\mathbf{w}_N} \sum_{i=1}^{n} \max\left[0, 1 - y_i\left(\left\langle \sum_{r=1}^{R} \mathbf{W}_{1,r} \circ \ldots \circ \mathbf{W}_{N,r}, \underline{\mathbf{X}}_i \right\rangle\right)\right]. \tag{3.15}$$

We refer to this new formulation as CP-SVM. Similar to CP-Logistic Regression, we discuss how we estimate these factor matrices in the next section.

3.4.3 Estimating the CP Factor Matrices

Unlike the traditional machine learning algorithms, the estimation of the factor matrices is a non-trivial task. To solve for these matrices, we adopt an alternating minimization scheme proposed by Zhou et al. [21]. The alternating minimization method involves optimizing over one variable while keeping the others fixed. This step is briefly outlined in Algorithm 1. Using this for our CP-structured algorithms, we need to rewrite the inner product

$$\left\langle \sum_{r=1}^{R} \mathbf{W}_{1,r} \circ \ldots \circ \mathbf{W}_{N,r}, \underline{\mathbf{X}}_i \right\rangle \tag{3.16}$$

that is present in both CP-Logistic Regression and CP-SVM. We can do this by using the property of the CP decomposition stated in Section 2.3.2. Specifically, when updating factor matrix $\mathbf{W}_i$, we rewrite the inner product as

$$\left\langle \sum_{r=1}^{R} \mathbf{W}_{1,r} \circ \ldots \circ \mathbf{W}_{N,r}, \underline{\mathbf{X}}_i \right\rangle = \langle \mathbf{W}_i, \mathcal{M}_i\left(\underline{\mathbf{X}}\right)(\mathbf{W}_D \odot \ldots \odot \mathbf{W}_{i+1} \odot \mathbf{W}_{i-1} \odot \ldots \odot \mathbf{W}_1)\rangle \tag{3.17}$$

This allows us the single out the factor matrix that we want to optimize over, making computation simpler when using built-in optimizers in Python and MATLAB.

There are a couple key advantages in using the alternating minimization algorithm. First, in the literature, this algorithm has been shown to almost always converges to at least a local minimum [21, 27, 28]. To find the best solution, the algorithm can be ran several times with different initial factor matrices. Second, the low-rank optimization problem over the factor matrices is non-convex [29]. Thus, this problem becomes difficult to solve using common unconstrained solvers, such as gradient descent.

Algorithm 1 Alternating Minimization for Estimating CP Factors [21]

Require: Data $\underline{\mathbf{X}} \in \mathbb{R}^{D_1 \times \ldots \times D_N}$: $\{(\underline{\mathbf{X}}_i, y_i)\}_{i=1}^n$; Loss function: $f(\underline{\mathbf{X}}_i, y_i, \mathbf{A}_i)$; Rank: R; Regularization parameter: λ; Iterations: T

1: Randomly Initialize: $\mathbf{A}_i \in \mathbb{R}^{D_i \times R}$ for $i = 1, \ldots, N$
2: **for** $t = 1, \ldots, T$ **do**
3: **for** $i = 1, \ldots, N$ **do**
4: $\mathbf{A}_i^{t+1} = \underset{\mathbf{A}_i}{\operatorname{argmin}} \sum_{j=1}^n f(\mathbf{X}_j, y_j, \mathbf{A}_1^{t+1}, \ldots, \mathbf{A}_{i-1}^{t+1}, \mathbf{A}_i, \mathbf{A}_{i+1}^t, \ldots, \mathbf{A}_N^t) + \lambda \|\mathbf{A}_i\|_2$
5: **end for**
6: **end for**

Ensure: $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_N$

In the literature, there are two ways to handle the non-convexity of this optimization problem. One way is to relax the rank constraint by adding a convex regularization term that induces low-rank (e.g. trace norm, nuclear norm) [30, 31]. The other solution is to employ this alternating minimization algorithm, as the optimization over one matrix, while holding the others fixed is convex. We chose to explore this procedure following Zhou et al. [21], as the algorithm is straightforward to implement using statistical software and packages in MATLAB or Python.

3.5 Numerical Experiments

We compare the performance of CP-Logistic Regression and CP-SVM with the traditional Logistic Regression and SVM algorithms presented in Section 3.3. We use both synthetic data and the MNIST dataset [32] to observe how the performance of these algorithms vary as we increase the available number of data samples and the rank of the CP-structured algorithms. To quantitatively compare these algorithms, we use the following metrics:

1. (Normalized) Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{n} \|\mathbf{w} - \hat{\mathbf{w}}\|_2^2, \tag{3.18}$$

where n is the total number of data samples, $\mathbf{w}$ are the true weights, and $\hat{\mathbf{w}}$ are the reconstructed weights.

2. Cosine Distance (or Cosine Similarity) [33]:

$$\cos(\theta) = \frac{\langle \mathbf{w}, \hat{\mathbf{w}} \rangle}{\|\mathbf{w}\|_2 \cdot \|\hat{\mathbf{w}}\|_2}, \tag{3.19}$$

where $\mathbf{w}$ are the true weights and $\hat{\mathbf{w}}$ are the reconstructed weights. The cosine distance measures the cosine of the angle between two vectors in n-dimensional space. Two vectors will have a cosine distance of 0 if they are perpendicular, and a value of 1 if the direction of the vectors is identical.

3. Prediction Accuracy:

$$\text{Accuracy} = \frac{\#\text{ of correct predictions}}{\#\text{ total number of samples } (n)}. \tag{3.20}$$

Lastly, recall that we added an ℓ_2 regularizer to all of the algorithms. We used cross-validation to compute the regularization parameter, which were $\lambda = 10^{-5}$ for the CP-structured algorithms and $\lambda = 10^{-2}$ for the traditional algorithms.

Experiments with Synthetic Data. The main objective of the experiments with synthetic data was to show the advantages of the CP-structured algorithms when the predictors had an exact low-rank structure. We generated synthetic data according to the model

$$y_i = \text{sign}(\langle \beta, \mathbf{X}_i \rangle + \epsilon), \tag{3.21}$$

where $\mathbf{X}_i \in \mathbb{R}^{15\times 15}$ with entries drawn independently and identically distributed from $\mathcal{N}(0,1)$, $\epsilon_i \sim \mathcal{N}(0,1)$, and $\beta \in \mathbb{R}^{15\times 15}$ was fixed to be a cross as shown in Figure 3.3. This cross has a rank of 2, and we use this as the value of the rank for our CP-structured algorithms. Note that with a rank value of 2, the CP-structured algorithms both have a total of 60 parameters, whereas the traditional algorithms have a total of 225 parameters. Our hypothesis was that the CP-structured algorithms would outperform the traditional algorithms when the number of available samples was more limited, as there are less parameters that needed to be estimated. In Figure 3.2, we can observe the validity of this hypothesis. As the sample sizes increase, the performance of the traditional machine learning algorithms converges to that of the CP-structured algorithms.

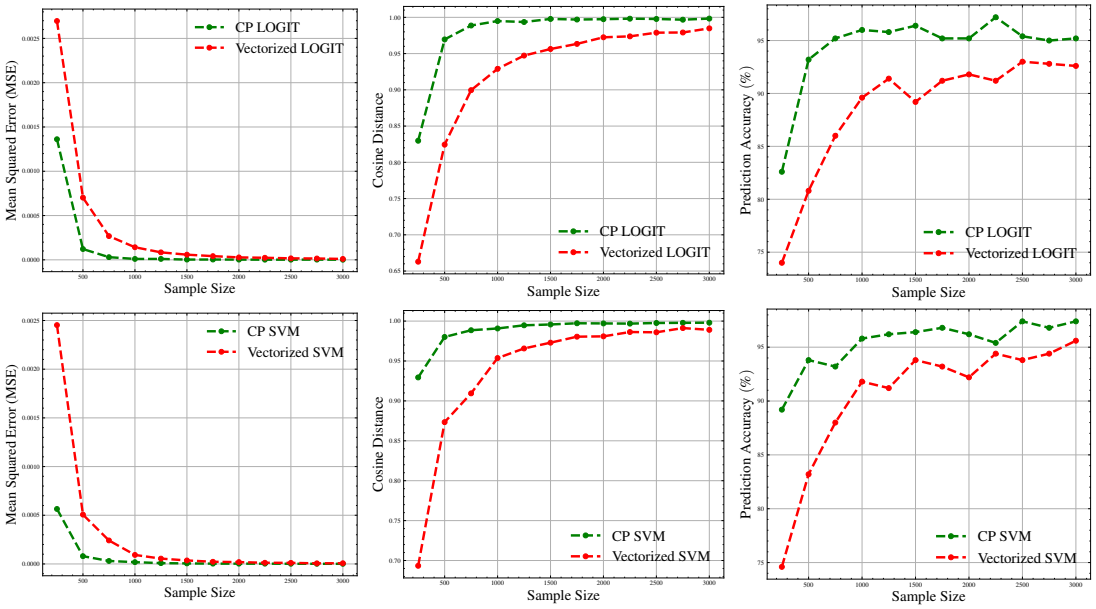


Figure 3.2: Performance comparison between CP-structured methods and traditional methods with increasing sample sizes for three different metrics: normalized MSE, cosine distance, and prediction accuracy (%). Row 1: Comparison between CP-LOGIT and traditional LOGIT. Row 2: Comparison between CP-SVM and traditional SVM.

In Figure 3.3, we can visually see the change in the reconstruction of the cross for different sample sizes. However, even when the number of samples is approximately 13 times number of parameters ($n = 3000$), the solved predictors from traditional Logistic Regression is still visually far from the original cross, whereas the predictors from CP-Logistic Regression is visually much closer. This highlights the effectiveness of the CP-structured algorithms that takes the low-rank property into consideration during optimization.

Experiments with MNIST Data. The experiments with the synthetic dataset showed the change in performance metrics for a fixed predictor that had an exact low-rank structure. The remaining question that we need to answer is how these metrics behave for a predictor that is approximately low-rank and the effect of not knowing the rank a priori. To simulate an approximately low-rank predictor, we took a "one" from the MNIST dataset, fixed the digit as our predictor, generated data according to Equation (3.21) and observed how the metrics changed for different values of the rank for a fixed sample size. We record the metrics in Table 3.1 and display the visual

Algorithm	Rank	MSE	Cosine Distance	Accuracy (%)
Vectorized LOGIT		0.000125	0.875	84.2
CP LOGIT	$r = 3$	0.000134	0.866	80.4
	$r = 4$	0.0000707	0.929	85.6
	$r = 5$	0.0000518	0.948	87.4
	$r = 6$	0.0000534	0.947	86.4
Vectorized SVM		0.000115	0.885	83.8
CP SVM	$r = 3$	0.000147	0.853	80.2
	$r = 4$	0.0000847	0.915	85.2
	$r = 5$	0.0000657	0.937	88.0
	$r = 6$	0.0000584	0.942	89.8

Table 3.1: Numerical comparison between CP-structured methods and traditional methods with increasing ranks for three different metrics: normalized MSE, cosine distance, and prediction accuracy (%). These metrics were averaged over 5 runs for a sample size of 2000.

reconstructions in Figure 3.4. In Table 3.1, we observe that a rank of 3 is insufficient for accurately estimating the MNIST digit, as the performance across all metrics is lower than that of the traditional algorithms. A rank value of $r = 5, 6$ seem to be the best performing values, as the errors are both numerically and visually lower. As the ranks increase, we hypothesize that the performance will eventually degrade, as the number of parameters for the CP-structured algorithms converge to that of the traditional algorithms, similar to the phenomenon we saw in the previous synthetic experiment.

3.6 Conclusion and Future Work

In this chapter, we investigated tensor-based classification models using a CANDECOMP/PARAFAC factorization structure on the predictors of traditional machine learning algorithms, namely Support Vector Machines and Logistic Regression. Imposing these structures allowed us to exploit the structure of the data, while solving for fewer parameters. We showed with different performance metrics that our proposed method overall estimated a more accurate reconstruction of the weights. The experiments showed that the CP-structured algorithms performed best when the true predictor had either an approximate or an exact low rank structure. The most relevant direction to extend the ideas presented in this chapter is to explore classification models

using the Tucker decomposition. The Tucker decomposition allows for more flexibility in the choices of the ranks, which may be beneficial to tensor data such as RGB images. We believe it would be interesting to show that the Tucker-structured algorithms also increases the same performance metrics under both an exact and approximate low-rank structure.

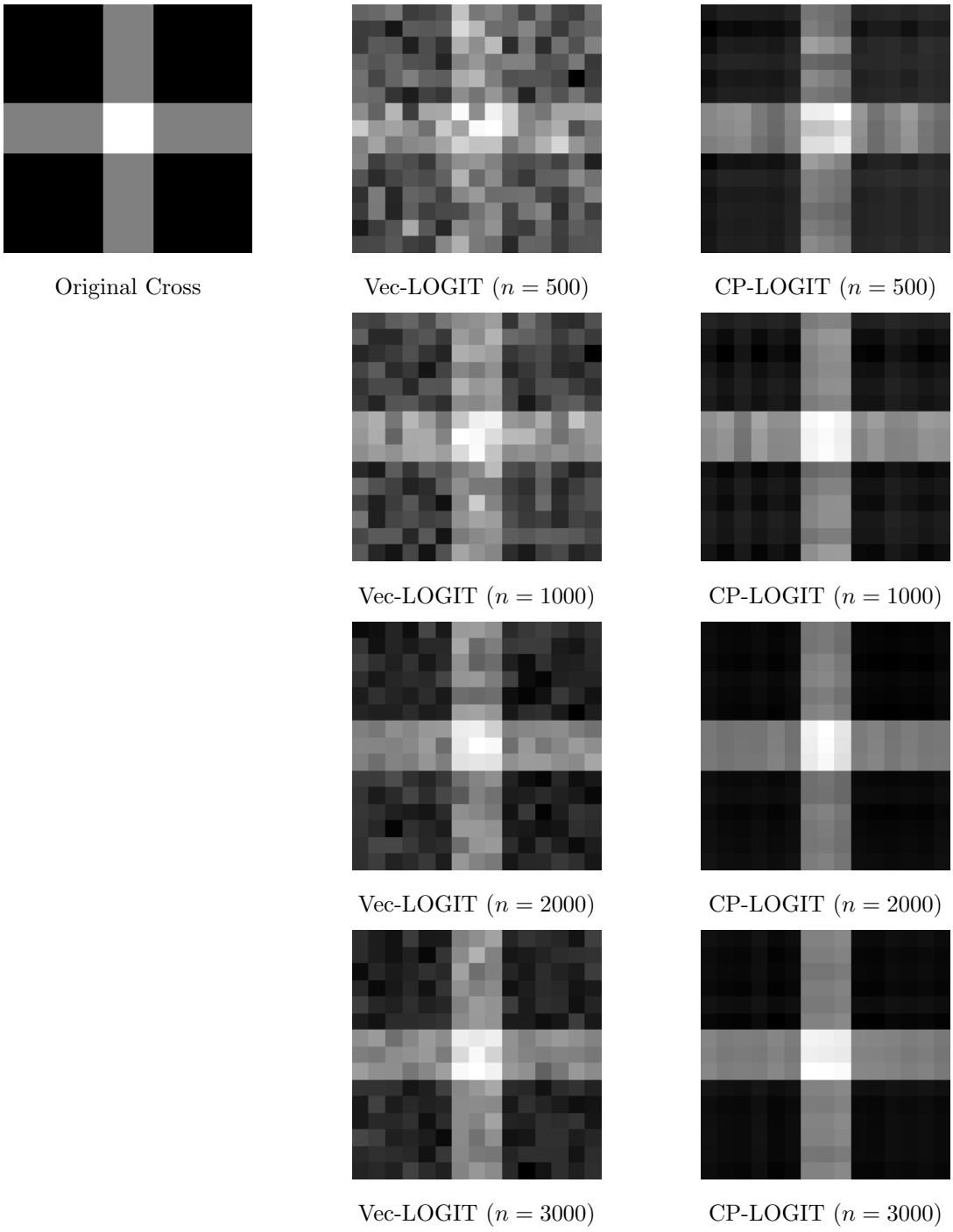
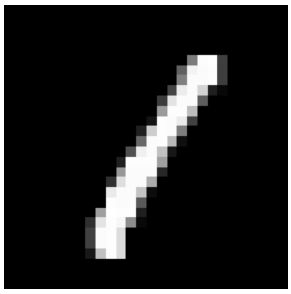
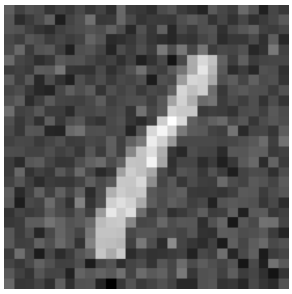


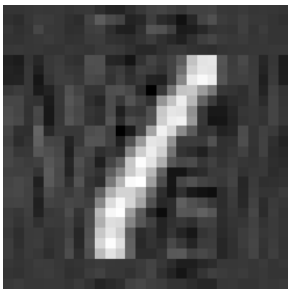
Figure 3.3: Visual comparison of CP-Logistic Regression (CP-LOGIT) and traditional Logistic Regression (Vec-LOGIT) when the predictors exhibit an exact low-rank structure with increasing sample sizes.



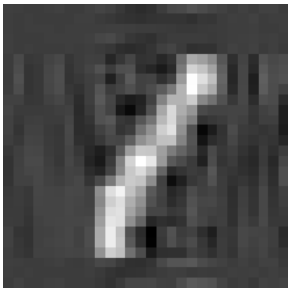
Original MNIST



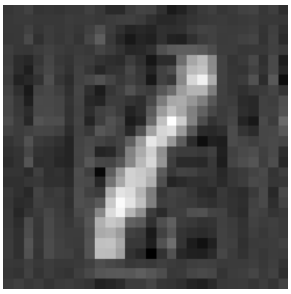
Vectorized SVM



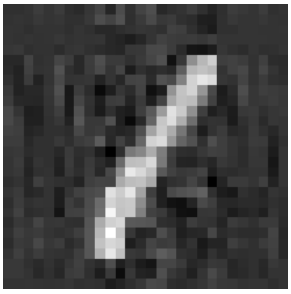
CP SVM $r = 3$



CP SVM $r = 4$



CP SVM $r = 5$



CP SVM $r = 6$

Figure 3.4: Visual comparison of CP-Support Vector Machine (CP-SVM) and traditional Support Vector Machine (Vec-SVM) when the predictors exhibit an approximate low-rank structure with increasing ranks for a sample size of 2000.

Chapter 4

Low-Rank Phase Retrieval with Structured Tensor Models

4.1 Introduction

Many practical applications involve solving systems of linear equations, where the objective is to estimate a signal $\mathbf{x} \in \mathbb{C}^n$ given sampling vectors $\mathbf{a}_i \in \mathbb{C}^n$ and observations of the form

$$y_i = \langle \mathbf{a}_i, \mathbf{x} \rangle, \; i = 1, \ldots, m. \tag{4.1}$$

This problem has been extensively studied in the literature and there are many different ways in solving these systems of linear equations in both under and over-determined cases (i.e. $m \ll n$ and $m \gg n$) [34]. However, if we posed this problem in such a way that we have no prior information about the signs or phases of $\langle \mathbf{a}_i, \mathbf{x} \rangle$, the problem becomes significantly more difficult. This problem is commonly referred to as phase retrieval (or quadratic sensing), where the objective to recover $\mathbf{x} \in \mathbb{C}^n$ given sampling vectors $\mathbf{a}_i \in \mathbb{C}^n$ and observations of the form

$$y_i = |\langle \mathbf{a}_i, \mathbf{x} \rangle|, \; i = 1, \ldots, m, \tag{4.2}$$

or equivalently, $y_i = |\langle \mathbf{a}_i, \mathbf{x} \rangle|^2$. Phase retrieval arises from a wide range of imaging domains such as X-ray crystallography [35], Fourier ptychography [36, 37], and astronomy [38]. In each of these domains, the measurement acquisition process generally involves an optical sensor that captures the diffracted patterns of the object of interest. However, the physical limitations of these sensors only allow us to observe the intensities (or magnitudes) of these patterns, leading to the quadratic system.

The importance of solving the phase retrieval problem in these imaging domains have led to many convex and non-convex solutions, such as the Gerchberg-Saxton algorithm [39], PhaseLift [40], AltMinPhase [3], and the variants of Wirtinger Flow [2, 41,

42], to name a few. However, the theoretical guarantees of all existing methods require the system to be over-determined (i.e. $m \gg n$). This requirement, which is considered to be the bottleneck of phase retrieval, mainly comes from the non-convex nature of the problem. In order to converge to the optimal solution, one needs enough samples to guarantee that the initial estimate of the signal is close to the true signal with high probability. This initial estimation step is called spectral initialization, where the term "spectral" comes from the use eigenvectors (or singular vectors) of properly designed matrices from data [43]. This step has been shown to be essential for solving the phase retrieval problem, and many variants of this step have been proposed in the literature.

Recently, there has been a surge of interest in solving the *low-rank phase retrieval* problem [44–48]. This problem can be viewed as a dynamic extension of the standard phase retrieval problem, where the objective is to recover a matrix of vectorized images rather than a single image. Formally, we want to estimate a low-rank matrix $\mathbf{X} \in \mathbb{C}^{n \times q}$, where $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_q]$ with $\mathbf{x}_k \in \mathbb{C}^n$, given sampling matrices $\mathbf{A}_k \in \mathbb{C}^{n \times m}$ and measurements

$$\mathbf{y}_k = |\mathbf{A}_k^* \mathbf{x}_k|, \; k = 1, \ldots, q. \tag{4.3}$$

In this problem formulation, we assume that there is a separate, independent set of sampling matrices $\mathbf{A}_k$ for each signal $\mathbf{x}_k$. Unlike the phase retrieval problem, this problem has several solutions that have strong theoretical guarantees even for the under-determined setting (i.e. $m \ll n$). These algorithms exploit the low-rank property of the matrix $\mathbf{X}$ with the extra set of sampling matrices in order to naturally reduce the sample complexity. However, our empirical results suggest that there is perhaps a gap between theory and practice, and that these solutions fail to accurately recover the images in the under-determined setting. In fact, in this setting, we observe that these algorithms often do not converge.

To overcome this challenge, we propose an algorithm called Tucker-Structured Phase Retrieval (TSPR) that models the sequence of images as a tensor rather than a matrix. With a tensor model, we can decompose the tensor using the Tucker decomposition [9] to estimate fewer parameters than the matrix counterpart. The reduction in the number

of parameters also decreases the number of degrees of freedom, allowing the recovery of the sequence of signals possible with a smaller sample complexity. To demonstrate the effectiveness of our approach, we conduct experiments on real video datasets with measurements generated from real and complex Gaussian vectors and coded diffraction patterns. Our results show that in all of these measurement settings, our algorithm outperforms existing algorithms in both the under and over-determined regimes.

4.2 Related Work

4.2.1 Phase Retrieval

To the best of our knowledge, the most notable approaches for solving the classical phase retrieval problem are the variants of Wirtinger Flow, specifically Truncated Wirtinger Flow (TWF) [41] and Reshaped Wirtinger Flow (RWF) [42]. These two methods extend the main ideas from the original Wirtinger Flow (WF) algorithm by Candès et al. [2] by making a few algorithmic modifications. These modifications reduce the sample complexity of recovering the true signal from measurements $\Omega(n\log(m))$ in WF to $\Omega(n)$ in TWF and RWF under the Gaussian measurement assumption.

Since low-rank phase retrieval is simply a dynamic extension of the classical phase retrieval problem, one might wonder why we cannot just use the existing WF algorithms to solve for each image $\mathbf{x}_k \in \mathbb{R}^n$ in the low-rank matrix $\mathbf{X}$. While this is a very plausible solution, this solution would only work if we had $m \geq Cn$ for some $C > 1$ measurements for each $\mathbf{x}_k$. Hence, this method would not be solving the low-rank phase retrieval problem in the under-determined regime, which is what we are aiming to solve in this chapter. Additionally, this would not be using the low-rank structure of the matrix $\mathbf{X}$ along with the extra set of sampling matrices, which is what we ultimately want to prudently leverage to reduce the sample complexity.

4.2.2 Unstructured Low-Rank Phase Retrieval

There are several provably efficient algorithms for solving the low-rank phase retrieval problem that vectorize each image and recover a low-rank matrix. We call such methods "unstructured" because they assume no structure in the images. Recently, Nayer et al. proposed AltMinLowRaP [45], an algorithm that theoretically improved their previous algorithm AltMinTrunc [44, 47], that both solved the unstructured low-rank phase retrieval problem. AltMinLowRaP involved alternately minimizing the factor matrices $\mathbf{U} \in \mathbb{C}^{n \times r}$ and $\mathbf{B} \in \mathbb{C}^{q \times r}$ that constructed the low-rank matrix $\mathbf{X} = \mathbf{U}\mathbf{B}^*$. Updating the factor matrix $\mathbf{U}$ consisted of minimizing the objective function

$$\underset{\mathbf{U}}{\operatorname{argmin}} \sum_k \|\mathbf{C}_k \mathbf{y}_k - \mathbf{A}_k^* \mathbf{U} \mathbf{b}_k\|_2^2, \tag{4.4}$$

where $\mathbf{b}_k$ is the k-th row of the matrix $\mathbf{B}$ and $\mathbf{C}_k$ is a diagonal phase matrix. Note that this objective function sums over all of the columns in $\mathbf{X}$, as the k-th column of $\mathbf{X}$ can be written as $\mathbf{x}_k = \mathbf{U}\mathbf{b}_k$. The intuition behind this summation can be viewed as each of the vectorized images $\mathbf{x}_k$ differing by $\mathbf{b}_k$, while sharing the same span($\mathbf{U}$). Optimizing for $\mathbf{U}$ involved minimizing this objective function using conjugate gradient least squares (CGLS) while keeping $\mathbf{b}_k$ fixed. The factor matrix $\mathbf{B}$ was initialized and updated by solving an r-dimensional noisy phase retrieval problem for each row of $\mathbf{B}$, $\mathbf{b}_k$. To see this, note that we can rewrite each of the measurements as

$$y_{i,k} = |\langle \mathbf{a}_{i,k}, \mathbf{x}_k \rangle| \tag{4.5}$$

$$= |\langle \mathbf{a}_{i,k}, \mathbf{U}\mathbf{b}_k \rangle| = |\langle \mathbf{U}^* \mathbf{a}_{i,k}, \mathbf{b}_k \rangle|. \tag{4.6}$$

Given an estimate of $\mathbf{U}$, we can solve for each $\mathbf{b}_k$ using any phase retrieval method, such as Reshaped Wirtinger Flow (RWF) [42]. Thus, AltMinLowRaP runs RWF q times (once for each image) to estimate $\mathbf{b}_k$ given the sampling matrix $\mathbf{U}^* \mathbf{a}_{i,k}$. Lastly, upon updating the matrix $\mathbf{U}$ and each vector $\mathbf{b}_k$, the phase matrices were also updated by taking the phases of $\mathbf{x}_k = \mathbf{U}\mathbf{b}_k$ by

$$\mathbf{C}_k = \text{Diag}(\text{Phase}(\mathbf{A}_k^* \mathbf{U} \mathbf{b}_k)). \tag{4.7}$$

Due to the non-convex nature of this problem, the factor matrix $\mathbf{U}$ was initialized via a spectral method, as previously done in WF [2] and TWF [41]. The matrix $\mathbf{U}$ was

initialized by taking the top r eigenvectors of the surrogate matrix

$$\mathbf{Y} = \frac{1}{mq}\sum_{i=1}^{m}\sum_{k=1}^{q} y_{i,k}^2 \mathbf{a}_{i,k}\mathbf{a}_{i,k}^* \mathbf{1}_{\{y_{i,k}^2 \leq \frac{\alpha^2}{mq}\sum_{t,v} y_{t,v}^2\}}, \tag{4.8}$$

for some trimming threshold α. The intuition behind this matrix is that given enough samples, the expectation of this matrix is equivalent to

$$\mathbb{E}[y_{i,k}\mathbf{a}_{i,k}\mathbf{a}_{i,k}^*] = 2\mathbf{x}_k\mathbf{x}_k^* + \|\mathbf{x}_k\|^2\mathbf{I_n}. \tag{4.9}$$

Thus, the subspace spanned by the top r eigenvectors of $\mathbf{Y}$ can recover exactly $\mathbf{U}$. The double summation over the measurements and samples in the surrogate matrix and truncation is what guaranteed AltMinLowRaP a smaller sample complexity over existing methods. Our algorithm is an improvement over AltMinLowRaP that empirically works better in both the under and (some) over-sampled regimes. Although our algorithm does not yet have a theoretical analysis of the sample complexity, our empirical results show that our algorithm can work better in practice.

4.3 Tensor Structured Low-Rank Phase Retrieval

As previously mentioned, one natural way of reducing the sample complexity is to estimate less parameters by assuming an additional low-rank structure on each of the vectorized images $\mathbf{x}_k$. In order to impose this structure, we need to model the sequence of q images as a tensor by reshaping and stacking each of the vectorized images from $\mathbf{x}_k \in \mathbb{C}^n$ into $\mathbf{X}_k \in \mathbb{C}^{n_1 \times n_2}$, where $n = n_1 n_2$. The objective of TSPR is to recover this tensor $\underline{\mathbf{X}} \in \mathbb{C}^{n_1 \times n_2 \times q}$ given measurements $\mathbf{y}_k$ and sampling matrices $\mathbf{A}_k$, where $\underline{\mathbf{X}}$ can be factorized using either the Tucker or the CP decomposition.

For low-rank phase retrieval, the Tucker decomposition is more suitable than the CP decomposition, as it allows flexibility in our choices of the ranks. We need this flexibility since the structure of each image $\mathbf{X}_k$ may not exactly be low-rank, whereas the temporal dimension will most likely be low-rank. The Tucker decomposition allows the rank of each mode to be different, making it more appropriate than the CP decomposition. Now, using the Tucker formulation, the objective of TSPR is to estimate the factors

$\underline{\mathbf{G}} \in \mathbb{C}^{r_1 \times r_2 \times r_3}$, $\mathbf{D} \in \mathbb{C}^{n_1 \times r_1}$, $\mathbf{E} \in \mathbb{C}^{n_2 \times r_2}$, and $\mathbf{F} \in \mathbb{C}^{q \times r_3}$ that make up the tensor $\underline{\mathbf{X}}$ as

$$\underline{\mathbf{X}} = \underline{\mathbf{G}} \times_1 \mathbf{D} \times_2 \mathbf{E} \times_3 \mathbf{F}. \tag{4.10}$$

We want to solve for these factors by first initializing them via a spectral method and then estimating them using alternating minimization and CGLS.

4.3.1 Spectral Initialization

The idea behind our spectral initialization step is to construct a tensor that is close to the true tensor $\underline{\mathbf{X}}$ with high probability. Once we construct this tensor, we can use higher-order SVD (HOSVD) [49] to initialize our core tensor and factor matrices. To do this, we adopt the initialization technique of Truncated Wirtinger Flow (TWF) [41] to obtain an initial estimate of the vectorized image $\mathbf{x}_k$. Specifically, we want to first take the leading eigenvector of the constructed matrix

$$\mathbf{Y}_k = \sum_{i=1}^{m} y_{i,k}^2 \mathbf{a}_{i,k} \mathbf{a}_{i,k}^* \mathbf{1}_{\{|y_{i,k}|^2 \leq \alpha^2 \lambda_k^2\}}, \tag{4.11}$$

where

$$\lambda_k = \sqrt{\frac{1}{m} \sum_{i=1}^{m} y_{i,k}}. \tag{4.12}$$

If $\mathbf{z}_k$ is the leading eigenvector of $\mathbf{Y}_k$, we compute the initial estimate of $\mathbf{x}_k$ as

$$\mathbf{x}_k = \sqrt{\frac{mn}{\sum_{i=1}^{m} \|\mathbf{a}_{i,k}\|_2^2}} \lambda_k \mathbf{z}_k, \tag{4.13}$$

which appropriately normalizes $\mathbf{z}_k$ to approximately have the same norm as $\mathbf{x}_k$. Upon computing each $\mathbf{x}_k$ for $k = 1, \ldots, q$, we reshape $\mathbf{x}_k$ back into its original dimensions and stack them to create the initial tensor. This initialization step is outlined in Algorithm 2.

4.3.2 Alternating Minimization

Upon initialization, we can alternately update the core tensor and each factor matrix using CGLS and RWF. Recall that in AltMinLowRaP, we minimized an objective function that was formed by plugging in $\mathbf{x}_k = \mathbf{U}\mathbf{b}_k$. Similarly, we can minimize the same

Algorithm 2 TSPR Initialization

Require: Observations: $\{y_{i,k} \,|\, 1 \leq i \leq m, 1 \leq k \leq q\}$; Sampling vectors: $\{\mathbf{a}_{i,k} \,|\, 1 \leq i \leq m, 1 \leq k \leq q\}$; Trimming threshold: α; Ranks: $= [r_1, r_2, r_3]$

1: **for** $k = 1, \ldots, q$ **do**

2: Compute $\lambda_k = \sqrt{\frac{1}{m}\sum_{i=1}^{m} y_{i,k}}$.

3: Compute $\mathbf{z}_k$ as leading eigenvector of

$$\mathbf{Y}_k = \sum_{i=1}^{m} y_{i,k}^2 \mathbf{a}_{i,k}\mathbf{a}_{i,k}^* \mathbf{1}_{\{|y_{i,k}|^2 \leq \alpha^2 \lambda_k^2\}}$$

4: Compute $\mathbf{x}_k = \sqrt{\frac{mn}{\sum_{i=1}^{m} \|\mathbf{a}_{i,k}\|_2^2}} \lambda_k \mathbf{z}_k$.

5: Reshape $\mathbf{x}_k \in \mathbb{C}^n$ into $\mathbf{X}_k \in \mathbb{C}^{n_1 \times n_2}$.

6: **end for**

7: Stack tensor into $\underline{\mathbf{X}} = [\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_q]$

8: Initialize factors using HOSVD:

$$\mathbf{D}^0, \mathbf{E}^0, \mathbf{F}^0, \underline{\mathbf{G}}^0 = \text{HOSVD}(\underline{\mathbf{X}}, \text{ranks})$$

Ensure: $\mathbf{D}^0, \mathbf{E}^0, \mathbf{F}^0, \underline{\mathbf{G}}^0$

function, but by rewriting $\mathbf{x}_k$ using our Tucker factors. In specific, we can write each $\mathbf{x}_k$ as

$$\mathbf{x}_k = (\mathbf{f}_k \otimes \mathbf{E} \otimes \mathbf{D})\text{vec}(\underline{\mathbf{G}}), \tag{4.14}$$

where $\mathbf{f}_k$ is the k-th row of the factor matrix $\mathbf{F}$. The reason behind writing $\mathbf{x}_k$ in terms of $\mathbf{f}_k$ is the same reasoning used for the unstructured case – each image $\mathbf{x}_k$ differs by $\mathbf{f}_k$. By plugging in $\mathbf{x}_k$, the update step of the core tensor $\underline{\mathbf{G}}$ involves minimizing the function

$$\sum_k \|\mathbf{C}_k \mathbf{y}_k - \mathbf{A}_k^*(\mathbf{f}_k \otimes \mathbf{E} \otimes \mathbf{D})\text{vec}(\underline{\mathbf{G}})\|_2^2 . \tag{4.15}$$

We use CGLS to update the vectorized form of $\underline{\mathbf{G}}$ given matrix $\mathbf{A}_k^*(\mathbf{f}_k \otimes \mathbf{E} \otimes \mathbf{D})$ and reshape it before the next update step. Now, in order to use CGLS for matrices $\mathbf{D}$ and $\mathbf{E}$, we need to rewrite $\mathbf{x}_k$ in such a way that we have $\text{vec}\,(\mathbf{D})$ and $\text{vec}\,(\mathbf{E})$ are on the right-hand side as in Equation (4.15). To do this, we can use the property of the $\text{vec}\,(\cdot)$ operator as mentioned in Section 2.3.1. Using this property, the update step for factor

Algorithm 3 Tucker-Structured Phase Retrieval (TSPR)

Require: Observations: $\{y_{i,k} \,|\, 1 \le i \le m, 1 \le k \le q\}$; Sampling vectors: $\{\mathbf{a}_{i,k} \,|\, 1 \le i \le m, 1 \le k \le q\}$; Initial factors: $\mathbf{D}^0, \mathbf{E}^0, \mathbf{F}^0, \underline{\mathbf{G}}^0$; Iterations: T; RWF Iterations: T_{RWF}

1: **for** $t = 1, \ldots, T$ **do**
2: **for** $k = 1, \ldots, q$ **do**
3: Update $\mathbf{f}_k^{t+1} = \text{RWF}([\mathbf{D}^t, \mathbf{E}^t, \underline{\mathbf{G}}^t, \mathbf{A}_k^*], \mathbf{y}_k, T_{RWF})$
4: Compute $\mathbf{x}_k^{t+1} = (\mathbf{f}_k^{t+1} \otimes \mathbf{E}^t \otimes \mathbf{D}^t)\text{vec}(\underline{\mathbf{G}}^t)$
5: Update diagonal phase matrix $\mathbf{C}_k^{t+1} = \text{Diag}(\text{Phase}(\mathbf{A}_k^* \mathbf{x}_k^{t+1}))$
6: **end for**
7: Update $\mathbf{D}^{t+1}$, $\mathbf{E}^{t+1}$, $\underline{\mathbf{G}}^{t+1}$ by minimizing (4.15)
8: **end for**
9: Reconstruct tensor $\underline{\mathbf{X}}^T = \underline{\mathbf{G}}^T \times_1 \mathbf{D}^T \times_2 \mathbf{E}^T \times_3 \mathbf{F}^T$

Ensure: $\underline{\mathbf{X}}^T$

matrix $\mathbf{D}$ involves minimizing the objective function

$$\sum_k \|\mathbf{C}_k \mathbf{y}_k - \mathbf{A}_k^* (\mathbf{S}_k^* \otimes \mathbf{I}_{\mathbf{n_1}}) \text{vec}(\mathbf{D})\|^2, \tag{4.16}$$

where $\mathbf{S}_k = \mathcal{M}_1(\underline{\mathbf{G}})(\mathbf{f}_k \otimes \mathbf{E})^*$. Similarly, we can follow the same steps for factor matrix $\mathbf{E}$. Using the same property, the update step for factor matrix $\mathbf{E}$ involves minimizing the objective function

$$\sum_k \|\mathbf{C}_k \mathbf{y}_k - \mathbf{A}_k^* (\mathbf{I}_{\mathbf{n_2}} \otimes \mathbf{T}_k^*) \text{vec}(\mathbf{E}^*)\|^2, \tag{4.17}$$

where $\mathbf{T}_k = \mathcal{M}_2(\underline{\mathbf{G}})(\mathbf{f}_k \otimes \mathbf{D})^*$. To update each row vector $\mathbf{f}_k$, note that we can rewrite $y_{i,k}$ as

$$y_{i,k} = |\langle \mathbf{a}_{i,k}, \mathbf{x}_k \rangle| \tag{4.18}$$

$$= |\langle \mathbf{a}_{i,k}, \mathcal{M}_3(\underline{\mathbf{G}})(\mathbf{E} \otimes \mathbf{D})^* \mathbf{f}_k \rangle| = |\langle \mathcal{M}_3(\underline{\mathbf{G}})(\mathbf{E} \otimes \mathbf{D})^* \mathbf{a}_{i,k}, \mathbf{f}_k \rangle|. \tag{4.19}$$

With this formulation, updating each $\mathbf{f}_k$ simplifies to solving a noisy r-dimensional phase retrieval problem with sampling matrix $\mathcal{M}_3(\underline{\mathbf{G}})(\mathbf{E} \otimes \mathbf{D})^* \mathbf{a}_{i,k}$. We can use any classical phase retrieval method to solve for $\mathbf{f}_k$, but we use RWF [42] to directly compare to AltMinLowRaP. This update step is summarized in Algorithm 3.

4.4 Numerical Experiments

We compare the performance of TSPR with two closely related algorithms, AltMinTrunc and AltMinLowRaP, using one synthetic dataset and two real video datasets, Mouse

and Plane. We consider measurements generated by real Gaussian matrices, complex Gaussian matrices, and coded diffraction patterns (CDP). To quantitatively compare these algorithms, we use the phase-invariant matrix distance [45] defined as

$$\text{mat-dist}^2(\hat{\mathbf{X}}, \mathbf{X}) = \sum_{k=1}^{q} \text{dist}^2(\hat{\mathbf{x}}_k, \mathbf{x}_k), \tag{4.20}$$

where $\mathbf{X}$ is the true matrix, $\hat{\mathbf{X}}$ is the reconstructed matrix and

$$\text{dist}(\hat{\mathbf{x}}, \mathbf{x}) = \min_{\phi \in [0, 2\pi]} \|\mathbf{x} - e^{\sqrt{-1}\phi}\hat{\mathbf{x}}\|. \tag{4.21}$$

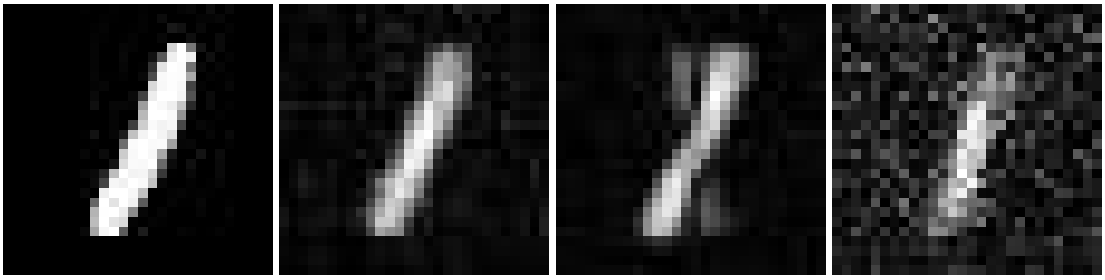
Note that the distance metric above is written in terms of the columns of the matrices $\mathbf{X}$ and $\hat{\mathbf{X}}$. Some of the results went through a "model correction" step as proposed by Nayer et al. [45]. The model correction step is running any phase retrieval algorithm for each $\mathbf{x}_k$ for $k = 1, \ldots, q$ given matrices $\mathbf{A}_k$ and measurements $\mathbf{y}_k$. This step can be viewed as the low-rank phase retrieval algorithm being a "warm start" for the classical phase retrieval algorithm, refining the signals where needed. We also provide a reconstruction of the videos as a supplement (available on Github) and display single frames in this thesis.

Experiments with Synthetic Data: The first experiment that we conducted was with the MNIST dataset [32], and it served to convince us that modelling the images as a tensor for the under-sampled regime was indeed a good idea. We took the first 50 images of the digit "1" from the MNIST dataset and stacked them to create a tensor $\underline{\mathbf{X}} \in \mathbb{R}^{28 \times 28 \times 50}$. We then generated measurements according to the model

$$\mathbf{y}_k = |\mathbf{A}_k^* \text{vec}(\mathbf{X}_k)|, \; k = 1, \ldots, q, \tag{4.22}$$

where each column of $\mathbf{A}_k$ was drawn from $\mathbf{a}_{i,k} \sim \mathcal{N}(0, \mathbf{I})$ (real Gaussian distribution). Upon recovering the tensor $\underline{\mathbf{X}}$ using TSPR, we observed that the reconstructed images were clearer than existing methods, along with the numerical errors being smaller. We display one frame of the reconstructed tensor in Figure 4.1 and show the numerical errors and number of measurements in Table 4.1.

Experiments with the Mouse Dataset: The mouse dataset is a video of a mouse moving slowly towards a camera, provided by Nayer et al. [45]. The mouse video



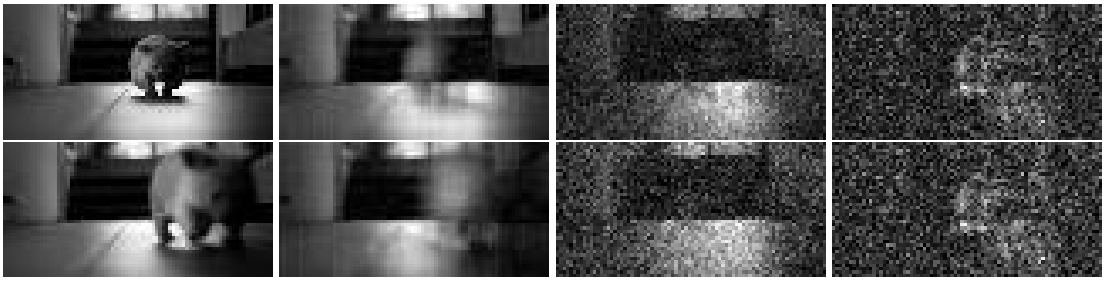
Original TSPR AltMinLowRaP AltMinTrunc

Figure 4.1: Results from recovering a stack of MNIST ones digits from real Gaussian measurements.

consisted of 90 frames, where each frame was downsized to be of dimensions 40×80. Upon constructing the tensor $\underline{\mathbf{X}} \in \mathbb{C}^{40\times80\times90}$, we generated measurements according to the model

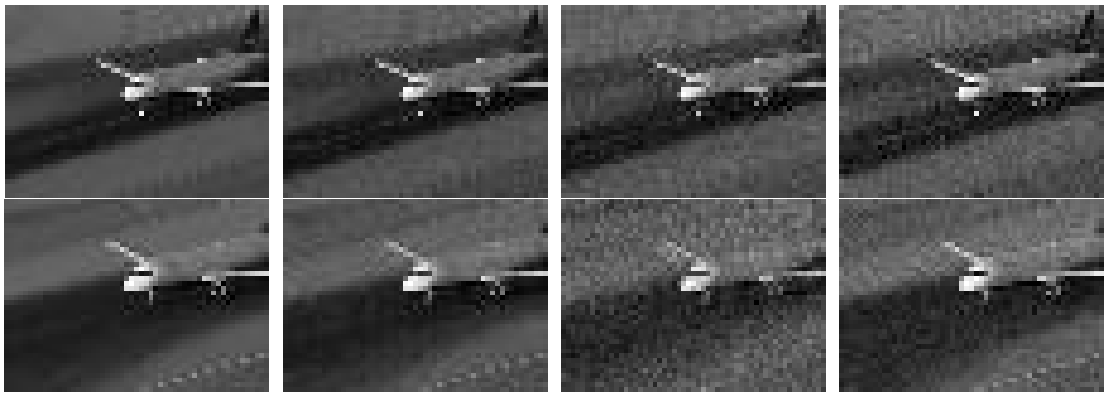
$$\mathbf{y}_k = |\mathbf{A}_k^* \text{vec}(\mathbf{X}_k)|,\ k = 1, \ldots, q, \tag{4.23}$$

where each column of $\mathbf{A}_k$ was drawn either from $\mathbf{a}_{i,k} \sim \mathcal{N}(0, \mathbf{I_n})$ or $\mathbf{a}_{i,k} \sim \mathcal{CN}(0, \mathbf{I_n})$ (circularly complex Gaussian distribution). We compare the three algorithms in two under-determined settings under these measurements. The numerical results are recorded in Table 4.1 with two of the reconstructed frames shown in Figure 4.2. In Table 4.1, we can see that TSPR outperformed the other two algorithms in both under-determined settings by estimating significantly less parameters. In fact, we observe that for two different ranks, AltMinTrunc did not converge and had a resulting error that was significantly higher than the others. These values were obtained by running $T = 20$ iterations of the total algorithm and $T_{RWF} = 25$ where applicable. We would like to note that each iteration of TSPR also runs several iterations of CGLS. For our experiments, we ran $T_{CGLS} = 50$ iterations, which results in a total of 1000 iterations, excluding the iterations from RWF. For the trimming threshold, we used a value of $\alpha = 3$, as suggested in TWF [41]. The ranks were generally chosen by trial and error, and the results did not go through a model correction step, as it seemed to increase the errors both numerically and visually. We would also like to note that even though TSPR yielded a lower numerical reconstruction error, we can see in Figure 4.2 that the reconstructed image is still not as clear as the original image. This is an intrinsic tradeoff of the Tucker model,



Original TSPR AltMinLowRaP AltMinTrunc

Figure 4.2: Results from recovering a video of a moving mouse from complex Gaussian measurements. Rows 1 and 2: reconstructed images of frames 60 and 70, respectively.



Original TSPR AltMinLowRaP AltMinTrunc

Figure 4.3: Results from recovering a video of a plane from CDP measurements. Rows 1 and 2: reconstructed images of frames 10 and 80, respectively.

as each frame may not be exactly low-rank. We want to choose the ranks corresponding to the image dimensions (i.e. r_1, r_2) to be small so that we can get convergence up to some modelling error, but not too small such that the reconstructed images are unclear. Based on our experiments, we observed that for ranks r_1 and r_2, using ranks slightly less than half of the dimensions of image (i.e. $r_1 < 0.5n_1$ and $r_2 < 0.5n_2$) worked well, whereas for r_3 (or r in the matrix model), we can be more conservative in our choices and choose a value much smaller.

Experiments with the Plane Dataset: The plane dataset is a video of a plane slowly landing on a runway, also provided by Nayer et al. [45]. The plane video consisted of 90 frames, where each frame was downsized to be of dimensions 40×55 for efficiency. Upon constructing the tensor $\underline{\mathbf{X}} \in \mathbb{C}^{40\times55\times90}$, we generated measurements according to

Experiment	Samples	Algorithm	Rank	Distance
MNIST	$m \approx 0.25n$	TSPR	$r = [10, 10, 3]$	0.460
(Real Gaussian)		AltMinLowRaP	$r = 3$	0.610
		AltMinTrunc	$r = 3$	0.780
Mouse	$m = 0.25n$	TSPR	$r = [20, 25, 5]$	2.851
(Real Gaussian)		AltMinLowRaP	$r = 5$	6.175
		AltMinTrunc	$r = 5$	7.277
Mouse	$m = 0.75n$	TSPR	$r = [20, 25, 5]$	1.217
(Complex Gaussian)			$r = [20, 25, 10]$	1.170
		AltMinLowRaP	$r = 5$	4.379
			$r = 10$	3.435
		AltMinTrunc	$r = 5$	78.118
			$r = 10$	77.319
Plane	$m = 2n$	TSPR	$r = [15, 20, 10]$	0.437
(CDP)			$r = [20, 25, 10]$	0.571
			$r = [30, 35, 10]$	1.008
		AltMinLowRaP	$r = 10$	0.869
		AltMinTrunc	$r = 10$	0.894

Table 4.1: Results for the experiments with the Mouse and Plane datasets. The value n refers to the dimensions of $\mathbf{x}_k$ and m refers to the number of measurements generated for each $\mathbf{x}_k$. The # of parameters value refers to the total number of parameters that need to be solved for all images $\mathbf{x}_k$. The distance metric is the phase-invariant distance defined in equation (4.20).

the CDP model

$$\mathbf{y}_{l,k} = |\tilde{\mathbf{F}}\mathbf{M}_l \text{vec}(\mathbf{X}_k)|, \, l = 1, \ldots, L, \, k = 1, \ldots q, \tag{4.24}$$

where $\tilde{\mathbf{F}}$ is the discrete Fourier transform (DFT) matrix and $\mathbf{M}$ is a diagonal mask matrix with elements drawn randomly from $\{1, -1, j, -j\}$. Since the CDP model can only generate measurements $m = Ln$ for each image for some integer L, the objective of this experiment was to show the effectiveness of TSPR in the over-determined setting. Upon running all three algorithms with the same parameters as the Mouse dataset, each result went through a model correction step. In Figure 4.3, we see that while all three algorithms can visually reconstruct the frames of this video, but Table 4.1 shows that the error for TSPR is significantly lower. However, the errors are only lower for certain values of the Tucker rank. This is most likely because as these ranks increase, the total number of parameters slowly converge to that of the unstructured methods, making recovery more difficult.

4.5 Conclusion and Future Work

In this chapter, we showed that by modeling the sequence of images as a tensor, we can obtain a more accurate reconstruction in both the under and over-sampled regimes. Our algorithm, TSPR, adopted a mixture of optimization techniques from AltMin-LowRaP and Truncated Wirtinger Flow to improve upon existing methods. TSPR involved a spectral initialization method that used higher-order SVD with alternating minimization via conjugate gradient least squares. Currently, TSPR lacks the theoretical guarantees in comparison to unstructured solutions. One important avenue for future research can be to extend our algorithm but with theoretical guarantees on the sample complexity required for accurate recovery. Our results show that there *exist* Tucker-structured models with better performance; we believe that perhaps finding a more principled approach for choosing these ranks is an important challenge for future work.

Bibliography

[1] J. Mairal, J. Ponce, G. Sapiro, A. Zisserman, and F. Bach. "Supervised Dictionary Learning". In: *Advances in Neural Information Processing Systems*. Vol. 21. 2008.

[2] E. J. Candès, X. Li, and M. Soltanolkotabi. "Phase Retrieval via Wirtinger Flow: Theory and Algorithms". In: *IEEE Transactions on Information Theory* 61.4 (2015), pp. 1985–2007. ISSN: 0018-9448. DOI: 10.1109/TIT.2015.2399924.

[3] P. Netrapalli, P. Jain, and S. Sanghavi. "Phase Retrieval Using Alternating Minimization". In: *Advances in Neural Information Processing Systems*. Vol. 26. 2013, pp. 2796–2804. URL: https://papers.nips.cc/paper/2013/file/242c100dc94f871b6d7215b868a875f8-Paper.pdf.

[4] A. Ahmed, B. Recht, and J. Romberg. "Blind Deconvolution Using Convex Programming". In: *IEEE Transactions on Information Theory* 60.3 (2014), pp. 1711–1732. DOI: 10.1109/TIT.2013.2294644.

[5] L. O'Donnell and C.-F. Westin. "An Introduction to Diffusion Tensor Image Analysis". In: *Neurosurgery Clinics of North America* 22 (2011), pp. 185–96, viii. DOI: 10.1016/j.nec.2010.12.004.

[6] K. Makantasis, A. D. Doulamis, N. D. Doulamis, and A. Nikitakis. "Tensor-Based Classification Models for Hyperspectral Data Analysis". In: *IEEE Transactions on Geoscience and Remote Sensing* 56.12 (2018), pp. 6884–6898. DOI: 10.1109/TGRS.2018.2845450.

[7] S. M. Kwon and A. D. Sarwate. "Learning Predictors from Multidimensional Data with Tensor Factorizations". In: *Aresty Rutgers Undergraduate Research Journal* 1 (Oct. 2021). DOI: 10.14713/arestyrurj.v1i3.165.

[8] S. M. Kwon and A. D. Sarwate. *Low-Rank Phase Retrieval with Structured Tensor Models*. Tech. rep. arXiv:2202.08260. arXiv, 2022. URL: https://arxiv.org/abs/2202.08260.

[9] T. G. Kolda and B. W. Bader. "Tensor Decompositions and Applications". In: *SIAM Review* 51.3 (2009), pp. 455–500. DOI: 10.1137/07070111X.

[10] M. Udell and A. Townsend. "Why Are Big Data Matrices Approximately Low Rank?" In: *SIAM Journal on Mathematics of Data Science* 1.1 (2019), pp. 144–160. DOI: 10.1137/18M1183480. URL: https://doi.org/10.1137/18M1183480.

[11] K. Kourou, T. P. Exarchos, K. P. Exarchos, M. V. Karamouzis, and D. I. Fotiadis. "Machine learning applications in cancer prognosis and prediction". In: *Computational and Structural Biotechnology Journal* 13 (2015), pp. 8–17. ISSN: 2001-0370. DOI: https://doi.org/10.1016/j.csbj.2014.11.005. URL: https://www.sciencedirect.com/science/article/pii/S2001037014000464.

[12] A. Das, S. Mishra, and S. Gopalan. "Predicting CoVID-19 community mortality risk using machine learning and development of an online prognostic tool". In: *PeerJ* 8 (2020), e10083. DOI: 10.7717/peerj.10083.

[13] L. Deng and X. Li. "Machine Learning Paradigms for Speech Recognition: An Overview". In: *IEEE Transactions on Audio, Speech, and Language Processing* 21.5 (2013), pp. 1060–1089. DOI: 10.1109/TASL.2013.2244083.

[14] E. Candès and B. Recht. "Exact Matrix Completion via Convex Optimization". In: *Commun. ACM* 55.6 (2012), pp. 111–119. ISSN: 0001-0782. DOI: 10.1145/2184319.2184343. URL: https://doi.org/10.1145/2184319.2184343.

[15] P. Jain, P. Netrapalli, and S. Sanghavi. "Low-Rank Matrix Completion Using Alternating Minimization". In: *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, 2013, pp. 665–674. DOI: 10.1145/2488608.2488693.

[16] V. Muthukumar, A. Narang, V. Subramanian, M. Belkin, D. J. Hsu, and A. Sahai. "Classification vs regression in overparameterized regimes: Does the loss function

matter?" In: *Journal of Machine Learning Research* 22 (2021), 222:1–222:69. URL: https://jmlr.org/papers/volume22/20-603/20-603.pdf.

[17] O. Shamir. "The Sample Complexity of Learning Linear Predictors with the Squared Loss". In: *Journal of Machine Learning Research* 16.1 (2015), pp. 3475–3486. ISSN: 1532-4435. URL: https://www.jmlr.org/papers/volume16/shamir15a/shamir15a.pdf.

[18] C. Jia, Y. Kong, Z. Ding, and Y. R. Fu. "Latent Tensor Transfer Learning for RGB-D Action Recognition". In: *Proceedings of the 22nd ACM International Conference on Multimedia*. Association for Computing Machinery, 2014, pp. 87–96. DOI: 10.1145/2647868.2654928. URL: https://doi.org/10.1145/2647868.2654928.

[19] N. Kreimer and M. Sacchi. "Tensor completion via nuclear norm minimization for 5D seismic data reconstruction". In: *Geophysics* 78 (2012), pp. 1–5. DOI: 10.1190/segam2012-0529.1.

[20] Y. Li, H. Zhu, D. Shen, W. Lin, J. Gilmore, and J. Ibrahim. "Multiscale Adaptive Regression Models for Neuroimaging Data". In: *Journal of the Royal Statistical Society. Series B, Statistical methodology* 73 (Sept. 2011), pp. 559–578. DOI: 10.1111/j.1467-9868.2010.00767.x.

[21] H. Zhou, L. Li, and H. Zhu. "Tensor Regression with Applications in Neuroimaging Data Analysis". In: *Journal of the American Statistical Association* 108.502 (2013), pp. 540–552. ISSN: 1537-274X. DOI: 10.1080/01621459.2013.776499. URL: http://dx.doi.org/10.1080/01621459.2013.776499.

[22] X. Li, H. Zhou, and L. Li. "Tucker Tensor Regression and Neuroimaging Analysis". In: *Statistics in Biosciences* 10 (Apr. 2013). DOI: 10.1007/s12561-018-9215-6.

[23] A. R. Zhang, Y. Luo, G. Raskutti, and M. Yuan. "ISLET: Fast and optimal low-rank tensor regression via importance sketching". In: *SIAM Journal on Mathematics of Data Science* 2.2 (2020), pp. 444–479. URL: https://doi.org/10.1137/19M126476X.

[24] M. Ghassemi, Z. Shakeri, A. D. Sarwate, and W. U. Bajwa. "Learning Mixtures of Separable Dictionaries for Tensor Data: Analysis and Algorithms". In: *IEEE Transactions on Signal Processing* 68.1 (2020), pp. 33–48. DOI: 10.1109/TSP.2019.2952046. URL: https://dx.doi.org/10.1109/TSP.2019.2952046.

[25] X. Tan, Y. Zhang, S. Tang, J. Shao, F. Wu, and Y. Zhuang. "Logistic Tensor Regression for Classification". In: Oct. 2012, pp. 573–581. ISBN: 978-3-642-36668-0. DOI: 10.1007/978-3-642-36669-7_70.

[26] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. New York, NY: Springer Publishing Company, Incorporated, 2014. ISBN: 1461471370.

[27] J. Bezdek and R. Hathaway. "Convergence of alternating optimization". In: *Neural, Parallel & Scientific Computations* 11 (2003), pp. 351–368. URL: https://dl.acm.org/doi/10.5555/964885.964886.

[28] S. Wright. "Coordinate Descent Algorithms". In: *Mathematical Programming* 151 (Feb. 2015). DOI: 10.1007/s10107-015-0892-3.

[29] E. Candès and B. Recht. "Exact Matrix Completion via Convex Optimization". In: *Communications of the ACM* 9 (Nov. 2008), pp. 717–772. DOI: 10.1007/s10208-009-9045-5.

[30] R. Tomioka and T. Suzuki. "Convex Tensor Decomposition via Structured Schatten Norm Regularization". In: *Advances in Neural Information Processing Systems* (2013).

[31] K. Wimalawarne, R. Tomioka, and M. Sugiyama. "Theoretical and Experimental Analyses of Tensor-Based Regression and Classification". In: *Neural Computation* 28 (Sept. 2015). DOI: 10.1162/NECO_a_00815.

[32] L. Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[33] H. Nguyen and L. Bai. "Cosine Similarity Metric Learning for Face Verification". In: *Asian Conference on Computer Vision* 6493 (2010), pp. 709–720. DOI: 10.1007/978-3-642-19309-5_55.

[34] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[35] R. P. Millane. "Phase retrieval in crystallography and optics". In: *Journal of the Optical Society of America A* 7.3 (1990), pp. 394–411. DOI: 10.1364/JOSAA.7.000394.

[36] G. Jagatap, Z. Chen, S. Nayer, C. Hegde, and N. Vaswani. "Sample Efficient Fourier Ptychography for Structured Data". In: *IEEE Transactions on Computational Imaging* 6 (2020), pp. 344–357. DOI: 10.1109/TCI.2019.2948758.

[37] J. Holloway, M. S. Asif, M. K. Sharma, N. Matsuda, R. Horstmeyer, O. Cossairt, and A. Veeraraghavan. "Toward Long-Distance Subdiffraction Imaging Using Coherent Camera Arrays". In: *IEEE Transactions on Computational Imaging* 2.3 (2016), pp. 251–265. DOI: 10.1109/TCI.2016.2557067.

[38] M. D. Butala, R. A. Frazin, Y. Chen, and F. Kamalabadi. "A Monte Carlo Technique for Large-Scale Dynamic Tomography". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* 3 (2007), pp. 1217–1220. DOI: 10.1109/ICASSP.2007.367062.

[39] R. Gerchberg. "A practical algorithm for the determination of phase from image and diffraction plane pictures". In: *Optik* 35 (1972), pp. 237–246.

[40] E. J. Candès, Y. C. Eldar, T. Strohmer, and V. Voroninski. "Phase Retrieval via Matrix Completion". In: *SIAM Review* 57.2 (2015), pp. 225–251. DOI: 10.1137/151005099.

[41] Y. Chen and E. Candès. "Solving Random Quadratic Systems of Equations Is Nearly as Easy as Solving Linear Systems". In: *Advances in Neural Information Processing Systems* 28 (2015). URL: https://proceedings.neurips.cc/paper/2015/file/7380ad8a673226ae47fce7bff88e9c33-Paper.pdf.

[42] H. Zhang, Y. Liang, and Y. Chi. "A Nonconvex Approach for Phase Retrieval: Reshaped Wirtinger Flow and Incremental Algorithms". In: *Journal of Machine Learning Research* 18.141 (2017), pp. 1–35. URL: http://jmlr.org/papers/v18/16-572.html.

[43] Y. Chen, Y. Chi, J. Fan, and C. Ma. *Spectral Methods for Data Science: A Statistical Perspective*. Tech. rep. arXiv:2012.08496v2 [stat.ML]. arXiv, 2021. URL: https://arxiv.org/abs/2012.08496.

[44] N. Vaswani, S. Nayer, and Y. C. Eldar. "Low-Rank Phase Retrieval". In: *IEEE Transactions on Signal Processing* 65 (2017), pp. 4059–4074. DOI: 10.1109/ICASSP.2017.7952997.

[45] S. Nayer, P. Narayanamurthy, and N. Vaswani. "Provable Low Rank Phase Retrieval". In: *IEEE Transactions on Information Theory* 66.9 (2020), pp. 5875–5903. DOI: 10.1109/TIT.2020.2984478.

[46] Z. Chen, G. Jagatap, S. Nayer, C. Hegde, and N. Vaswani. "Low Rank Fourier Ptychography". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2018), pp. 6538–6542. DOI: 10.1109/ICASSP.2018.8462480.

[47] S. Nayer, N. Vaswani, and Y. C. Eldar. "Low rank phase retrieval". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2017), pp. 4446–4450. DOI: 10.1109/ICASSP.2017.7952997.

[48] K. Liu, J. Wang, Z. Xing, L. Yang, and J. Fang. "Low-Rank Phase Retrieval via Variational Bayesian Learning". In: *IEEE Access* 7 (2019), pp. 5642–5648. DOI: 10.1109/ACCESS.2018.2889518.

[49] L. D. Lathauwer, B. D. Moor, and J. Vandewalle. "A Multilinear Singular Value Decomposition". In: *SIAM Journal on Matrix Analysis and Applications* 21.4 (2000), pp. 1253–1278. ISSN: 0895-4798. DOI: 10.1137/S0895479896305696. URL: https://doi.org/10.1137/S0895479896305696.