8-2018

# Computational Methods for Matrix/Tensor Factorization and Deep Learning Image Denoising

Joon Hee Choi
*Purdue University*

# COMPUTATIONAL METHODS FOR MATRIX/TENSOR FACTORIZATION AND DEEP LEARNING IMAGE DENOISING

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Joon Hee Choi

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2018

Purdue University

West Lafayette, Indiana

# THE PURDUE UNIVERSITY GRADUATE SCHOOL
# STATEMENT OF DISSERTATION APPROVAL

Dr. Stanley Chan, Chair

School of Electrical and Computer Engineering

Dr. Charles A. Bouman

School of Electrical and Computer Engineering

Dr. Mireille Boutin

School of Electrical and Computer Engineering

Dr. Amy R. Reibman

School of Electrical and Computer Engineering

**Approved by:**

Dr. Venkataramanan Balakrishnan

Head of the School Graduate Program

To My Lovely Wife and Daughter, My Parents and My Parents in Law.

# ACKNOWLEDGMENTS

It feels surreal to be writing the acknowledgement section. I returned to academia after eight years of work experience as a project manager, which placed me on a challenging journey in the doctoral program. Alongside my much younger peers, I struggled mentally and physically to relearn and update my academic knowledge and skills in computer engineering. Therefore, I know I would not have been able to reach this last milestone in my PhD journey without the help and patience from so many generous and incredible individuals I have met along the way. I would like to express my sincerest gratitude to all of them.

First, I would like to extend my deepest appreciation and gratitude to my advisor and mentor, Professor Stanley Chan. I have been extremely fortunate to have met Professor Chan at a time when I was without direction and hope. Not only did he graciously take me under his wings, but he has been infinitely supportive, inspiring, and genuinely caring for the last three and a half years. Professor Chan genuinely cares about his students and respects the unique views and research interests each of us brings to the table. What makes him extraordinary, and myself extremely lucky, is that he provides firm structure and a nurturing environment for his students so that we can maximize productivity as well as balance life. Through good times and bad, he would always listen, patiently work with me to find solutions, and encourage me to pursue my passion and dreams. I never knew how to express my deep gratitude but I want to use this space to let him know that he is the best mentor I have met and that I would not have been able to complete my PhD journey in such a meaningful way had it not been for him.

Professor SVN Vishwanathan is another person to whom I am deeply indebted. Professor Vishwanathan was my first advisor at Purdue before he left the school. He accepted me as his student when I lacked basic knowledge and skills in computer

programming. He taught me everything from coding to research. In the end, I was able to achieve the unthinkable—having my paper accepted to the prestigious Neural Information Processing Systems (NIPS). Professor Vishwanathan transformed me from a bewildered older student to a confident and hopeful doctoral student.

I am deeply grateful to my committee members, Professor Charles A. Bouman, Professor Mireille Boutin and Professor Amy R. Reibman, for their guidance and insightful discussions on my research. It is a great honor to have my dissertation work vetted by such renowned scholars in the fields of image processing.

I would also like to acknowledge the helpful guidance and support I have received from Dr. Abhinav Vishnu at Pacific Northwest National Laboratory (PNNL) during my internship; the collegiality and support from my labmates Omar Elgendy, Xiran Wang, Xiangyu Qu, Zhiyuan Mao, Nicholas Chimitt, Yiheng Chi, Pinar Yanardag, Jiasen Yang, Hyokun Yun, Bin Shen and Nan Ding; and the deep friendship and love from my fellow Korean colleagues Chiho Choi, Soonam Lee, Woosuk Lee, Song Noh, Hyundok Cho, Minwoong Kim, Yunsung Nam, Joonsoo Kim, Kangwoo Lee and Heejin Park, and my friends in South Korea Sungbae Cho, Youngsuk Kim and Kiyong Kim. Thanks to everyone for helping me through difficult times and laughing with me during exciting moments in my life.

Last but not least, I would like to express my deepest love and appreciation to my family, who mean everything to me: My lovely wife Na Youn Lee, my dearest daughter Jianne Choi, my self-sacrificing parents, my generous parents-in-law, and my cool brothers Kun Hee and Tae Woo.

This dissertation was made possible thanks to “all” of your strong support. Thank you again.

TABLE OF CONTENTS

Page

Page

Page

# LIST OF TABLES

# LIST OF FIGURES

Figure Page

# ABSTRACT

Choi, Joon Hee PhD, Purdue University, August 2018. Computational Methods for Matrix/Tensor Factorization and Deep Learning Image Denoising . Major Professor: Stanley Chan.

Feature learning is a technique to automatically extract features from raw data. It is widely used in areas such as computer vision, image processing, data mining and natural language processing. In this thesis, we are interested in the computational aspects of feature learning. We focus on rank matrix and tensor factorization and deep neural network models for image denoising.

With respect to matrix and tensor factorization, we first present a technique to speed up alternating least squares (ALS) and gradient descent (GD) − two commonly used strategies for tensor factorization. We introduce an efficient, scalable and distributed algorithm that addresses the data explosion problem. Instead of a computationally challenging sub-step of ALS and GD, we implement the algorithm on parallel machines by using only two sparse matrix-vector products. Not only is the algorithm scalable but it is also on average 4 to 10 times faster than competing algorithms on various data sets. Next, we discuss our results of non-negative matrix factorization for hyperspectral image data in the presence of noise. We introduce a spectral total variation regularization and derive four variants of the alternating direction method of multiplier algorithm. While all four methods belong to the same family of algorithms, some perform better than others. Thus, we compare the algorithms using stimulated Raman spectroscopic image will be demonstrated.

For deep neural network models, we focus on its application to image denoising. We first demonstrate how an optimal procedure leveraging deep neural networks and convex optimization can combine a given set of denoisers to produce an overall

better result. The proposed framework estimates the mean squared error (MSE) of individual denoised outputs using a deep neural network; optimally combines the denoised outputs via convex optimization; and recovers lost details of the combined images using another deep neural network. The framework consistently improves denoising performance for both deterministic denoisers and neural network denoisers. Next, we apply the deep neural network to solve the image reconstruction issues of the Quanta Image Sensor (QIS), which is a single-photon image sensor that oversamples the light field to generate binary measures.

# 1. INTRODUCTION

Modern data analysis is surround by two basic problems: (1) How to efficiently extract meaningful information from a massive volume of data? (2) How does the massive data improve classical tasks such as estimating signals embedded in noise? The former question concerns about the *algorithm* aspect of large-scale data analysis. As we will see in this thesis, many well-established computational methods in linear algebra become computationally inadequate when the data volume grows. The second question concerns about the *performance* of estimation methods when data becomes abundant. The goal of this thesis is to address these two problems from two specific angles. First, we study a factorization problem for multidimensional arrays (called the tensors), and develop an efficient algorithm that can be executed on distributed machines. We then analyze how noise would influence the factorization, and methods to improve robustness. Second, we study a regression problem using deep neural networks. We find that despite the abundance of data, many neural networks are trained under specific conditions, thus limiting their performance. We thus develop a globally optimal method to combine these neural network estimators. We also investigate how deep neural networks can help improving image reconstruction on a new type of image sensors.

## 1.1 Objective 1: Efficient Matrix and Tensor Factorization Methods

Matrix factorization is a widely used unsupervised learning technique that allows us to decompose a matrix into products of feature matrices. Traditionally, matrix factorization has been the backbone of many signal processing techniques such as the LU decomposition, QR decomposition and singular value decomposition (SVD). Of all these methods, we are particularly interested in the class of rank factorization.

Rank factorization is typically used to provide low-rank matrix approximation of a matrix so that we can infer meaningful features embedded in the matrix. Given an $m \times n$ matrix $\boldsymbol{Y}$ of rank $r$, a rank factorization is a product of $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{H}$ where $\boldsymbol{W}$ is an $m \times r$ matrix and $\boldsymbol{H}$ is an $r \times n$ matrix. These rank factorization can also be represented as a form of the bilinear model

$$\boldsymbol{Y} = \sum_{i=1}^{r} \boldsymbol{w}_i \circ \boldsymbol{h}_i, \tag{1.1}$$

where "$\circ$" denotes the outer product of two vectors. Thus, we can build an approximate representation of the data matrix $\boldsymbol{Y}$ as a sum of rank-one matrices.

The focus of the first part of this thesis is the *tensor factorization*, an extension of matrix factorization to multidimensional arrays. For a three-dimensional array, tensor factorization considers the approximation

$$\mathcal{Y} \approx \sum_{i=1}^{r} \boldsymbol{a}_i \circ \boldsymbol{b}_i \circ \boldsymbol{c}_i \tag{1.2}$$

where "$\circ$" denotes the outer product of vectors and $\mathcal{Y}$ is a three-dimensional array of size $m \times n \times k$.

Tensor factorization appears naturally in applications involving multi-modal observations. For example, in hyperspectral imaging we acquire data with two spatial dimensions and a spectral dimension. (See Figure 1.1). A typical question is then to evaluate the composition of the material shown in the image, e.g., chemical components of soil. In other applications such as recommendation systems, tensor factorization is also common. For example, when analyzing a recommendation system involving users, items and their evolution over time, we have to consider a three-dimensional array consisting of a user dimension, an item dimension, and a temporal dimension. As the size of the array increases, the computational cost to perform the above stated decompositions will eventually become so high that we will not be able to process the data quickly without sacrificing the quality.

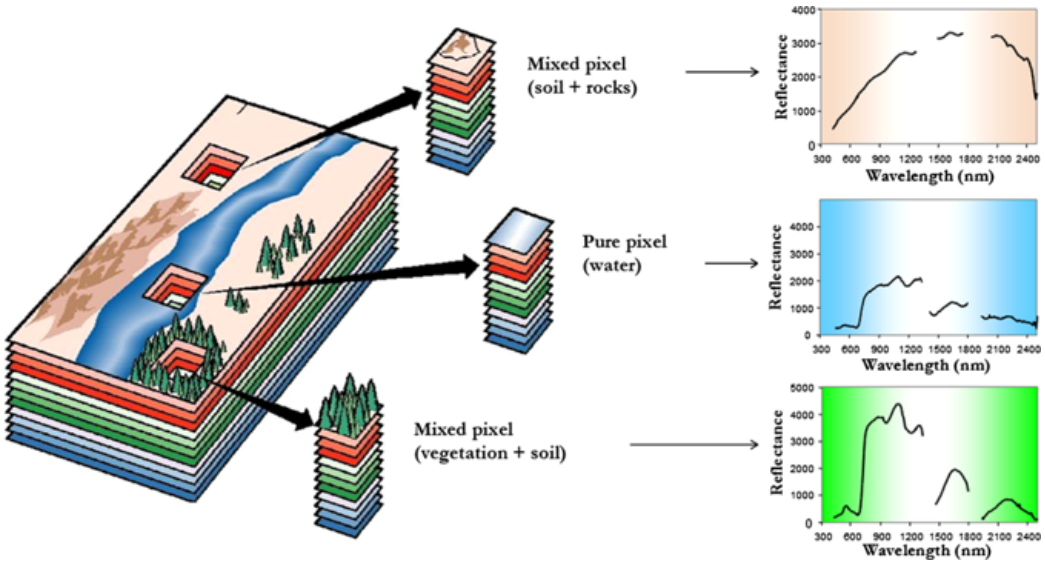



Fig. 1.1.: Hyperspectral image data [1]

There are two specific sub-objectives of Part 1 of the thesis. We summarize them in the followings.

**Objective 1(a)**: The objective is to develop an efficient tensor factorization that can be scaled to billions of non-zero entries. At a high level, our proposed method considers a flatting technique by rewriting the tensor $\mathcal{Y}$ as a matrix $\boldsymbol{Y}^1 \in \mathbb{R}^{mn\times k}$, and decompose $\boldsymbol{Y}^1$ as

$$\boldsymbol{Y}^1 \approx \boldsymbol{A}\left(\boldsymbol{B}\odot\boldsymbol{C}\right)^T. \tag{1.3}$$

where "$\odot$" is a Khatri-Rao product (see Chapter 2 for the detail) and $(\boldsymbol{B}\odot\boldsymbol{C}) \in \mathbb{R}^{mk\times r}$. Typically, $(\boldsymbol{B}\odot\boldsymbol{C})$ is very large, and so it poses significant computational challenge when computing $\boldsymbol{A}\left(\boldsymbol{B}\odot\boldsymbol{C}\right)^T$. Algorithms addressing these computational challenges have been studied in recent years. Bader and Kolda [2] proposed a state-of-art method solving the data explosion problem. Also, Kang et al. [3] proposed an algorithm addressing the computational challenges on parallel machines. We tackle this problem by exploring special structures of the matrices so that sparse computation techniques can be employed. We will discuss this method in Chapter 2. We com-

pare our method with other methods addressing the computational challenges [2, 3] in terms of computation time.

**Objective 1(b)**: The objective here is to study the algorithmic aspect of a slightly more specialized matrix factorization problem called the *non-negative matrix factorization* (NMF). The goal NMF is to factorize a matrix $\boldsymbol{Y}$ as the product of two non-negative matrices $\boldsymbol{W}$ and $\boldsymbol{H}$. Typically, this problem is formulate through a minimization problem

$$\begin{aligned} &\underset{\boldsymbol{W},\boldsymbol{H}}{\text{minimize}} \ \|\boldsymbol{Y} - \boldsymbol{W}\boldsymbol{H}\|_F^2, \\ &\text{subject to } \boldsymbol{W} \geq 0, \ \boldsymbol{H} \geq 0 \end{aligned} \tag{1.4}$$

To solve (1.4), there are many existing approaches, e.g., the multiplicative update method [4], block principal pivoting method [5], and different online approaches [6–8]. Because of the bilinear quadratic minimization form of (1.4), one of the mainstream approaches is the alternating nonnegative least squares [9, 10], and other alternating direction algorithms [11]. In this part of the thesis, we consider a special alternating direction algorithm. We show that even for the same alternating direction algorithm, there are various options of controlling how variables are updated. We analyze their performance, and compare their complexity.

## 1.2 Objective 2: Deep Learning for Image Denoising

The second part of the thesis is to investigate a class of deep neural network methods for the task of image restoration, in particular denoising type of problems. Here, by denoising we meant that given a noisy observation $\boldsymbol{y} \in R^n$, which is assumed to be corrupted by i.i.d. Gaussian noise:

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{\eta},$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the clean image, our goal is to estimate the latent image $\boldsymbol{x}$. Image denoising is a testbed for a number of image restoration problems. Therefore, studying image denoising has important implication to other types of image restoration problems.

**Objective 2(a)**: The objective here is to develop a globally optimal combination approach to combine neural network image denoisers.

The latest development of image denoising has been focusing on deep learning based methods, e.g., [12–18]. While these methods have demonstrated superior performance over the deterministic counterparts [19–23], they suffer fundamentally to a problem known as the model mismatch. In particular, there are three model mismatch problems we often see in practice:

- Denoiser Characteristic: Different denoiser models have different characteristics. For example, total variation works well for piecewise constant images and BM3D [20] works well for images with repeated patterns.
- Noise Level: Neural network image denoisers are typically trained under specific noise levels. However, when the actual noise level deviates from the trained noise level, the performance of the denoisers drop.
- Image Class: A denoiser trained for a particular class of images may work well for the class and not work for other classes.

Therefore, the following question should be answered: Given a set of image denoisers, each having a different denoising capability, is there a provably optimal way of combining these denoisers to produce an overall better result? An answer to this question is fundamental to designing ensembles of weak estimators for complex scenes. In Chapter 4, we present an optimal procedure leveraging deep neural networks and convex optimization. The proposed framework, called the Consensus Neural Network (CsNet), introduces three new concepts in image denoising: (1) A deep neural network to estimate the mean squared error (MSE) of individual denoised outputs without needing the ground truth; (2) A provably optimal procedure to combine the

denoised outputs via convex optimization; (3) An image boosting procedure using a deep neural network to improve contrast and to recover lost details of the combined images.

**Objective 2(b)**: The objective here is to present a set of preliminary results for a new type of single-photon image sensors called the Quanta Image Sensor (QIS).

QIS is envisioned to be a candidate for the next generation image sensor after CMOS. However, image reconstruction of the sensor is faced with a problem the data is binary: if photon count is below the threshold, the value is 0; if above, the value 1. Most of the existing methods for the image reconstruction of QIS are based on optimization such as maximum-likelihood (ML) or maximum a-posteriori (MAP) estimation, but those are very slow. A recent research [24] converted the Poisson random variable to Gaussian then applied an existing image denoising method instead of solving complex ML or MAP. We apply a deep neural network to solve the image reconstruction issues of QIS. In Chapter 5, we present our preliminary findings on this topic. Experimental results show that the proposed network produces significantly better reconstruction results compared to existing methods.

### 1.3 Contributions of The Thesis

There are four main contributions of this thesis. They are listed as follows:

- Objective 1(a): Distributed Factorization of Tensors

  We propose a technique for significantly speeding up Alternating Least Squares (ALS) and Gradient Descent (GD), two widely used algorithms for tensor factorization. By exploiting properties of the Khatri-Rao product, we show how to efficiently address a computationally challenging sub-step of both algorithms. Our algorithm, DFacTo, only requires two sparse matrix-vector products and is easy to parallelize. DFacTo is not only scalable but also on average 4 to 10 times faster than competing algorithms on a variety of datasets. For instance, DFacTo only takes 480 seconds on 4 machines to perform one iteration of the

ALS algorithm and 1,143 seconds to perform one iteration of the GD algorithm on a 6.5 million $\times$ 2.5 million $\times$ 1.5 million dimensional tensor with 1.2 billion non-zero entries.

- Objective 1(b): Comparison of ADMM Algorithms for Noisy Non-negative Matrix Factorization

  We present a set of empirical results of non-negative matrix factorization for hyperspectral image data in the presence of noise. We introduce a spectral total variation regularization, and derive four variants of the alternating direction method of multiplier algorithm. While all four methods belong to the same family of algorithms, we show that some performs better than others. Comparisons of the algorithms using stimulated Raman spectroscopic image will be demonstrated.

- Objective 2(a): Optimal Combination of Image Denoisers

  We present an optimal framework combining multiple weak image denoisers, called Consensus Neural Network (CsNet). The framework consists of three components: MSE Estimator; Optimal Combination; and Denoising Dooster. CsNet first uses a novel deep neural network for estimating MSE of the denoised images from a set of initial image denoisers. Then, using the estimated MSE, CsNet solves a convex optimization problem where the optimality is guaranteed and combine the denoised images. Finally, the combined estimate is boosted by a novel deep neural network booster. Experimental results show CsNet outperforms other state-of-the-art denoising algorithms on tasks including: overcoming noise level mismatch combining denoisers for different image classes combining different denoiser types

- Objective 2(b): Image Reconstruction for Quanta Image Sensors using Deep Neural Networks

Unlike the existing image reconstruction algorithms based on optimization, we present the first deep neural network approach for QIS image reconstruction. Our deep neural network takes the binary bit stream of QIS as input, learns the nonlinear transformation and denoising simultaneously. Different from [25] which assumes a sparsity prior, our network learns the denoiser directly; And compared to [26], our network has a significantly deeper layer to learn the transformation. We present two designs: one mimics the entire Transform-Denoise pipeline, and the other one substitutes part of the Transform-Denoise pipeline. We show that both networks has significantly better performance than the existing Transform-Denoise method [24].

## 1.4 Publications Resulting from this Thesis

The following papers are results of this thesis.

1. **Joon Hee Choi**, Omar Elgendy and Stanley Chan, Integrating Disparate Sources of Experts for Robust Image Denoising, *submitted to IEEE Transaction on Image Processing (TIP)*, 2018.

2. **Joon Hee Choi**, Omar Elgendy and Stanley Chan, Image Reconstruction for Quanta Image Sensors Using Deep Neural Networks, *to appear in International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2018.

3. Chiman Kwan, **Joon Hee Choi**, Stanley Chan, Jin Zhou and Bence Budavari, Resolution Enhancement for Hyperspectral Images: A Super-Resolution and Fusion Approach, *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp 6180-6184, 2017.

4. Chien-Sheng Liao, **Joon Hee Choi**, Delong Zhang, Stanley H. Chan and Ji-Xin Cheng. Denoising Stimulated Raman Spectroscopic Images by Total Variation Minimization, *Journal of Physical Chemistry C*, 119 (33), pp. 19397-19403, July, 2015.

5. Chiho Choi, Ayan T. Sinha, **Joon Hee Choi**, Sujin Jang and Karthik Ramani, Collaborative Filtering Approach to Real-Time Hand Pose Estimation, *The IEEE International Conference on Computer Vision (ICCV)*, pp. 2336-2344, 2015.

6. **Joon Hee Choi** and S.V.N. Vishwanathan, DFacTo: Distributed Factorization of Tensors, *Advances in Neural Information Processing Systems (NIPS)*, pp. 1296-1304, 2014.

# PART I

# MATRIX AND TENSOR FACTORIZATION

# 2. DFACTO: DISTRIBUTED FACTORIZATION OF TENSORS

In this chapter, we address a computational challenge of tensor factorization by presenting a distributed algorithm. We compare our method with three existing methods. This work was published in Advances in Neural Information Processing Systems (NIPS), 2014 [27].

## 2.1 Introduction

### 2.1.1 Matrix, Tensor, and their Applications

In classical linear algebra, a matrix can be considered as a rectangular array consisting of *rows* and *columns* of numbers. Because of the planner structure of the array, mathematical properties of matrices are widely studied, e.g., inverses, norms, symmetry, spectral decomposition, and computational techniques, [28, 29]. For a long time, matrix-based techniques has been the backbone of many engineering and signal processing subjects, e.g., filter bank design [30, 31], optimal control [32], and many other disciplines. However, as we encounter more complicated data in modern data analytic, e.g., data volumes involving high-dimensional arrays (arrays with dimension more than just rows and columns), matrices becomes inadequate.

We refer to high-dimensional arrays with finite number of entries as *tensors* in this thesis. This is a narrow defined scope of a tensor, as this thesis does not consider tensors in advanced calculus, e.g., Ricci Calculus in general relativity and differential geometry [33]. Instead, we consider a tensor as a cubicle storing numbers. Such data analytic perspective was first discussed by [34] in a psychometric journal, and Sanchez and Kowalski in an analytic chemistry journal [35]. In modern data science appli-

cations, such analytic perspective is becoming more popular [36, 37]. For instance, on a social network evolving over time, one can form a users-users-time tensor which contains snapshots of interactions between members of the social network [38]. Another example would be an online store such as Amazon.com where users routinely review various products. One can form a users- items-words tensor from the review text [39]. A tensor can also be formed by considering the various contexts in which a user has interacted with an item [40]. One last example: a tensor can be formed by considering the data collected by the Never Ending Language Learner (the Read the Web) project which contains triples of noun phrases and the context in which they occur, such as, ("George Harrison", "plays", "guitars") [41]. [42] considered a tensor data to represent regions of a particular density in a three-dimensional CT scan, allowing the visualization of internal organs, bones, or other structures.

### 2.1.2 Tensor Factorization

**Objective**: The objective of this study is to investigate efficient algorithms for factorizing tensors into low rank products. Rank factorization is an important data analysis tool that provides low-rank approximation so that we can extract features. Before we discuss our proposed methods, we first discuss the difference between tensor factorization and matrix factorization.

Given an $m$-by-$n$ matrix $\boldsymbol{Y}$ of rank $r$, a matrix rank factorization is a product of $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{H}$ where $\boldsymbol{W}$ is an $m$-by-$r$ matrix and $\boldsymbol{H}$ is an $r$-by-$n$ matrix. Rank factorization can also be represented as a form of the bilinear model (see Figure 2.1)

$$\boldsymbol{Y} = \sum_{i=1}^{r} \boldsymbol{w}_i \circ \boldsymbol{h}_i, \tag{2.1}$$

where $\circ$ denotes the outer product of two vectors. Thus, we can build an approximate representation of the data matrix $\boldsymbol{Y}$ as a sum of rank-one matrices.

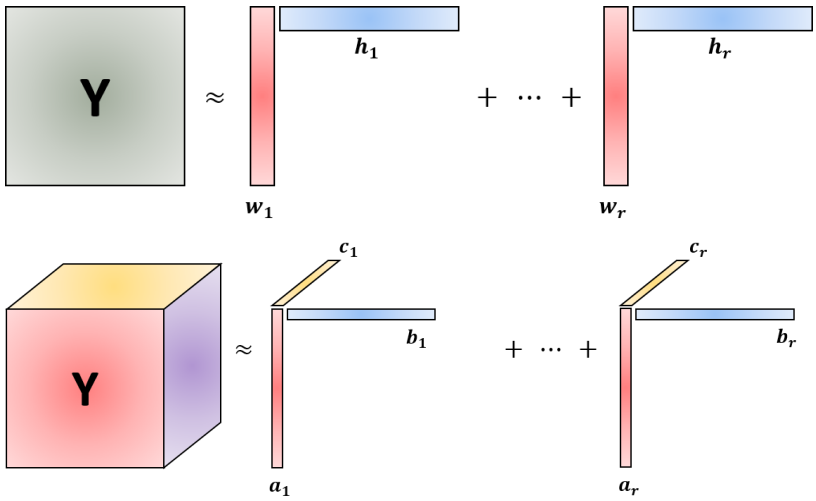



Fig. 2.1.: [Top] Matrix-factorization. [Bottom] Tensor-factorization.

Tensor factorization extends the matrix factorization to multidimensional arrays. In this thesis, we are particularly interested in the *three-way tensor* factorization, which considers

$$\mathcal{Y} \approx \sum_{i=1}^{r} \boldsymbol{a}_i \circ \boldsymbol{b}_i \circ \boldsymbol{c}_i \tag{2.2}$$

where $\circ$ denotes the outer product of vectors, $\mathcal{Y}$ is of size $m \times n \times k$, $\boldsymbol{A}$ of size $m \times r$, $\boldsymbol{B}$ of size $n \times r$ and $\boldsymbol{C}$ of size $k \times r$. Figure 2.1 shows this tensor factorization graphically. Decomposition of this type is also called the CANDECOMP or the PARAFAC decomposition [37,43–45]. An alternative to PARAFAC is the High-Order SVD (HOSVD) or the Tucker-3 decomposition, see, e.g., [46–52]. In this thesis, we shall focus on PARAFAC due to is applicability in many engineering problems.

**Challenges**: The challenges of tensor factorization can be understood from three aspects. First, the rank of a tensor is ill-posed. Using the PARAFAC decomposition, a tensor can be written as a sum of rank-one outer products of vectors. The smallest number of these rank-one prodducts, $R$, is called the rank of the tensor. However, while writing a matrix as a sum of rank-1 outer products (i.e., SVD) is always possible, writing a tensor as a sum of rank-1 outer products (i.e., PARAFAC) is not

always possible. Therefore, unlike matrices, the rank of a tensor $\mathcal{Y} \in \mathbb{R}^{m \times n \times k}$ is not necessarily $\min(m, n, k)$. See [53–56] for further discussions. This thesis does not address the rank issue. Instead, we will focus on developing efficient algorithms.

Second, unlike matrices where we have standard decomposition methods, e.g., QR factorization and LU factorization, tensors do not have well-defined decomposition methods. In fact, most of the existing algorithms are based on minimizing certain energy functions by alternating minimizations [57–60]. However, since the optimization problem is non-linear, the convergence to a global minimum is generally not guaranteed [61]. It might have a local minimum where the cost function ceases to decrease. Some techniques for improving the guess are discussed in [62]. In this thesis, we will focus on two classes of minimization techniques: the Alternating Least Squares (ALS) and the gradient descent (GD). We are specially interested in how these algorithms can be speed up for large-tensors.

Third, as data size grows, the number of entries of tensors also grow. This problem is exacerbated when the dimensions of tensor we need to factorize are very large (of the order of hundreds of thousands or millions), or when sparse tensors contain millions to billions of non-zero entries. For instance, a tensor we formed using review text from Amazon.com has dimensions of 6.5 million -by- 2.5 million -by- 1.5 million and contains approximately 1.2 billion non-zero entries.

### 2.1.3 Related Methods

The mainstream tensor factorization optimization methods can be classified into two categories: The Alternating Least Squares (ALS) [58, 61, 63, 64], and Gradient Descent (GD) [62,65]. We will discuss these two methods in details in Section A.2 and Section A.3, respectively. In a nutshell, the key step in both methods is to multiply a matricized tensor and a Khatri-Rao product of two matrices (line 4. However, this process leads to a computationally-challenging, intermediate data explosion problem when the data size is large.

Some studies have identified this intermediate data explosion problem and have suggested ways of addressing it. First, the Tensor Toolbox [2] uses the method of reducing indices of the tensor for sparse datasets and entrywise multiplication of vectors and matrices for dense datasets. However, it is not clear how to store data or how to distribute the tensor factorization computation to multiple machines (see Appendix). That is, there is a lack of *distributable* algorithms in existing studies. Another possible strategy to solve the data explosion problem is to use GigaTensor [3]. Unfortunately, while GigaTensor does address the problem of parallel computation, it is relatively slow. To summarize, existing algorithms for tensor factorization such as the excellent Tensor Toolbox of [2], or the Map-Reduce based GigaTensor algorithm of [3] often do not scale to large problems.

### 2.1.4 Our Contributions

In this chapter, we introduce an efficient, scalable and distributed algorithm, DFacTo, that addresses the data explosion problem. Since most large-scale real datasets are sparse, we will focus exclusively on sparse tensors. This is well justified because previous studies have shown that designing specialized algorithms for sparse tensors can yield significant speedups [2]. We show that DFacTo can be applied to both ALS and GD, and naturally lends itself to a distributed implementation. Therefore, it can be applied to massive real datasets which cannot be stored and manipulated on a single machine. For ALS, DFacTo is on average around 5 times faster than GigaTensor and around 10 times faster than the Tensor Toolbox on a variety of datasets. In the case of GD, DFacTo is on average around 4 times faster than CP-OPT [66] from the Tensor Toolbox. On the Amazon.com review dataset, DFacTo only takes 480 seconds on 4 machines to perform one iteration of ALS and 1,143 seconds to perform one iteration of GD.

As with any algorithm, there is a trade-off: DFacTo uses 3 times more memory than the Tensor Toolbox, since it needs to store 3 flattened matrices as opposed to

a single tensor. However, in return, our algorithm only requires two sparse matrix-vector multiplications, making DFacTo easy to implement using any standard sparse linear algebra library. Therefore, there are two merits of using our algorithm: 1) computations are distributed in a natural way; and 2) only standard operations are required.

## 2.2 Notation and Preliminaries

Our notation is standard, and closely follows [37]. Also see [36]. Lower case letters such as $x$ denote scalars, bold lower case letters such as $\mathbf{x}$ denote vectors, bold upper case letters such as $\mathbf{X}$ represent matrices, and calligraphic letters such as $\mathcal{X}$ denote three-dimensional tensors.

The $i$-th element of a vector $\mathbf{x}$ is written as $x_i$. In a similar vein, the $(i, j)$-th entry of a matrix $\mathbf{X}$ is denoted as $x_{i,j}$ and the $(i, j, k)$-th entry of a tensor $\mathcal{X}$ is written as $x_{i,j,k}$. Furthermore, $\mathbf{x}_{i,:}$ (resp. $\mathbf{x}_{:,i}$) denotes the $i$-th row (resp. column) of $\mathbf{X}$. We will use $\mathbf{X}_{\Omega,:}$ (resp. $\mathbf{X}_{:,\Omega}$) to denote the sub-matrix of $\mathbf{X}$ which contains the rows (resp. columns) indexed by the set $\Omega$. For instance, if $\Omega = \{2, 4\}$, then $\mathbf{X}_{\Omega,:}$ is a matrix which contains the second and fourth rows of $\mathbf{X}$. Extending the above notation to tensors, we will write $\mathbf{X}_{i,:,:}$, $\mathbf{X}_{:,j,:}$ and $\mathbf{X}_{:,:,k}$ to respectively denote the horizontal, lateral and frontal *slices* of a third-order tensor $\mathcal{X}$. The column, row, and tube *fibers* of $\mathcal{X}$ are given by $\mathbf{x}_{:,j,k}$, $\mathbf{x}_{i,:,k}$, and $\mathbf{x}_{i,j,:}$ respectively.

Sometimes a matrix or tensor may not be fully observed. We will use $\Omega^{\mathbf{X}}$ or $\Omega^{\mathcal{X}}$ respectively to denote the set of indices corresponding to the observed (or equivalently non-zero) entries in a matrix $\mathbf{X}$ or a tensor $\mathcal{X}$. Extending this notation, $\Omega^{\mathbf{X}}_{i,:}$ (resp. $\Omega^{\mathbf{X}}_{:,j}$) denotes the set of column (resp. row) indices corresponding to the observed entries in the $i$-th row (resp. $j$-th column) of $\mathbf{X}$. We define $\Omega^{\mathcal{X}}_{i,:,:}$, $\Omega^{\mathcal{X}}_{:,j,:}$, and $\Omega^{\mathcal{X}}_{:,:,k}$ analogously as the set of indices corresponding to the observed entries of the $i$-th horizontal, $j$-th lateral, or $k$-th frontal slices of $\mathcal{X}$. Also, $nnzr(\mathbf{X})$ (resp. $nnzc(\mathbf{X})$) denotes the number of rows (resp. columns) of $\mathbf{X}$ which contain at least one non-zero element.

$\mathbf{X}^\top$ denotes the transpose, $\mathbf{X}^\dagger$ denotes the Moore-Penrose pseudo-inverse, and $\|\mathbf{X}\|$ (resp. $\|\mathcal{X}\|$) denotes the Frobenius norm of a matrix $\mathbf{X}$ (resp. tensor $\mathcal{X}$) [67]. Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$, the linear operator $vec(\mathbf{A})$ yields a vector $\mathbf{x} \in \mathbb{R}^{nm}$, which is obtained by stacking the columns of $\mathbf{A}$. On the other hand, given a vector $\mathbf{x} \in \mathbb{R}^{nm}$, the operator $unvec_{(n,m)}(\mathbf{x})$ yields a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$.

$\mathbf{A} \otimes \mathbf{B}$ denotes the Kronecker product, $\mathbf{A} \odot \mathbf{B}$ the Khatri-Rao product, and $\mathbf{A} * \mathbf{B}$ the Hadamard product of matrices $\mathbf{A}$ and $\mathbf{B}$. The outer product of vectors $\mathbf{a}$ and $\mathbf{b}$ is written as $\mathbf{a} \circ \mathbf{b}$ (see *e.g.*, [68]). Definitions of these standard matrix products can be found in Appendix A.1:

### 2.2.1 Flattening Tensors

Just like the $vec(\cdot)$ operator flattens a matrix, a tensor $\mathcal{X}$ may also be unfolded or flattened into a matrix in three ways namely by stacking the horizontal, lateral, and frontal slices. We use $\mathbf{X}^n$ to denote the $n$-mode flattening of a third-order tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$; $\mathbf{X}^1$ is of size $I \times JK$, $\mathbf{X}^2$ is of size $J \times KI$, and $\mathbf{X}^3$ is of size $K \times IJ$. The following relationships hold between the entries of $\mathcal{X}$ and its unfolded versions:

$$x_{i,j,k} = x^1_{i,j+(k-1)J} = x^2_{j,k+(i-1)K} = x^3_{k,i+(j-1)I}. \tag{2.3}$$

We can view $\mathbf{X}^1$ as consisting of $K$ stacked frontal slices of $\mathcal{X}$, each of size $I \times J$. Similarly, $\mathbf{X}^2$ consists of $I$ slices of size $J \times K$ and $\mathbf{X}^3$ is made up of $J$ slices of size $K \times I$. If we use $\mathbf{X}^{n,m}$ to denote the $m$-th slice in the $n$-mode flattening of $\mathcal{X}$, then observe that the following holds:

$$x^1_{i,j+(k-1)J} = x^{1,k}_{i,j}, \quad x^2_{j,k+(i-1)K} = x^{2,i}_{j,k}, \quad x^3_{k,i+(j-1)I} = x^{3,j}_{k,i}. \tag{2.4}$$

One can state a relationship between the rows and columns of various flattenings of a tensor, which will be used to derive our distributed tensor factorization algorithm in Section 2.3.

**Lemma 2.2.1** *Let* $(n, n') \in \{(2,1),(3,2),(1,3)\}$*, and let* $\mathbf{X}^n$ *and* $\mathbf{X}^{n'}$ *be the* $n$ *and* $n'$*-mode flattening respectively of a tensor* $\mathcal{X}$*. Moreover, let* $\mathbf{X}^{n,m}$ *be the* $m$*-th slice in* $\mathbf{X}^n$*, and* $\mathbf{x}^{n'}_{m,:}$ *be the* $m$*-th row of* $\mathbf{X}^{n'}$*. Then,* $vec(\mathbf{X}^{n,m}) = \mathbf{x}^{n'}_{m,:}$.

**Proof** Using (2.3) and (2.4), we can write

$$x^3_{k,i+(j-1)I} = x^{1,k}_{i,j} = x^1_{i,j+(k-1)J}.$$

The result for $(n, n') = (1, 3)$ follows directly from (A.7) by letting $k = m$. For other values of $n$ and $n'$, the arguments are analogous. ■

## 2.3 DFacTo

Recall that the main challenge of implementing ALS or GD for solving tensor factorization lies in multiplying a matricized tensor and a Khatri-Rao product of two matrices: $\mathbf{X}^1(\mathbf{C} \odot \mathbf{B})$[1] . If $\mathbf{B}$ is of size $J \times R$ and $\mathbf{C}$ is of size $K \times R$, explicitly forming $(\mathbf{C} \odot \mathbf{B})$ requires $O(JKR)$ memory and is infeasible when $J$ and $K$ are large. This is called the intermediate data explosion problem in the literature [3]. The lemma below will be used to derive our efficient algorithm, which avoids this problem. Although the proof can be inferred using results in [37], we give an elementary proof for completeness.

**Lemma 2.3.1** *The* $r$*-th column of* $\mathbf{X}^1(\mathbf{C} \odot \mathbf{B})$ *can be computed as*

$$\left[\mathbf{X}^1(\mathbf{C} \odot \mathbf{B})\right]_{:,r} = \left[unvec_{(K,I)}\left(\left(\mathbf{X}^2\right)^\top \mathbf{b}_{:,r}\right)\right]^\top \mathbf{c}_{:,r} \tag{2.5}$$

**Proof** We need to show that

$$\left[\mathbf{X}^1(\mathbf{C} \odot \mathbf{B})\right]_{:,r} = \left[unvec_{(K,I)}\left(\left(\mathbf{X}^2\right)^\top \mathbf{b}_{:,r}\right)\right]^\top \mathbf{c}_{:,r} = \begin{bmatrix} \mathbf{b}_{:,r}^\top \mathbf{X}^{2,1} \mathbf{c}_{:,r} \\ \vdots \\ \mathbf{b}_{:,r}^\top \mathbf{X}^{2,I} \mathbf{c}_{:,r} \end{bmatrix}.$$

[1] We mainly concentrate on the update to $\mathbf{A}$ since the updates to $\mathbf{B}$ and $\mathbf{C}$ are analogous.

Or equivalently it suffices to show that $\left[\mathbf{X}^1 \left(\mathbf{C} \odot \mathbf{B}\right)\right]_{i,r} = \mathbf{b}_{:,r}^\top \mathbf{X}^{2,i} \mathbf{c}_{:,r}$. Using (A.9)

$$vec\left(\mathbf{b}_{:,r}^\top \mathbf{X}^{2,i} \mathbf{c}_{:,r}\right) = \left(\mathbf{c}_{:,r}^\top \otimes \mathbf{b}_{:,r}^\top\right) vec\left(\mathbf{X}^{2,i}\right). \tag{2.6}$$

Observe that $\mathbf{b}_{:,r}^\top \mathbf{X}^{2,i} \mathbf{c}_{:,r}$ is a scalar. Moreover, using Lemma 2.2.1 we can write $vec\left(\mathbf{X}^{2,i}\right) = \mathbf{x}_{i,:}^1$. This allows us to rewrite the above equation as

$$\mathbf{b}_{:,r}^\top \mathbf{X}^{2,i} \mathbf{c}_{:,r} = \left(\mathbf{x}_{i,:}^1\right)^\top \left(\mathbf{c}_{:,r} \otimes \mathbf{b}_{:,r}\right) = \left[\mathbf{X}^1 \left(\mathbf{C} \odot \mathbf{B}\right)\right]_{i,r},$$

which completes the proof. ■

Unfortunately, a naive computation of $\left[\mathbf{X}^1 \left(\mathbf{C} \odot \mathbf{B}\right)\right]_{:,r}$ by using (2.5) does not solve the intermediate data explosion problem. This is because $\left(\mathbf{X}^2\right)^\top \mathbf{b}_{:,r}$ produces a $KI$ dimensional vector, which is then reshaped by the $unvec_{(K,I)}(\cdot)$ operator into a $K \times I$ matrix. However, as the next lemma asserts, only a small number of entries of $\left(\mathbf{X}^2\right)^\top \mathbf{b}_{:,r}$ are non-zero.

For convenience, let a vector produced by $(\mathbf{X}^2)^\top \mathbf{b}_{:,r}$ be $\boldsymbol{b}_{:,r}$ and a matrix produced by $\left[unvec_{(K,I)}(\boldsymbol{b}_{:,r})\right]^\top$ be $\mathbf{M}^r$.

**Lemma 2.3.2** *The number of non-zeros in* $\boldsymbol{b}_{:,r}$ *is at most* $nnzr((\mathbf{X}^2)^\top)$ *and* $nnzc(\mathbf{X}^2)$.

**Proof** Multiplying an all-zero row in $(\mathbf{X}^2)^\top$ and $\mathbf{b}_{:,r}$ produces zero. Therefore, the number of non-zeros in $\boldsymbol{b}_{:,r}$ is equal to the number of rows in $(\mathbf{X}^2)^\top$ that contain at least one non-zero element. Also, by definition, $nnzr((\mathbf{X}^2)^\top)$ is equal to $nnzc(\mathbf{X}^2)$. ■

As a consequence of the above lemma, we only need to explicitly compute the non-zero entries of $\boldsymbol{b}_{:,r}$. However, the problem of reshaping $\boldsymbol{b}_{:,r}$ via the $\left[unvec_{(K,I)}(\cdot)\right]^\top$ operator still remains. The next lemma shows how to overcome this difficulty.

**Lemma 2.3.3** *The location of the non-zero entries of* $\mathbf{M}^r$ *depends on* $(\mathbf{X}^2)^\top$ *and is independent of* $\mathbf{b}_{:,r}$.

**Proof** The product of the $(k+(i-1)K)$-*th* row of $(\mathbf{X}^2)^\top$ and $\mathbf{b}_{:,r}$ is the $(k+(i-1)K)$-*th* element of $\boldsymbol{b}_{:,r}$. And, this element is the $(i,k)$-*th* entry of $\mathbf{M}^r$ by definition of

$\left[unvec_{(K,I)}(\cdot)\right]^\top$. Therefore, if all the entries in the $(k+(i-1)K)$-*th* row of $(\mathbf{X}^2)^\top$ are zero, then the $(i,k)$-*th* entry of $\mathbf{M}^r$ is zero regardless of $\mathbf{b}_{:,r}$. Consequently, the location of the non-zero entries of $\mathbf{M}^r$ is independent of $\mathbf{b}_{:,r}$, and is only determined by $(\mathbf{X}^2)^\top$. ■

Given $\boldsymbol{\mathcal{X}}$ one can compute $(\mathbf{X}^2)^\top$ to know the locations of the non-zero entries of $\mathbf{M}^r$. In other words, we can infer the non-zero pattern and therefore preallocate memory for $\mathbf{M}^r$. We will show below how this allows us to perform the $\left[unvec_{(K,I)}(\cdot)\right]^\top$ operation for free.

Recall the Compressed Sparse Row (CSR) Format, which stores a sparse matrix as three arrays namely *values*, *columns*, and *rows*. Here, *values* represents the non-zero values of the matrix; while *columns* stores the column indices of the non-zero values. Also, *rows* stores the indices of the *columns* array where each row starts. For example, if a sparse matrix $\mathbf{M}^r$ is

$$\mathbf{M}^r = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 3 & 4 \end{bmatrix},$$

then the CSR of $\mathbf{M}^r$ is

$$\begin{aligned} value(\mathbf{M}^r) &= \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \\ col(\mathbf{M}^r) &= \begin{bmatrix} 0 & 2 & 1 & 2 \end{bmatrix} \\ row(\mathbf{M}^r) &= \begin{bmatrix} 0 & 2 & 4 \end{bmatrix}. \end{aligned}$$

Different matrices with the same sparsity pattern can be represented by simply changing the entries of the *value* array. For our particular case, what this means is that we can pre-compute $col(\mathbf{M}^r)$ and $row(\mathbf{M}^r)$ and pre-allocate $value(\mathbf{M}^r)$. By writing the non-zero entries of $\boldsymbol{b}_{:,r}$ into $value(\mathbf{M}^r)$ we can "reshape" $\boldsymbol{b}_{:,r}$ into $\mathbf{M}^r$.

Let the matrix with all-zero rows in $(\mathbf{X}^2)^\top$ removed be $(\hat{\mathbf{X}}^2)^\top$. Then, Algorithm 1 shows the DFacTo algorithm for computing $\mathbf{N} := \mathbf{X}^1 (\mathbf{C} \odot \mathbf{B})$. Here, the input values

are $(\hat{\mathbf{X}}^2)^\top$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{M}^r$ preallocated in CSR format. By storing the results of the product of $(\hat{\mathbf{X}}^2)^\top$ and $\mathbf{b}_{:,r}$ directly into $value(\mathbf{M}^r)$, we can obtain $\mathbf{M}^r$ because $\mathbf{M}^r$ was preallocated in the CSR format. Then, the product of $\mathbf{M}^r$ and $\mathbf{c}_{:,r}$ yields the *r-th* column of $\mathbf{N}$. We obtain the output $\mathbf{N}$ by repeating these two sparse matrix-vector products $R$ times.

**Algorithm 1:** DFacTo algorithm for Tensor Factorization

**Input:** $(\hat{\mathbf{X}}^2)^\top$, $\mathbf{B}$, $\mathbf{C}$, $value(\mathbf{M}^r)$ $col(\mathbf{M}^r)$, $row(\mathbf{M}^r)$
**Output:** $\mathbf{N}$
**while** *r=1, 2,…, R* **do**
  $value(\mathbf{M}^r) \leftarrow (\hat{\mathbf{X}}^2)^\top \mathbf{b}_{:,r}$
  $\mathbf{n}_{:,r} \leftarrow \mathbf{M}^r \mathbf{c}_{:,r}$
**end**

**Algorithm 2:** DFacTo(ALS) algorithm for Tensor Factorization

**Input:** $\mathbf{X}^1$, $\mathbf{X}^2$, $\mathbf{X}^3$
**Initialize:** $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$
**while** *stopping criterion not met* **do**
  **while** *r=1, 2,…, R* **do**
    $\mathbf{n}_{:,r} \leftarrow unvec_{(K,I)}\left(\left(\mathbf{X}^2\right)^\top \mathbf{b}_{:,r}\right)^\top \mathbf{c}_{:,r}$
  **end**
  $\mathbf{A} \leftarrow \mathbf{N}\left(\mathbf{C}^\top\mathbf{C} * \mathbf{B}^\top\mathbf{B}\right)^{-1}$
  Normalize columns of $\mathbf{A}$
  **while** *r=1, 2,…, R* **do**
    $\mathbf{n}_{:,r} \leftarrow unvec_{(I,J)}\left(\left(\mathbf{X}^3\right)^\top \mathbf{c}_{:,r}\right)^\top \mathbf{a}_{:,r}$
  **end**
  $\mathbf{B} \leftarrow \mathbf{N}\left(\mathbf{A}^\top\mathbf{A} * \mathbf{C}^\top\mathbf{C}\right)^{-1}$
  Normalize columns of $\mathbf{B}$
  **while** *r=1, 2,…, Right* **do**
    $\mathbf{n}_{:,r} \leftarrow unvec_{(J,K)}\left(\left(\mathbf{X}^1\right)^\top \mathbf{a}_{:,r}\right)^\top \mathbf{b}_{:,r}$
  **end**
  $\mathbf{C} \leftarrow \mathbf{N}\left(\mathbf{B}^\top\mathbf{B} * \mathbf{A}^\top\mathbf{A}\right)^{-1}$
  Normalize columns of $\mathbf{C}$
**end**

It is immediately obvious that using the above lemmas to compute $\mathbf{N}$ requires no extra memory other than storing $\mathbf{M}^r$, which contains at most $nnzc(\mathbf{X}^2) \leq \Omega^{\mathcal{X}}$ non-zero entries. Therefore, we completely avoid the intermediate data explosion problem. Moreover, the same subroutine can be used for both ALS and GD. The ALS and GD algorithms of the DFacTo model is summarized in Algorithms 2 and 3. We can solve the problem of overfitting by adding a $\lambda\mathbf{I}$ term in $\mathbf{C}^\top\mathbf{C} * \mathbf{B}^\top\mathbf{B}$, $\mathbf{A}^\top\mathbf{A} * \mathbf{C}^\top\mathbf{C}$, and $\mathbf{B}^\top\mathbf{B} * \mathbf{A}^\top\mathbf{A}$ of Algorithms 2 (lines 7, 12, 17) and 3 (lines 7, 11, 15).

### 2.3.1 Distributed Memory Implementation

Our algorithm is easy to parallelize using a master-slave architecture of MPI(Message Passing Interface). At every iteration, the master transmits $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ to the slaves.

**Algorithm 3:** DFacTo(GD) algorithm for Tensor Factorization

1 **Input:** $\mathbf{X}^1$, $\mathbf{X}^2$, $\mathbf{X}^3$
2 **Initialize:** $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$
3 **while** *stopping criterion not met* **do**
4  **while** *r=1, 2,..., R* **do**
5  $\mathbf{n}_{:,r} \leftarrow unvec_{(K,I)}((\mathbf{X}^2)^\top \mathbf{b}_{:,r})^\top \mathbf{c}_{:,r}$
6  **end**
7  $\nabla\mathbf{A} \leftarrow \mathbf{N} + \mathbf{A}\left(\mathbf{C}^\top\mathbf{C} * \mathbf{B}^\top\mathbf{B}\right)$
8  **while** *r=1, 2,..., R* **do**
9  $\mathbf{n}_{:,r} \leftarrow unvec_{(I,J)}\left((\mathbf{X}^3)^\top \mathbf{c}_{:,r}\right)^\top \mathbf{a}_{:,r}$
10  **end**
11  $\nabla\mathbf{B} \leftarrow \mathbf{N} + \mathbf{B}\left(\mathbf{A}^\top\mathbf{A} * \mathbf{C}^\top\mathbf{C}\right)$
12  **while** *r=1, 2,..., Right* **do**
13  $\mathbf{n}_{:,r} \leftarrow unvec_{(J,K)}\left((\mathbf{X}^1)^\top \mathbf{a}_{:,r}\right)^\top \mathbf{b}_{:,r}$
14  **end**
15  $\nabla\mathbf{C} \leftarrow \mathbf{N} + \mathbf{C}\left(\mathbf{B}^\top\mathbf{B} * \mathbf{A}^\top\mathbf{A}\right)$
16  $\alpha \leftarrow Linesearch(\mathbf{A}, \mathbf{B}, \mathbf{C}, \nabla\mathbf{A}, \nabla\mathbf{B}, \nabla\mathbf{C})$
17  $\mathbf{A} \leftarrow \mathbf{A} - \alpha\nabla\mathbf{A}$
18  $\mathbf{B} \leftarrow \mathbf{B} - \alpha\nabla\mathbf{B}$
19  $\mathbf{C} \leftarrow \mathbf{C} - \alpha\nabla\mathbf{C}$
20 **end**

The slaves hold a fraction of the rows of $\mathbf{X}^2$ using which a fraction of the rows of $\mathbf{N}$ is computed. By performing a synchronization step, the slaves can exchange rows of $\mathbf{N}$. In ALS, this $\mathbf{N}$ is used to compute $\mathbf{A}$ which is transmitted back to the master. Then, the master updates $\mathbf{A}$, and the iteration proceeds. In GD, the slaves transmit $\mathbf{N}$ back to the master, which computes $\nabla\mathbf{A}$. Then, the master computes the step size by a line search algorithm, updates $\mathbf{A}$, and the iteration proceeds.

### 2.3.2 Complexity Analysis

A naive computation of $\mathbf{N}$ requires $\left(JK + \ \Omega^{\mathcal{X}}\ \right) R$ flops; forming $\mathbf{C} \odot \mathbf{B}$ requires $JKR$ flops and performing the matrix-matrix multiplication $\mathbf{X}^1 (\mathbf{C} \odot \mathbf{B})$ requires $\ \Omega^{\mathcal{X}}\ R$ flops. Our algorithm requires only $\left(nnzc(\mathbf{X}^2) + \ \Omega^{\mathcal{X}}\ \right) R$ flops; $\ \Omega^{\mathcal{X}}\ R$ flops for computing $\boldsymbol{b}_{:,r}$ and $nnzc(\mathbf{X}^2)R$ flops for computing $\mathbf{M}^r\mathbf{c}_{:,r}$. Note that, typically, $nnzc(\mathbf{X}^2) \ll$ both $JK$ and $\ \Omega^{\mathcal{X}}\ $ (see Table 2.1). In terms of memory, the naive algorithm requires $O(JKR)$ extra memory, while our algorithm only requires $nnzc(\mathbf{X}^2)$ extra space to store $\mathbf{M}^r$.

### 2.3.3 Related Work

Two papers that are most closely related to our work are the GigaTensor algorithm proposed by [3] and the Sparse Tensor Toolbox of [2]. As discussed above, both algorithms attack the problem of computing $\mathbf{N}$ efficiently. In order to compute $\mathbf{n}_{:,r}$, GigaTensor computes two intermediate matrices $\mathbf{N}_1 := \mathbf{X}^1 * \left(\mathbf{1}_I \odot (\mathbf{c}_{:,r} \otimes \mathbf{1}_J)^\top\right)$ and $\mathbf{N}_2 := bin\left(\mathbf{X}^1\right) * \left(\mathbf{1}_I \odot (\mathbf{1}_K \otimes \mathbf{b}_{:,r})^\top\right)$. Next, $\mathbf{N}_3 := \mathbf{N}_1 * \mathbf{N}_2$ is computed, and $\mathbf{n}_{:,r}$ is obtained by computing $\mathbf{N}_3\ \mathbf{1}_{JK}$. As reported in [3], GigaTensor uses 2 $\Omega^{\mathcal{X}}$ extra storage and 5 $\Omega^{\mathcal{X}}$ flops to compute one column of $\mathbf{N}$. The Sparse Tensor Toolbox stores a tensor as a vector of non-zero values and a matrix of corresponding indices. Entries of $\mathbf{B}$ and $\mathbf{C}$ are replicated appropriately to create intermediate vectors. A Hadamard product is computed between the non-zero entries of the matrix and intermediate vectors, and a selected set of entries are summed to form columns of $\mathbf{N}$. The

algorithm uses 2 $\Omega^{\mathcal{X}}$ extra storage and 5 $\Omega^{\mathcal{X}}$ flops to compute one column of $\mathbf{N}$. See Appendix A.4 for a detailed illustrative example which shows all the intermediate calculations performed by our algorithm as well as the algorithm of [3] and [2].

Also, [66] suggests the gradient-based optimization of CANDECOMP/PARAFAC (CP) using the same method as [2] to compute $\mathbf{X}^1 (\mathbf{C} \odot \mathbf{B})$. [66] refers to this gradient-based optimization algorithm as CPOPT and the ALS algorithm of CP using the method of [2] as CPALS. Following [66], we use these names, CPALS and CPOPT.

## 2.4 Experimental Evaluation

Our experiments are designed to study the scaling behavior of DFacTo on both publicly available real-world datasets as well as synthetically generated data. We contrast the performance of DFacTo (ALS) with GigaTensor [3] as well as with CPALS [2], while the performance of DFacTo (GD) is compared with CPOPT [66]. We also present results to show the scaling behavior of DFacTo when data is distributed across multiple machines.

**Datasets** See Table 2.1 for a summary of the real-world datasets we used in our experiments. The NELL-1 and NELL-2 datasets are from [3] and consists of (noun phrase 1, context, noun phrase 2) triples from the "Read the Web" project [41]. NELL-2 is a version of NELL-1, which is obtained by removing entries whose values are below a threshold.

The Yelp Phoenix dataset is from the Yelp Data Challenge [2], while Cellartracker, Ratebeer, Beeradvocate and Amazon.com are from the Stanford Network Analysis Project (SNAP) home page. All these datasets consist of product or business reviews. We converted them into a users $\times$ items $\times$ words tensor by first splitting the text into words, removing stop words, using Porter stemming [69], and then removing user-item pairs which did not have any words associated with them. In addition, for the Amazon.com dataset we filtered words that appeard less than 5 times or in fewer

[2]https://www.yelp.com/dataset_challenge/dataset

than 5 documents. Note that the number of dimensions as well as the number of non-zero entries reported in Table 2.1 differ from those reported in [39] because of our pre-processing.

Table 2.1.: Summary statistics of the datasets used in our experiments.

| Dataset | $I$ | $J$ | $K$ | $\Omega^{\hat{\mathcal{X}}}$ | $nnzc(\mathrm{X}^1)$ | $nnzc(\mathrm{X}^2)$ | $nnzc(\mathrm{X}^3)$ |
|---|---|---|---|---|---|---|---|
| Yelp | 46.0K | 11.5K | 84.5K | 9.9M | 4.3M | 6.1M | 229.8K |
| Cellartracker | 36.5K | 412.4K | 163.5K | 25.0M | 19.2M | 5.9M | 1.3M |
| NELL-2 | 12.1K | 9.2K | 28.8K | 76.9M | 16.6M | 21.5M | 337.4K |
| Beeradvocate | 33.4K | 66.1K | 204.1K | 78.8M | 19.0M | 12.1M | 1.6M |
| Ratebeer | 29.1K | 110.3K | 294.0K | 77.1M | 22.4M | 7.8M | 2.9M |
| NELL-1 | 2.9M | 2.1M | 25.5M | 143.7M | 113.3M | 119.1M | 17.4M |
| Amazon | 6.6M | 2.4M | 1.7M | 1.2B | 525.3M | 389.6M | 29.9M |

We also generated the following two kinds of synthetic data for our experiments:

- the number of non-zero entries in the tensor is held fixed but we vary $I$, $J$, and $K$.
- the dimensions $I$, $J$, and $K$ are held fixed but the number of non-zeros entries varies.

To simulate power law behavior, both the above datasets were generated using the following preferential attachment model [70]: the probability that a non-zero entry is added at index $(i, j, k)$ is given by $p_i \times p_j \times p_k$, where $p_i$ (resp. $p_j$ and $p_k$) is proportional to the number of non-zero entries at index $i$ (resp. $j$ and $k$).

**Implementation and Hardware** All experiments were conducted on a computing cluster where each node has two 2.1 GHz 12-core AMD 6172 processors with 48 GB physical memory per node. Our algorithms are implemented in C++ using the Eigen library[3] and compiled with the Intel Compiler. We downloaded Version 2.5 of the Tensor Toolbox, which is implemented in MATLAB[4]. Since open source code for

[3] `http://eigen.tuxfamily.org`
[4] `http://www.sandia.gov/~tgkolda/TensorToolbox/`

GigaTensor is not freely available, we developed our own version in C++ following the description in [3]. Also, we used MPICH2[5] in order to distribute the tensor factorization computation to multiple machines. All our codes are available for download under an open source license from `http://www.joonheechoi.com/research`.

**Scaling on Real-World Datasets** Both CPALS and our implementation of GigaTensor are uni-processor codes. Therefore, for this experiment we restricted ourselves to datasets which can fit on a single machine. When initialized with the same starting point, DFacTo and its competing algorithms will converge to the same solution. Therefore, we only compare the CPU time per iteration of the different algorithms. The results are summarized in Table 2.2. On many datasets DFacTo (ALS) is around 5 times faster than GigaTensor and 10 times faster than CPALS; the differences are more pronounced on the larger datasets. Also, DFacTo (GD) is around 4 times faster than CPOPT.

Table 2.2.: Times per iteration (in seconds) of DFacTo (ALS), GigaTensor, CPALS, DFacTo (GD), and CPOPT on datasets which can fit in a single machine ($R$=10).

| Dataset | DFacTo (ALS) | GigaTensor | CPALS | DFacTo (GD) | CPOPT |
|---|---|---|---|---|---|
| Yelp Phoenix | 9.52 | 26.82 | 46.52 | 13.57 | 45.9 |
| Cellartracker | 23.89 | 80.65 | 118.25 | 35.82 | 130.32 |
| NELL-2 | 32.59 | 186.30 | 376.10 | 80.79 | 386.25 |
| Beeradvocate | 43.84 | 224.29 | 364.98 | 94.85 | 481.06 |
| Ratebeer | 44.20 | 240.80 | 396.63 | 87.36 | 349.18 |
| NELL-1 | 322.45 | 772.24 | - | 742.67 | - |

The differences in performance between DFacTo (ALS) and CPALS and between DFacTo (GD) and CPOPT can partially be explained by the fact that DFacTo (ALS, GD) is implemented in C++ while CPALS and CPOPT use MATLAB. However, it must be borne in mind that both MATLAB and our implementation use an optimized BLAS library to perform their computationally intensive numerical linear algebra operations.

[5] `http://www.mpich.org/static/downloads/`

Compared to the Map-Reduce version implemented in Java and used for the experiments reported in [3], our C++ implementation of GigaTensor is significantly faster and more optimized. As per [3], the Java implementation took approximately 10,000 seconds per iteration to handle a tensor with around $10^9$ non-zero entries, when using 35 machines. In contrast, the C++ version was able to handle one iteration of the ALS algorithm on the NELL-1 dataset on a single machine in 772 seconds. However, because DFacto (ALS) uses a better algorithm, it is able to handsomely outperform GigaTensor and only takes 322 seconds per iteration.

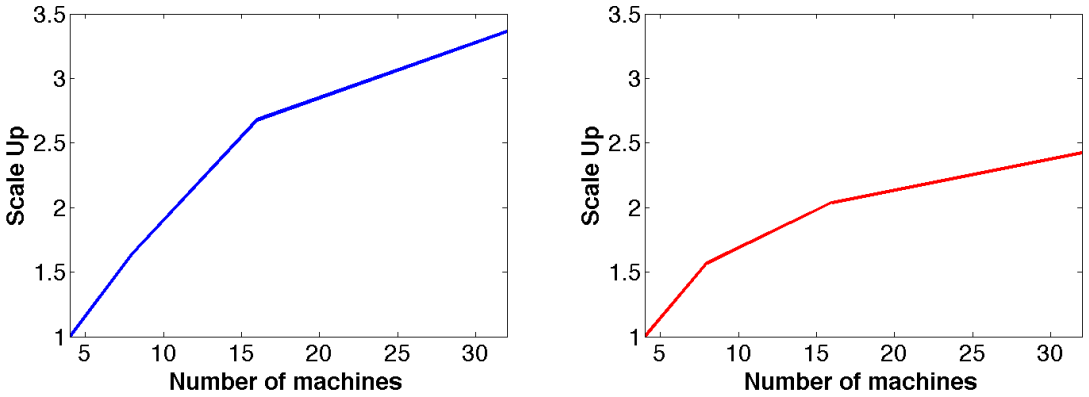
Also, the execution time of DFacTo (GD) is longer than that of DFacTo (ALS) because DFacTo (GD) spends more time on the line search algorithm to obtain an appropriate step size.

**Scaling across Machines** Our goal is to study scaling behavior of the time per iteration as datasets are distributed across different machines. Towards this end we worked with two datasets. NELL-1 is a moderate-size dataset which our algorithm can handle on a single machine, while Amazon is a large dataset which does not fit on a single machine. Table 2.3 shows that the iteration time decreases as the number of machines increases on the NELL-1 and Amazon datasets. While the decrease in iteration time is not completely linear, the computation time excluding both synchronization and line search time decreases linearly. The Y-axis in Figure 2.2 indicates $T_4/T_n$ where $T_n$ is the single iteration time with $n$ machines on the Amazon dataset.

**Synthetic Data Experiments** We perform two experiments with synthetically generated tensor data. In the first experiment we fix the number of non-zero entries to be $10^6$ and let $I = J = K$ and vary the dimensions of the tensor. For the second experiment we fix the dimensions and let $I = J = K$ and the number of non-zero entries is set to be $2I$. The scaling behavior of the three algorithms on these two datasets is summarized in Table 2.4. Since we used a preferential attachment model to generate the datasets, the non-zero indices exhibit a power law behavior. Consequently, the number of columns with non-zero elements ($nnzc(\cdot)$) for $\mathbf{X}^1$, $\mathbf{X}^2$ and $\mathbf{X}^3$ is very

Table 2.3.: Total Time and CPU time per iteration (in seconds) as a function of number of machines for the NELL-1 and Amazon datasets ($R$=10).

| | DFacTo (ALS) | | | | DFacTo (GD) | | | |
|---|---|---|---|---|---|---|---|---|
| | NELL-1 | | Amazon | | NELL-1 | | Amazon | |
| Machines | Iter. | CPU | Iter. | CPU | Iter. | CPU | Iter. | CPU |
| 1 | 322.45 | 322.45 | - | - | 742.67 | 104.23 | - | - |
| 2 | 205.07 | 167.29 | - | - | 492.38 | 55.11 | - | - |
| 4 | 141.02 | 101.58 | 480.21 | 376.71 | 322.65 | 28.55 | 1143.7 | 127.57 |
| 8 | 86.09 | 62.19 | 292.34 | 204.41 | 232.41 | 16.24 | 727.79 | 62.61 |
| 16 | 81.24 | 46.25 | 179.23 | 98.07 | 178.92 | 9.70 | 560.47 | 28.61 |
| 32 | 90.31 | 34.54 | 142.69 | 54.60 | 209.39 | 7.45 | 471.91 | 15.78 |




(a) DFacTo(ALS) (b) DFacTo(GD)

Fig. 2.2.: The scalability of DFacTo with respect to the number of machines on the Amazon dataset

close to the total number of non-zero entries in the tensor. Therefore, as predicted by theory, DFacTo (ALS, GD) does not enjoy significant speedups when compared to GigaTensor, CPALS and CPOPT. However, it must be noted that DFacto (ALS) is faster than either GigaTensor or CPALS in all but one case and DFacTo (GD) is faster than CPOPT in all cases. We attribute this to better memory locality which arises as a consequence of reusing the memory for $\mathbf{N}$ as discussed in Section 2.3.

**Rank Variation Experiments** Table 2.5 shows the time per iteration on various ranks ($R$) with the NELL-2 dataset. We see that the computation time of our al-

Table 2.4.: Time per iteration (in seconds) on synthetic datasets (non-zeros = $10^6$ or $2I$, $R$=10)

| $I = J = K$ | Non-zeros | DFacTo (ALS) | GigaTensor | CPALS | DFacTo (GD) | CPOPT |
|---|---|---|---|---|---|---|
| $10^4$ | $10^6$ | 1.14 | 2.80 | 5.10 | 2.32 | 5.21 |
| $10^5$ | $10^6$ | 2.72 | 6.71 | 6.11 | 5.87 | 11.70 |
| $10^6$ | $10^6$ | 7.26 | 11.86 | 16.54 | 16.51 | 29.13 |
| $10^7$ | $10^6$ | 41.64 | 38.19 | 175.57 | 121.30 | 202.71 |
| $10^4$ | $2 \cdot 10^4$ | 0.05 | 0.09 | 0.52 | 0.09 | 0.57 |
| $10^5$ | $2 \cdot 10^5$ | 0.92 | 1.61 | 1.50 | 1.81 | 2.98 |
| $10^6$ | $2 \cdot 10^6$ | 12.06 | 22.08 | 15.84 | 21.74 | 26.04 |
| $10^7$ | $2 \cdot 10^7$ | 144.48 | 251.89 | 214.37 | 275.19 | 324.2 |

gorithm increases lineraly in $R$ like the time complexity analyzed in Section 2.3.2.

Table 2.5.: Time per iteration (in seconds) on various $R$

| $R$ | 5 | 10 | 20 | 50 | 100 | 200 | 500 |
|---|---|---|---|---|---|---|---|
| NELL-2 | 15.84 | 31.92 | 58.71 | 141.43 | 298.89 | 574.63 | 1498.68 |

## 2.5 Discussion and Conclusion

We presented a technique for significantly speeding up the Alternating Least Squares (ALS) and the Gradient Descent (GD) algorithm for tensor factorization by exploiting properties of the Khatri-Rao product. Not only is our algorithm, DFacto, computationally attractive, but it is also more memory efficient compared to existing algorithms. Furthermore, we presented a strategy for distributing the computations across multiple machines.

We hope that the availability of a scalable tensor factorization algorithm will enable practitioners to work on more challenging tensor datasets, and therefore lead to advances in the analysis and understanding of tensor data. Towards this end we

intend to make our code freely available for download under a permissive open source license.

Although we mainly focused on tensor factorization using ALS and GD, it is worth noting that one can extend the basic ideas behind DFacTo to other related problems such as joint matrix completion and tensor factorization. We presented such a model in Section A.5. In fact, we believe that this joint matrix completion and tensor factorization model by itself is somewhat new and interesting in its own right, despite its resemblance to other joint models including tensor factorization such as [71]. In our joint model, we are given a user $\times$ item ratings matrix $\mathbf{Y}$, and some side information such as a user $\times$ item $\times$ words tensor $\mathcal{X}$. Preliminary experimental results suggest that jointly factorizing $Y$ and $\mathcal{X}$ outperforms vanilla matrix completion.

# 3. COMPARISON OF ADMM ALGORITHMS FOR NOISY NON-NEGATIVE MATRIX FACTORIZATION

## 3.1 Introduction

### 3.1.1 Non-negative Matrix Factorization

We consider the classical non-negative matrix factorization problem with a special concern about the *noise* present in the observation. More precisely, we assume that for a given pair of matrices $\boldsymbol{W} \in \mathbb{R}^{M\times R}$ and $\boldsymbol{H} \in \mathbb{R}^{R\times N}$, the observation model is given by

$$\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{H} + \boldsymbol{\eta}, \tag{3.1}$$

where $\boldsymbol{Y} \in \mathbb{R}^{M\times N}$ is the observed data, and $\boldsymbol{\eta}$ is a zero-mean iid Gaussian noise matrix with variance $\sigma^2$. Here, the dimension $R$ specifies the number of columns of $\boldsymbol{W}$ (also the number of rows of $\boldsymbol{H}$) on which the rank of the observation model is defined.

Non-negative matrix factorization has important applications in computer vision [72], recommendation system [73], remote sensing [74], and hyperspectral imaging [75]. In this chapter, we are primarily interested in using non-negative factorization to identify chemical signatures from stimulated Raman scattering (SRS) imaging data [76]. In chemometrics, such process is widely known as the multivariate curve resolution (MCR) technique [77].

Given the observation model in (3.1), non-negative matrix factorization is typically posed as a minimization problem

$$\underset{\boldsymbol{W}\geq 0,\boldsymbol{H}\geq 0}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{W}\boldsymbol{H}\|_F^2, \tag{3.2}$$

where the goal is to minimize the $\ell^2$-norm of the residue between the observed data $\boldsymbol{Y}$ and the predicted data $\boldsymbol{WH}$. In the absence of noise, (3.2) can be solved by a multiplicative algorithm [4], the alternating least-squares method [78], or more recently the alternating direction method of multiplier (ADMM) [11]. However, it should be reminded that since (3.2) is a non-convex problem, there is in general no guarantee of global optimality except under some restrictive conditions [79].

### 3.1.2 Problem Statement

The focus of this chapter is to perform non-negative matrix factorization in the presence of $\boldsymbol{\eta}$. When $\boldsymbol{\eta}$ is present, solving (3.2) is inadequate to return a pair of clean ($\boldsymbol{W}$, $\boldsymbol{H}$). In this case, a common wisdom is to consider some regularization functions on $\boldsymbol{W}$ and $\boldsymbol{H}$ and turn the problem into

$$\underset{\boldsymbol{W}\geq 0,\boldsymbol{H}\geq 0}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{WH}\|_F^2+\lambda f(\boldsymbol{W},\boldsymbol{H}), \tag{3.3}$$

where $f(\boldsymbol{W},\boldsymbol{H})$ is the regularization on $\boldsymbol{W}$ and $\boldsymbol{H}$, and $\lambda$ is the associated parameter. For example, one may choose $f(\boldsymbol{W},\boldsymbol{H})=\|\boldsymbol{W}\|_F^2+\|\boldsymbol{H}\|_F^2$ to constrain the energy of $\boldsymbol{W}$ and $\boldsymbol{H}$ [75], or use the spectral total variation regularization [80]- [81]

$$\underset{\boldsymbol{W}\geq 0,\boldsymbol{H}\geq 0}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{WH}\|_F^2+\lambda\|\boldsymbol{WH}\|_{STV}, \tag{3.4}$$

which we shall discuss in Section 3.2 of this chapter. The price we have to pay, of course, is that (3.3) is often more difficult than (3.2).

Our primary concern of this chapter is how to solve (3.4) efficiently. Readers who are familiar with total variation may probably recognize that (3.4) is a variant of the total variation minimization, which suggests that we can use the ADMM algorithm. However, as we will discuss in this chapter, the way we setup the augmented Lagrangian function for the ADMM algorithm has some drastic influence on the performance, even if they are derived from the same procedure.

### 3.1.3 Contributions

The majority of the previous work on non-negative matrix factorization has been focusing on proposing new algorithms to solve (3.2) [4]- [11], with some others on finding better $f(\boldsymbol{W}, \boldsymbol{H})$ in (3.3) [82]- [83]. While we also have these two goals in mind, we put extra emphasis on the analysis of *when the ADMM algorithm works* and when it does not work. The importance of such analysis is that it informs us the proper design of the ADMM algorithm for some non-convex problems. There are two contributions of this chapter.

First, we discuss four types of algorithms that can be derived for solving (3.4). We show that each one is a variant of ADMM, and we comment on where they were used in the literature. Second, we provide numerical evidence that some performs well but some performs badly. We provide our preliminary understanding of the situation.

The rest of the chapter is organized in the following way. In Section 3.2 we discuss the spectral total variation (3.4) in some details. Then, in Section 3.3 we discuss various possible ways of solving (3.4) using the ADMM algorithm. Since none of these algorithms can guarantee global optimum, in Section 3.4 we study their empirical behavior on some synthetic datasets. A discussion and conclusion is given in Section 3.5.

## 3.2 Background

In this section we provide a brief discussion of the spectral total variation defined in (3.4), and a quick review of the ADMM algorithm. For notation simplicity, we define the operations ten$(\cdot)$, mat$(\cdot)$ and vec$(\cdot)$ for a tensor $\mathcal{X}$, its matrix representation $\boldsymbol{X} \in \mathbb{R}^{MN \times K}$, and vector representation $\boldsymbol{x} \in \mathbb{R}^{MNK \times 1}$ as

$$\mathcal{X} = \text{ten}(\boldsymbol{X}), \quad \boldsymbol{X} = \text{mat}(\boldsymbol{x}) \quad \text{and} \quad \boldsymbol{x} = \text{vec}(\mathcal{X}).$$

Moreover, whenever an inner product is needed, we write

$$\boldsymbol{X}^T\boldsymbol{Y} \stackrel{\text{def}}{=} \text{vec}(\mathcal{X})^T\text{vec}(\mathcal{Y}).$$

### 3.2.1 Spectral Total Variation

We define the (anisotropic) spectral total variation of $\boldsymbol{X}$ as

$$\|\boldsymbol{X}\|_{STV} = \beta_x\|\boldsymbol{D}_x\boldsymbol{x}\|_1 + \beta_y\|\boldsymbol{D}_y\boldsymbol{x}\|_1 + \beta_z\|\boldsymbol{D}_z\boldsymbol{x}\|_1, \tag{3.5}$$

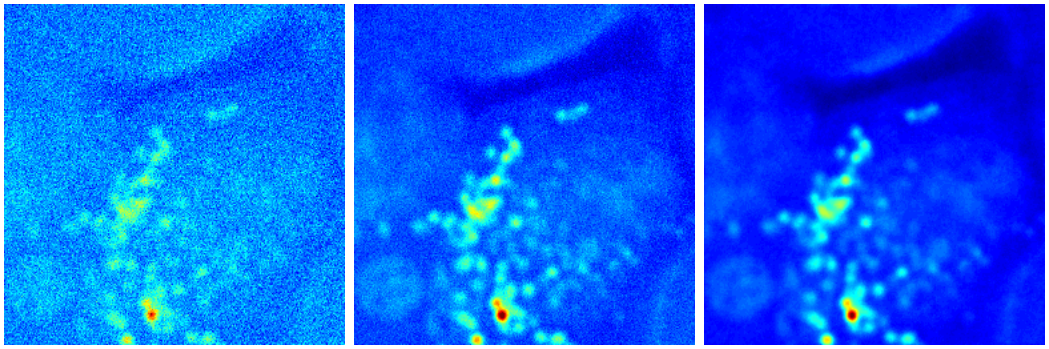
where $(\boldsymbol{D}_x, \boldsymbol{D}_y, \boldsymbol{D}_z)$ are the forward finite-difference operators along the horizontal, vertical and temporal directions, with parameters $(\beta_x, \beta_y, \beta_z)$, respectively. $\|\boldsymbol{X}\|_{STV}$ generalizes conventional total variation to multiple dimensions [81].

To illustrate the effectiveness of the STV on real data, we consider a stimulated Raman scattering image of size $512 \times 512 \times 80$ for a cell. The image shown on the left hand side is the center $128 \times 128$ portion of the 45th frame of the product $\boldsymbol{W}\boldsymbol{H}$ where $\boldsymbol{W}$ and $\boldsymbol{H}$ are results of (3.2) with $R = 8$. It can be seen that the predicted observation $\boldsymbol{W}\boldsymbol{H}$ remains noisy, although the rank we posed on the problem is small. In the same figure we also show the result of solving (3.4) using Algorithm 4. It is evident from the result that the STV reduces a significant amount of noise in the data.

### 3.2.2 ADMM Algorithm

ADMM algorithm solves constrained optimization problems of the following form:

$$\underset{\boldsymbol{x},\boldsymbol{y}}{\text{minimize}} \quad f(\boldsymbol{x}) + g(\boldsymbol{y}), \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} = \boldsymbol{c}. \tag{3.6}$$



(a) Input (b) Solution of (3.2) (c) Solution of (3.4)

Fig. 3.1.: Non-negative matrix factorization with and without regularization. Here we show the product of the solution $\boldsymbol{W}\boldsymbol{H}$.

To solve (3.6), we consider the augmented Lagrangian function

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = f(\boldsymbol{x}) + g(\boldsymbol{y}) + \boldsymbol{z}^T(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} - \boldsymbol{c}) + (\rho/2)\|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} - \boldsymbol{c}\|^2, \tag{3.7}$$

and solve for $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$ alternatively via the procedure

$$\boldsymbol{x}^{(k+1)} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \quad \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}^{(k)}, \boldsymbol{z}^{(k)}), \tag{3.8a}$$

$$\boldsymbol{y}^{(k+1)} = \underset{\boldsymbol{y}}{\operatorname{argmin}} \quad \mathcal{L}(\boldsymbol{x}^{(k+1)}, \boldsymbol{y}, \boldsymbol{z}^{(k)}), \tag{3.8b}$$

$$\boldsymbol{z}^{(k+1)} = \boldsymbol{z}^{(k)} + \rho\left(\boldsymbol{A}\boldsymbol{x}^{(k+1)} + \boldsymbol{B}\boldsymbol{y}^{(k+1)} - \boldsymbol{c}\right). \tag{3.8c}$$

Under mild conditions, e.g., when $f(\cdot)$ is strongly convex and $g(\cdot)$ is convex, the convergence of the algorithm is guaranteed [84].

One common engineering question about using ADMM is how to convert a given optimization problem (e.g., (3.4)) into the form of (3.6). Of course, the basic requirement is that the re-formulated problem (i.e., (3.6)) should be equivalent to the original problem at the optimal solution. However, there are usually multiple ways of re-formulating the problem by choosing different combinations of $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{c})$ in (3.6). In what follows we will discuss several possible options of the ADMM.

## 3.3 Variations of ADMM Algorithms

In this section we present four versions of ADMM to solve (3.4).

### 3.3.1 Single Variable Split

The first method we consider is to define an intermediate variable $\boldsymbol{Z}$ and solve for

$$\begin{aligned} \underset{\boldsymbol{W}\geq 0,\boldsymbol{H}\geq 0,\boldsymbol{Z}}{\text{minimize}} \quad & \frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{W}\boldsymbol{H}\|_F^2+\lambda\|\boldsymbol{Z}\|_{STV} \\ \text{subject to} \quad & \boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z}=\boldsymbol{0}. \end{aligned} \tag{3.9}$$

We call this method the *single variable split*, as it converts the original problem in (3.4) into the form of (3.6) by using one variable $\boldsymbol{Z}$. Such approach is very common in compressive sensing literature [84], where one converts problems of the form $\min_{\boldsymbol{x}} \ \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|^2+\lambda\|\boldsymbol{x}\|_1$ into $\min_{\boldsymbol{x},\boldsymbol{z}} \ \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|^2+\lambda\|\boldsymbol{z}\|_1$, subject to $\boldsymbol{z}=\boldsymbol{x}$.

To solve (3.9), we consider the augmented Lagrangian function

$$\begin{aligned} \mathcal{L}(\boldsymbol{W},\boldsymbol{H},\boldsymbol{Z},\boldsymbol{V}) = \ & \frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{W}\boldsymbol{H}\|_F^2+\lambda\|\boldsymbol{Z}\|_{STV} \\ & +\boldsymbol{V}^T(\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z})+\frac{\rho}{2}\|\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z}\|_F^2, \end{aligned} \tag{3.10}$$

where $\boldsymbol{V}$ is the Lagrangian multiplier. Then, we solve the following subproblems alternatively:

$$(\widehat{\boldsymbol{W}},\widehat{\boldsymbol{H}}) = \underset{\boldsymbol{W}\geq 0,\boldsymbol{H}\geq 0}{\operatorname{argmin}} \ \frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{W}\boldsymbol{H}\|_F^2+\boldsymbol{V}^T(\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z})+\frac{\rho}{2}\|\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z}\|_F^2 \tag{3.11a}$$

$$\widehat{\boldsymbol{Z}} = \underset{\boldsymbol{Z}}{\operatorname{argmin}} \ \lambda\|\boldsymbol{Z}\|_{STV}+\boldsymbol{V}^T(\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z})+\frac{\rho}{2}\|\widehat{\boldsymbol{W}}\widehat{\boldsymbol{H}}-\boldsymbol{Z}\|_F^2 \tag{3.11b}$$

$$\widehat{\boldsymbol{V}} = \boldsymbol{V}+\rho(\widehat{\boldsymbol{W}}\widehat{\boldsymbol{H}}-\widehat{\boldsymbol{Z}}). \tag{3.11c}$$

Since (3.11a) is quadratic, we can show the following result.

**Proposition 3.3.1** *The optimization problem in* (3.11a) *is equivalent to*

$$(\widehat{\boldsymbol{W}}, \widehat{\boldsymbol{H}}) = \underset{\boldsymbol{W}\geq 0, \boldsymbol{H}\geq 0}{argmin} \frac{1}{2} \left\| \boldsymbol{W}\boldsymbol{H} - \frac{\boldsymbol{Y} + \rho\boldsymbol{Z} - \boldsymbol{V}}{1+\rho} \right\|_F^2. \tag{3.12}$$

**Proof** By completing squares. ■

The result of Proposition 3.3.1 is in the same form as (3.2). Since (3.2) is the standard non-negative matrix factorization problem, we can solve (3.12) using an existing NMF solver. We also note that the problem (3.11b) is a standard STV minimization, which can be solved using an existing package [81]. To summarize the single variable split method, we show the algorithm in Algorithm 4. Here, we increase the parameter $\rho$ by $\gamma\rho$ to ensure convergence, where $\gamma > 1$.

**Algorithm 4:** Single Variable Split

```
while not converge do
    (W, H) ← Solve (3.12)
    Z ← Solve (3.11b)
    V ← V + ρ(WH − Z)
    ρ ← γρ
end
```

### 3.3.2 Multiple Variable Split

The second method we consider is to introduce multiple intermediate variables to handle both $\boldsymbol{W}\boldsymbol{H}$ and the STV of $\boldsymbol{W}\boldsymbol{H}$ simultaneously. To simplify our notations we shall consider one $\boldsymbol{D}$ instead of $(\boldsymbol{D}_x, \boldsymbol{D}_y, \boldsymbol{D}_z)$, and set $\beta = 1$. The difference between multiple variable split and single variable split is the number of intermediate variables we defined in the ADMM algorithm.

We write (3.4) as the following constrained problem:

$$\begin{aligned} \underset{\boldsymbol{W},\boldsymbol{H},\boldsymbol{Z},\boldsymbol{x},\boldsymbol{W}_+,\boldsymbol{H}_+}{\text{minimize}} \quad & \frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{W}\boldsymbol{H}\|_F^2+\lambda\|\boldsymbol{x}\|_1 \\ \text{subject to} \quad & \boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z}=\boldsymbol{0}, \boldsymbol{D}\boldsymbol{z}-\boldsymbol{x}=\boldsymbol{0}, \\ & \boldsymbol{W}=\boldsymbol{W}_+,\ \boldsymbol{H}=\boldsymbol{H}_+, \boldsymbol{W}_+\geq\boldsymbol{0},\ \boldsymbol{H}_+\geq\boldsymbol{0}. \end{aligned} \tag{3.13}$$

The first constraints in (3.13) is the same as the constraint in (3.9), which is to substitute $\boldsymbol{W}\boldsymbol{H}$. The second constraint is ensure that $\|\boldsymbol{x}\|_1=\|\boldsymbol{D}\boldsymbol{z}\|_1 \stackrel{\text{def}}{=} \|\boldsymbol{Z}\|_{STV}$. The third and the fourth constraints are used to ensure non-negativity of $\boldsymbol{W}$ and $\boldsymbol{H}$. The augmented Lagrangian function of (3.13) is

$$\begin{aligned} &\mathcal{L}(\boldsymbol{W},\boldsymbol{H},\boldsymbol{Z},\boldsymbol{x},\boldsymbol{W}_+,\boldsymbol{H}_+,\boldsymbol{V},\boldsymbol{u},\boldsymbol{P},\boldsymbol{Q}) \\ &=\frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{W}\boldsymbol{H}\|_F^2+\lambda\|\boldsymbol{x}\|_1+\boldsymbol{V}^T(\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z})+\frac{\rho}{2}\|\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z}\|_F^2 \\ &+\boldsymbol{u}^T(\boldsymbol{D}\boldsymbol{z}-\boldsymbol{x})+\frac{\mu}{2}\|\boldsymbol{D}\boldsymbol{z}-\boldsymbol{x}\|_F^2+\boldsymbol{P}^T(\boldsymbol{W}-\boldsymbol{W}_+)+\frac{\alpha_1}{2}\|\boldsymbol{W}-\boldsymbol{W}_+\|_F^2 \\ &+\boldsymbol{Q}^T(\boldsymbol{H}-\boldsymbol{H}_+)+\frac{\alpha_2}{2}\|\boldsymbol{H}-\boldsymbol{H}_+\|_F^2, \end{aligned} \tag{3.14}$$

where $\boldsymbol{V}$, $\boldsymbol{u}$, $\boldsymbol{P}$ and $\boldsymbol{Q}$ are Lagrangian multipliers, and $\rho$, $\mu$, $\alpha_1$ and $\alpha_2$ are parameters. (3.14) has been previously used in several occasions, e.g., [11], [82], [85], (usually without the STV term). The intuition behind is that by splitting (3.11a) and (3.11b) further, we can possibly reduce some computation. Algorithm 5 summarizes the method after completing squares and scaling the parameters.

### 3.3.3 Half Quadratic Penalty

The third method we consider is a simple variant of the single variable split. In this method, we remove the update of the Lagrange multiplier $\boldsymbol{V}$ in (3.11a)-(3.11c). This returns us a *half quadratic penalty* method, which solves

$$\underset{\boldsymbol{W}\geq\boldsymbol{0},\boldsymbol{H}\geq\boldsymbol{0},\boldsymbol{Z}}{\text{minimize}}\ \frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{W}\boldsymbol{H}\|_F^2+\lambda\|\boldsymbol{Z}\|_{STV}+\frac{\rho}{2}\|\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z}\|_F^2$$

**Algorithm 5:** Multiple Variable Split

**while** *not converge* **do**

$$\boldsymbol{W} \leftarrow \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{W}\boldsymbol{H}\|_F^2 + \boldsymbol{V}^T(\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z}) + \frac{\rho}{2}\|\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z}\|_F^2 + \boldsymbol{P}^T(\boldsymbol{W}-\boldsymbol{W}_+) + \frac{\alpha_1}{2}\|\boldsymbol{W}-\boldsymbol{W}_+\|_F^2$$

$$\boldsymbol{H} \leftarrow \underset{\boldsymbol{H}}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{W}\boldsymbol{H}\|_F^2 + \boldsymbol{V}^T(\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z}) + \frac{\rho}{2}\|\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z}\|_F^2 + \boldsymbol{Q}^T(\boldsymbol{H}-\boldsymbol{H}_+) + \frac{\alpha_2}{2}\|\boldsymbol{H}-\boldsymbol{H}_+\|_F^2$$

$$\boldsymbol{Z} \leftarrow \underset{\boldsymbol{Z}}{\operatorname{argmin}}\, \boldsymbol{V}^T(\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z}) + \tfrac{\rho}{2}\|\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z}\|_F^2 + \boldsymbol{u}^T(\boldsymbol{D}\boldsymbol{z}-\boldsymbol{x}) + \tfrac{\mu}{2}\|\boldsymbol{D}\boldsymbol{z}-\boldsymbol{x}\|_F^2$$

$$\boldsymbol{x} \leftarrow \underset{\boldsymbol{x}}{\operatorname{argmin}}\, \lambda\|\boldsymbol{x}\|_1 + \boldsymbol{u}^T(\boldsymbol{D}\boldsymbol{z}-\boldsymbol{x}) + \tfrac{\mu}{2}\|\boldsymbol{D}\boldsymbol{z}-\boldsymbol{x}\|_F^2$$

$$\boldsymbol{W}_+ \leftarrow \underset{\boldsymbol{W}_+\geq 0}{\operatorname{argmin}}\, \boldsymbol{P}^T(\boldsymbol{W}-\boldsymbol{W}_+) + \tfrac{\alpha_1}{2}\|\boldsymbol{W}-\boldsymbol{W}_+\|_F^2$$

$$\boldsymbol{H}_+ \leftarrow \underset{\boldsymbol{H}_+\geq 0}{\operatorname{argmin}}\, \boldsymbol{Q}^T(\boldsymbol{H}-\boldsymbol{H}_+) + \tfrac{\alpha_2}{2}\|\boldsymbol{H}-\boldsymbol{H}_+\|_F^2$$

$$\boldsymbol{V} \leftarrow \boldsymbol{V} + (\boldsymbol{W}\boldsymbol{H}-\boldsymbol{V}), \quad \boldsymbol{u} \leftarrow \boldsymbol{u} + (\boldsymbol{D}\boldsymbol{z}-\boldsymbol{x})$$

$$\boldsymbol{P} \leftarrow \boldsymbol{P} + (\boldsymbol{W}-\boldsymbol{W}_+), \quad \boldsymbol{Q} \leftarrow \boldsymbol{Q} + (\boldsymbol{H}-\boldsymbol{H}_+)$$

$$\rho \leftarrow \gamma\rho,\ \alpha_1 \leftarrow \gamma\alpha_1,\ \alpha_2 \leftarrow \gamma\alpha_2,\ \mu \leftarrow \gamma\mu.$$

**end**

by alternatingly minimizing

$$(\widehat{\boldsymbol{W}}, \widehat{\boldsymbol{H}}) = \underset{\boldsymbol{W}\geq 0, \boldsymbol{H}\geq 0}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{W}\boldsymbol{H}\|_F^2 + \frac{\rho}{2}\|\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z}\|_F^2 \tag{3.15}$$

$$\widehat{\boldsymbol{Z}} = \underset{\boldsymbol{Z}}{\operatorname{argmin}}\, \lambda\|\boldsymbol{Z}\|_{STV} + \frac{\rho}{2}\|\boldsymbol{W}\boldsymbol{H}-\boldsymbol{Z}\|_F^2. \tag{3.16}$$

Half quadratic minimization has been widely used in signal and image processing [86]-[87]. Using Proposition 3.3.1, we can derive the algorithm as shown in Algorithm 6.

**Algorithm 6:** Half Quadratic Penalty

**while** *not converge* **do**

$$(\boldsymbol{W}, \boldsymbol{H}) \leftarrow \underset{\boldsymbol{W}\geq 0, \boldsymbol{H}\geq 0}{\operatorname{argmin}}\ \tfrac{1}{2}\ \left\| \boldsymbol{W}\boldsymbol{H} - \tfrac{\boldsymbol{Y}+\rho\boldsymbol{Z}}{1+\rho} \right\|_F^2$$

$$\boldsymbol{Z} \leftarrow \text{Solve (3.16)}$$

$$\rho \leftarrow \gamma\rho$$

**end**

### 3.3.4 Algorithm-induced Prior

The fourth method we consider is a relatively new concept called algorithm-induced prior [88], [89]. The idea is to rewrite (3.11b) as

$$\widehat{\boldsymbol{Z}} = \underset{\boldsymbol{Z}}{\operatorname{argmin}} \ \frac{\rho}{2} \left\| \widehat{\boldsymbol{W}}\widehat{\boldsymbol{H}} + \frac{\boldsymbol{V}}{\rho} - \boldsymbol{Z} \right\|_F^2 + \lambda \|\boldsymbol{Z}\|_{STV}, \tag{3.17}$$

which can be done by completing squares. (3.17) can be viewed as a *denoising* step where the observed data is $\widehat{\boldsymbol{W}}\widehat{\boldsymbol{H}} + \boldsymbol{V}/\rho$, and the potential clean estimate is $\boldsymbol{Z}$.

The denoising perspective suggests us that we can replace $\|\boldsymbol{Z}\|_{STV}$ by any other prior that can lead to better denoising performance. Or more aggressively, we can replace the entire optimization in (3.17) by an off-the-shelf denoising algorithm (which we call a denoiser in general):

$$\widehat{\boldsymbol{Z}} \leftarrow \text{denoiser}\left(\widehat{\boldsymbol{W}}\widehat{\boldsymbol{H}} + \boldsymbol{V}/\rho\right).$$

Here, we use BM3D [90] as the denoiser for this problem. The overall algorithm is shown in Algorithm 7.

**Algorithm 7:** BM3D induced prior

```
while not converge do
    (W, H) ← Solve (3.12)
    Z ← BM3D (WH + V/ρ)
    V ← V + ρ(WH − Z)
    ρ ← γρ
end
```

## 3.4 Experiments and Discussions

### 3.4.1 Experimental Results

In this experiment, we set the internal parameters as $\mu = 1$, $\alpha_1 = 100$, $\alpha_2 = 100$, $\rho = 0.1$ and $\gamma = 1.1$. $\boldsymbol{W}$ and $\boldsymbol{H}$ are initialized with random matrices. Whenever total variation minimization is required, e.g., (3.11b), we use `deconvtv` [81] to solve, and whenever a NMF is needed, i.e., (3.2), we use our implementation of the ADMM algorithm in [11] to solve.

To create synthetic stimulated Raman scattering data so that we can compute the PSNR, we acquire a $256 \times 256 \times 50$ dimethyl sulfoxide (DMSO) solution with 100% concentration. We then apply a standard NMF to the reshaped data, i.e., a $65536 \times 50$ matrix, and limit the rank to $R = 5$. Denoting the decomposed matrices as $\boldsymbol{W}$ and $\boldsymbol{H}$, we generate synthetic observations $\boldsymbol{Y}$ using (3.1) by setting $\boldsymbol{\eta}$ as zero-mean iid Gaussian noise with standard deviation $\sigma = \{5/255, 10/255, 20/255\}$. The regularization parameter is set as $\lambda = \sigma^{1.1}$.

Non-negative matrix factorization is known to have permutation and scaling ambiguity. Therefore, for the estimated $\boldsymbol{W}$ (and $\boldsymbol{H}$), we compute the optimal permutation $\boldsymbol{P}$ so that columns of $\boldsymbol{WP}$ would match with that of the true $\boldsymbol{W}$. Moreover, we normalize $\boldsymbol{W}$ so that all values are in $[0, 1]$. In other words, we let

$$\widehat{\boldsymbol{W}} = \left( \frac{\boldsymbol{W} - \boldsymbol{W}_{\min}}{\boldsymbol{W}_{\max} - \boldsymbol{W}_{\min}} \right) \boldsymbol{P}, \text{ and } \widehat{\boldsymbol{H}} = \boldsymbol{P} \left( \frac{\boldsymbol{H} - \boldsymbol{H}_{\min}}{\boldsymbol{H}_{\max} - \boldsymbol{H}_{\min}} \right),$$

where $\boldsymbol{W}_{\min}$ and $\boldsymbol{W}_{\max}$ are the minimum and maximum of $\boldsymbol{W}$.

We show three sets of results in Table 3.1, namely, the PSNR of $\boldsymbol{W}$, $\boldsymbol{H}$, and $\boldsymbol{WH}$. It can be seen that the multiple-variable split method is consistently the worst, and is significantly worse than other three methods. For single-variable split, half-quadratic, and algorithm-induced prior, the performance is not much different, although the PSNR of $\boldsymbol{W}$ and $\boldsymbol{H}$ are consistently worse than that of $\boldsymbol{WH}$ (which is reasonable because we only regularize $\boldsymbol{WH}$).

Table 3.1.: PSNR of the algorithms at $\sigma = 5/255, 10/255, 20/255$.

| sigma | | Single Variable Split | Multiple Variable Split | Half Quadratic Penalty | Algorithm induced Prior |
|---|---|---|---|---|---|
| $\frac{5}{255}$ | $\boldsymbol{W}$ | 36.16 | 10.30 | 37.73 | 34.61 |
| | $\boldsymbol{H}$ | 41.01 | 16.15 | 48.57 | 38.54 |
| | $\boldsymbol{WH}$ | 48.35 | 16.62 | 49.02 | 46.21 |
| $\frac{10}{255}$ | $\boldsymbol{W}$ | 34.57 | 10.30 | 35.34 | 34.53 |
| | $\boldsymbol{H}$ | 40.61 | 16.13 | 42.77 | 40.52 |
| | $\boldsymbol{WH}$ | 46.32 | 16.71 | 46.22 | 45.91 |
| $\frac{20}{255}$ | $\boldsymbol{W}$ | 32.70 | 10.31 | 32.33 | 32.57 |
| | $\boldsymbol{H}$ | 41.01 | 16.09 | 37.93 | 38.85 |
| | $\boldsymbol{WH}$ | 44.94 | 16.93 | 43.98 | 45.07 |

In Figure 3.2 we show how the PSNR of $\boldsymbol{H}$ iterates for each algorithm. Generally speaking, all algorithms demonstrate an improving PSNR, despite several spiky behavior of the single-variable split and the algorithm-induced prior. The exact reason of the spikes is unknown, but we suspect it is related to the non-convexity of NMF.

### 3.4.2 Discussion

The results of this experiment has two implications. First, multiple-variable split should be avoided, especially when the problem is non-convex. Intuitively, the key difference between the three methods and multiple-variable split is that the formers are separate denoise-then-factorize processes whereas the latter is a combined denoise-factorize process. Since geometrically NMF is seeking support vectors that define a non-negative cone [91], a separate denoise-then-factorize process allows us to shrink the uncertainty caused by the noise before factorization. A combined denoise-factorize process, on the other hand, is more like to cause oscillation.

Second, single-variable split and half-quadratic penalty have similar behavior except that single-variable split usually converges faster due to the Lagrange multiplier. Algorithm-induced prior has some unpredictable behavior. This is most likely caused by the nonlinearity of BM3D. However, looking at the PSNR at $\sigma = 20/255$, al-

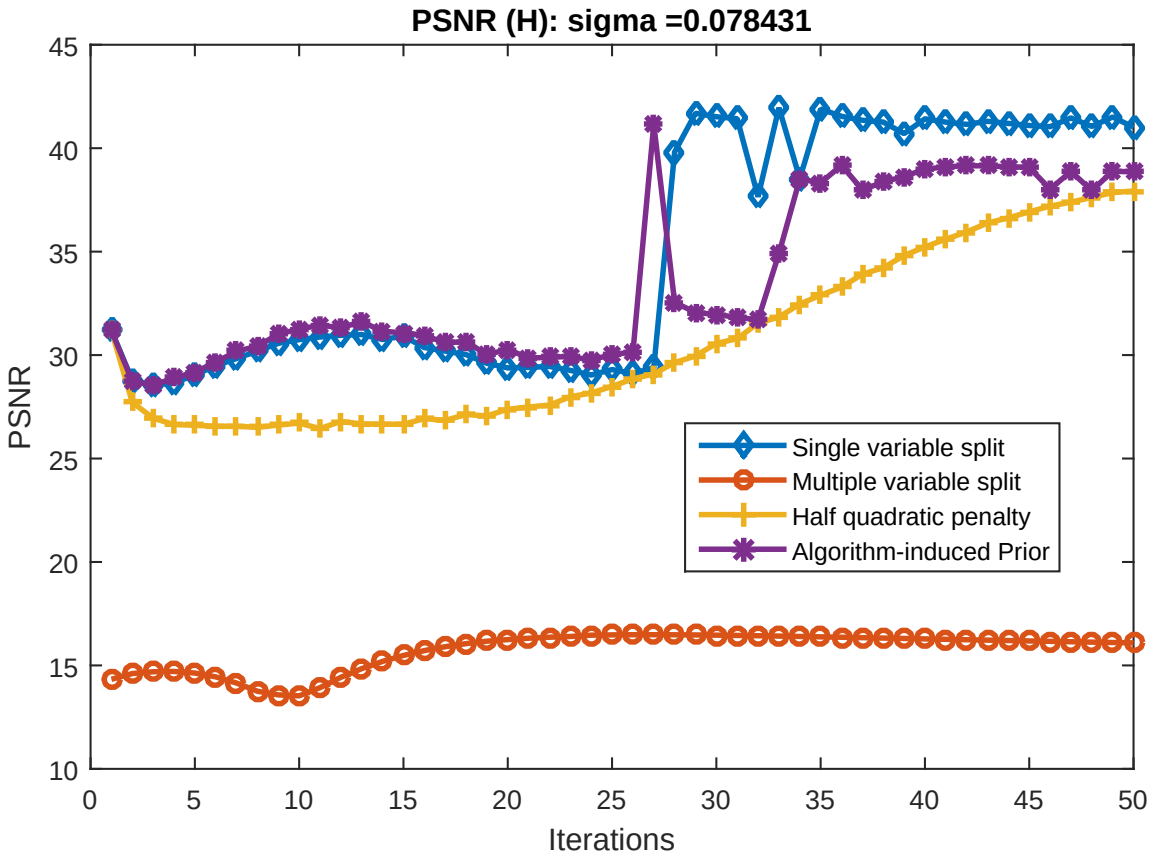



Fig. 3.2.: PSNRs of $\boldsymbol{H}$ at every iteration. $\sigma = 20/255$.

gorithm induced prior seems to give better $\boldsymbol{WH}$, implying that the denoiser has a strong influence on high noise settings.

### 3.5 Conclusion

We studied four variants of the ADMM algorithm to analyze and compare their performances of non-negative matrix factorization in the presence of noise. Since non-negative matrix factorization is non-convex but bi-convex, i.e. convex in a matrix when the other matrix fixed, one has to be careful when choosing the splitting strategy. We found from the experiment that the multiple-variable split can result in bad performance, whereas single variable split, half quadratic penalty and algorithm-induced priors lead to similar performance. We argue that the disparity in the perfor-

mance is caused by the extent to which the denoising part and the factorization part of the ADMM can reach. We also demonstrated the potential of algorithm-induced prior.

# PART II

# DEEP LEARNING IMAGE DENOISING

# 4. OPTIMAL COMBINATION OF IMAGE DENOISERS

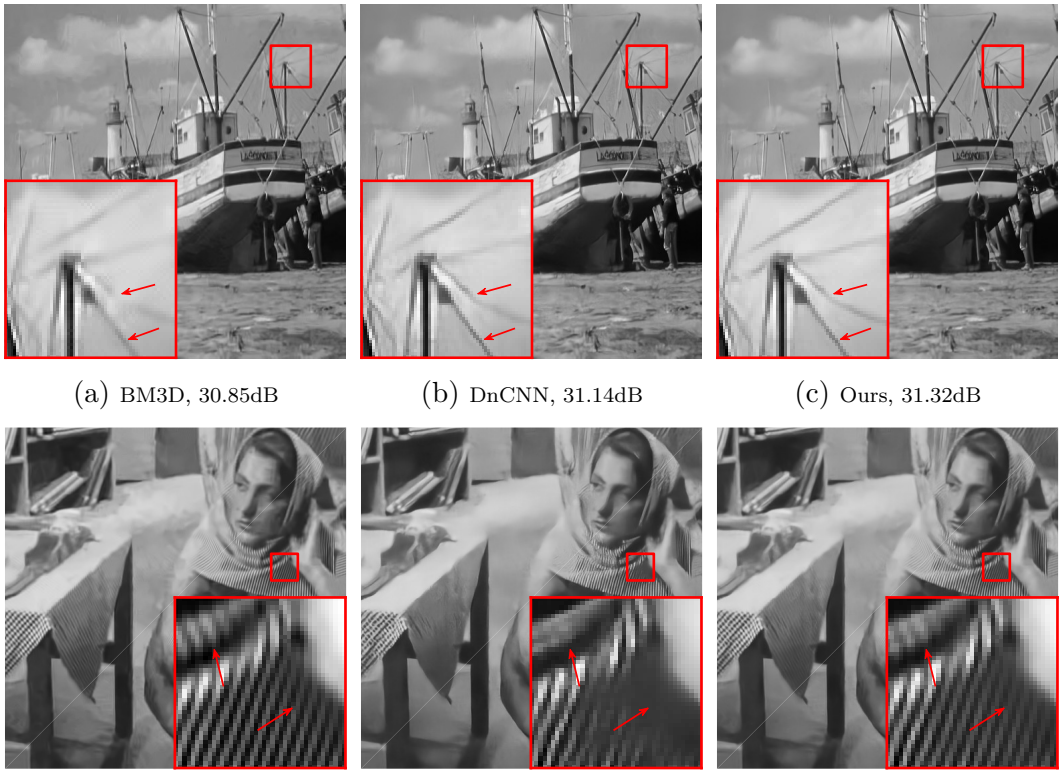
This Chapter addresses given a set of denoisers the optimal combination with two modules: MSE Estimator and Booster network. It is based on a paper to be submitted to IEEE Transactions on Image Processing [92].

## 4.1 Introduction

While image denoising algorithms over the past decade have produced very promising results, it is also safe to say that there is no single image denoiser can perform uniformly better than other denoisers. In fact, any image denoiser, either deterministic [19–23] or learning-based [12–18], has an implicit prior model that determines its denoising characteristics. Since a particular prior model encapsulates the statistics of a limited set of imaging conditions, the corresponding denoiser is only an expert for the images that it is designed to handle. We refer to this gap between the imaging model and the denoising task as a model mismatch.

Model mismatch is common in practice. The followings are three examples:

- Denoiser Characteristic: Every denoiser has a different characteristic. For example, total variation assumes sparsity of the gradients, thus works well for piecewise constant images; BM3D [20] assumes patch reoccurrence, thus works well for images with repeated patterns. Figure 4.1 shows an example of BM3D [20] and a neural network denoiser DnCNN [16]. For `Boat512`, it is clear that DnCNN performs better. However, for `Barbara512`, BM3D actually outperforms DnCNN due to the weak oscillating pattern on the cloth, a rare feature that is difficult to learn.



(a) BM3D, 30.85dB (b) DnCNN, 31.14dB (c) Ours, 31.32dB

(d) BM3D, 26.80dB (e) DnCNN, 26.49dB (f) Ours, 26.80dB

Fig. 4.1.: Comparison of BM3D [20], DnCNN [16] and the proposed CsNet.

- Noise Level: For neural network image denoisers, the performance is determined by the noise level under which the denoiser is trained. For example, if a denoiser is trained for i.i.d. Gaussian noise of standard deviation $\sigma$, it only works well for this particular $\sigma$. As soon as the noise level deviates, the performance will degrade.

- Image Class: A denoiser could be well trained for a particular class of images (e.g., building), but it may not work for other classes (e.g., face). For this type of class-aware issue, the typical solution is by means of scene classification. However, scene classification itself is an open problem and there is no consensus of the best approach. Therefore, it would be more convenient if the denoiser can automatically pick a class that gives the best performance without seeking classification algorithms.

The common question underlying these examples is that if we have a set of denoisers, each having a different characteristic, how do we combine them to produce a better result? Answering this question is fundamental to designing ensembles of expert image restoration methods for complex scenes. The goal of this work is to present a framework called the Consensus Neural Network (CsNet) which seeks consensus by using neural networks and convex optimization.

### 4.1.1 Related Work

Combining estimators is a long-lasting statistical problem. In as early as 1959, Graybill and Deal [93] started to consider linearly combining two unbiased scalar estimators to yield a new estimator that remains unbiased and has lower variance. More properties of the such combination scheme was discussed by Samuel-Cahn [94]. In [95], Rubin and Weisberg extended the idea by estimating weights from the samples. However, the estimators are still scalars and are assumed to be independent. Correlated scalar estimators are later studied by Keller and Olkin [96]. For vector estimators (which is the case for image denoisers), Odell et al. [97] presented a very comprehensive study. However, their result is limited to two vector estimators. The general case of multiple estimators is studied by Lavancier and Rochet [98], who proposed an optimization approach to estimate the weights.

Specific to image denoising, methods seeking linear combination of denoisers are scattered in the literature. The most popular approach is perhaps the linear expansion of thresholds by Blu and colleagues [99], using the Stein's unbiased risk estimator (SURE). In [100], Chaudhury et al. presented an improved bilateral filter using the SURE estimator. For learning based methods, the loss-specific training approach by Jancsary et al. [101] presented a regression tree field model to optimize the denoising performance over different metrics. There is also an end-to-end neural network solution for selecting denoisers by Agostinelli et al. [102], where the authors proposed to learn the weights using an auto-encoder.

The noise-level mismatch is discussed more often in the neural network literature. Conventional approach is to either truncate the noise level to the nearest trained level [103] or to train the network with a large number of examples covering all noise levels [16]. A more recent approach is to feed a noise map to the network and train the network to recognize the noise level [18]. However, this approach requires a redesign of the network structure. In contrast, CsNet uses the same structure for all initial denoisers.

### 4.1.2 Contributions

An overview of the proposed CsNet framework is shown in Figure 4.2. We summarize the three key contributions of this chapter in the followings:

- MSE Estimator. We present a novel deep neural network to estimate the mean square error (MSE) in the absence of the ground truth. Existing deep neural network based image quality assessment methods are designed to predict perceptual quality and not MSE. To the best of our knowledge, our deep learning based MSE estimator is the first of this kind in the literature.

- Optimal Combination. We present an optimal combination framework via convex optimization. By minimizing a quadratic function over a unit simplex, we prove that resulting combination is optimal in the MSE sense. We provide geometric interpolation of the solution, and a fast algorithm to determine the optimal point.

- Denoising Booster. We present a new deep neural network to boost the combined estimates. Unlike the existing deterministic boosters which are iterative, we cascade several simple neural networks to build the booster.

To help readers understand the design process, we proceed the chapter by first discussing the optimal combination and its associated theoretical properties in Section 4.2. Section 4.3 discusses the neural network estimator for estimating the MSE.

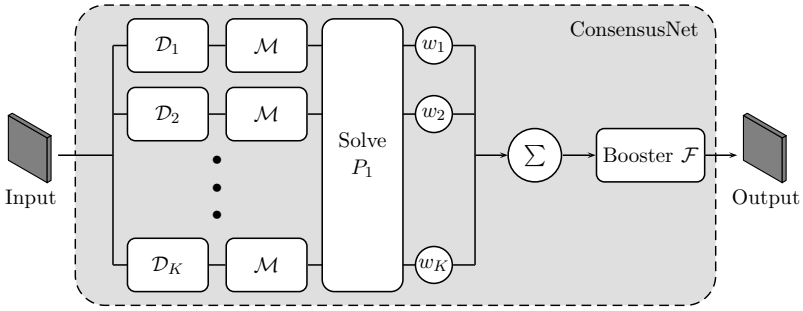



Fig. 4.2.: Structure of the proposed CsNet: Given a set of $K$ initial denoisers $\mathcal{D}_1, \ldots, \mathcal{D}_K$, CsNet uses an MSE estimator ($\mathcal{M}$) to estimate the MSE of each initial denoiser. After the MSEs are estimated, we solve a convex optimization problem ($P_1$) to determine the optimal weight $w_1, \ldots, w_K$. The combined estimate is then boosted using a booster neural network to improve contrast and details.

We emphasize that the neural network presented here is just one of the many possible ways of estimating the MSE. Readers preferring non-training based approaches can use estimators such as SURE, although we will provide examples where SURE does not work. Section 4.4 discusses the booster, and its cascade structure. Experiments are discussed in Section 4.5.

### 4.1.3 Notation

Throughout this chapter, we use lower case bold letters to denote vectors, e.g., $\boldsymbol{x} \in \mathbb{R}^N$, and upper case bold letters to denote matrices, e.g., $\boldsymbol{X} \in \mathbb{R}^{K \times K}$. An all-one vector is denoted as $\mathbf{1}$. Standard basis vectors are denoted as $\boldsymbol{e}_i$, i.e., $\boldsymbol{e}_i = [0, \ldots, 1, \ldots, 0]^T$. For any vector $\boldsymbol{x}$, $\|\boldsymbol{x}\|_2$ means the $\ell_2$-Euclidean norm, and for any matrix $\boldsymbol{A}$, $\|\boldsymbol{A}\|_2 = \max_{\|\boldsymbol{x}\|_2=1} \|\boldsymbol{A}\boldsymbol{x}\|_2$ denotes the matrix operator norm. To specify that a vector $\boldsymbol{x}$ is non-negative for all its elements, we write $\boldsymbol{x} \succeq 0$. For matrices, $\boldsymbol{A} \succeq 0$ means that $\boldsymbol{A}$ is positive semi-definite. Images in this chapter are normalized so that every pixel is in $[0, 1]$. Noise level of an i.i.d. Gaussian noise is specified by its standard deviation $\sigma$. For notational simplicity, we write $\sigma$ in the scale of $[0, 255]$, e.g., "$\sigma = 20$" means $\sigma = 20/255$. Finally, an image denoiser $\mathcal{D}$ is a mapping $\mathcal{D} : [0, 1]^N \to [0, 1]^N$. We assume $\mathcal{D}$ is bounded and is asymptotically invariant [104].

## 4.2 Optimal Combination of Estimators

### 4.2.1 Problem Formulation

Consider a linear forward model where a clean image $\boldsymbol{z} \in \mathbb{R}^N$ is corrupted by additive i.i.d. Gaussian noise $\boldsymbol{\eta} \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$ so that the observed image is $\boldsymbol{y} = \boldsymbol{z} + \boldsymbol{\eta}$. We apply a set of $K$ image denoisers $\mathcal{D}_1, \ldots, \mathcal{D}_K$ to yield $K$ initial estimates $\widehat{\boldsymbol{z}}_k = \mathcal{D}_k(\boldsymbol{y})$ for $k = 1, \ldots, K$. For convenience, we concatenate these initial estimates by constructing a matrix $\widehat{\boldsymbol{Z}} = [\widehat{\boldsymbol{z}}_1, \ldots, \widehat{\boldsymbol{z}}_K] \in \mathbb{R}^{N \times K}$. In this chapter, we focus on the *linear combination* of estimators. That is, for a given $\widehat{\boldsymbol{Z}}$, we construct the linearly combined estimate as

$$\widehat{\boldsymbol{z}} = \sum_{k=1}^{K} w_k \widehat{\boldsymbol{z}}_k = \widehat{\boldsymbol{Z}} \boldsymbol{w}, \tag{4.1}$$

where $\boldsymbol{w} \stackrel{\text{def}}{=} [w_1, \ldots, w_K]^T \in \mathbb{R}^K$ is the vector of combination weights. The goal of our work is to formulate an optimization problem to determine the optimal weights.

For analytic tractability, we use mean squared error (MSE) to measure the optimality, although it is known that alternative visual quality metrics correlate better to human visual systems [105]. Denote $\boldsymbol{z} \in \mathbb{R}^N$ as the ground truth. We define the MSE between the combined estimate $\widehat{\boldsymbol{z}}$ and the ground truth $\boldsymbol{z}$ as

$$\mathrm{MSE}(\widehat{\boldsymbol{z}}, \boldsymbol{z}) \stackrel{\text{def}}{=} \mathbb{E}\left[\|\widehat{\boldsymbol{z}} - \boldsymbol{z}\|^2\right] = \mathbb{E}\left[ \left\| \widehat{\boldsymbol{Z}} \boldsymbol{w} - \boldsymbol{z} \right\|^2 \right]. \tag{4.2}$$

The optimal combination problem can be posed as minimizing the MSE by seeking the weight vector $\boldsymbol{w} \in \mathbb{R}^K$:

$$\begin{aligned} \underset{\boldsymbol{w}}{\text{minimize}} \quad & \mathbb{E}\left[\|\widehat{\boldsymbol{Z}} \boldsymbol{w} - \boldsymbol{z}\|^2\right] \\ \text{subject to} \quad & \boldsymbol{w}^T \mathbf{1} = 1, \quad \text{and} \quad \boldsymbol{w} \succeq 0. \end{aligned} \tag{4.3}$$

Here, the constraint $\boldsymbol{w}^T \mathbf{1} = 1$ ensures that the sum of the weight is 1, and the constraint $\boldsymbol{w} \succeq 0$ ensures that the combined estimate remains in $[0, 1]^N$.

Let us simplify (4.3). First, we define $\boldsymbol{Z} = [\boldsymbol{z}, \ldots, \boldsymbol{z}] \in \mathbb{R}^{N \times K}$, i.e., a matrix with the ground truth $\boldsymbol{z}$ in each column. Since $\boldsymbol{w}^T \mathbf{1} = 1$, we can show that

$$\begin{aligned}
\mathbb{E}\left[ \left\| \widehat{\boldsymbol{Z}}\boldsymbol{w} - \boldsymbol{z} \right\|^2 \right] &= \mathbb{E}\left[ \left\| \widehat{\boldsymbol{Z}}\boldsymbol{w} - \boldsymbol{Z}\boldsymbol{w} \right\|^2 \right] \\
&= \mathbb{E}\left[ \boldsymbol{w}^T (\widehat{\boldsymbol{Z}} - \boldsymbol{Z})^T (\widehat{\boldsymbol{Z}} - \boldsymbol{Z}) \boldsymbol{w} \right] \\
&= \boldsymbol{w}^T \boldsymbol{\Sigma} \boldsymbol{w},
\end{aligned}$$

where $\boldsymbol{\Sigma}$ is defined as

$$\boldsymbol{\Sigma} \stackrel{\text{def}}{=} \mathbb{E}\left[ (\widehat{\boldsymbol{Z}} - \boldsymbol{Z})^T (\widehat{\boldsymbol{Z}} - \boldsymbol{Z}) \right].$$

We call $\boldsymbol{\Sigma}$ the covariance matrix[1]. Using this result, it can be shown that (4.3) is equivalent to

$$\begin{aligned}
\underset{\boldsymbol{w}}{\text{minimize}} \quad & \boldsymbol{w}^T \boldsymbol{\Sigma} \boldsymbol{w} \\
\text{subject to} \quad & \boldsymbol{w}^T \mathbf{1} = 1, \quad \text{and} \quad \boldsymbol{w} \succeq 0,
\end{aligned} \tag{$P_1$}$$

which is a convex problem because $\boldsymbol{\Sigma}$ is positive semi-definite and the feasible set is convex.

Before we discuss how to solve ($P_1$), we should first discuss how to obtain $\boldsymbol{\Sigma}$. The $(i,i)$-th entry of $\boldsymbol{\Sigma}$ is

$$\Sigma_{ii} = \mathbb{E}\left[ \|\widehat{\boldsymbol{z}}_i - \boldsymbol{z}\|^2 \right] \stackrel{\text{def}}{=} \text{MSE}_i,$$

which is the MSE of the $i$-th estimate. The $(i,j)$-th entry of $\boldsymbol{\Sigma}$ is the correlation between $\widehat{\boldsymbol{z}}_i$ and $\widehat{\boldsymbol{z}}_j$:

$$\Sigma_{ij} = \mathbb{E}\left[ (\widehat{\boldsymbol{z}}_i - \boldsymbol{z})^T (\widehat{\boldsymbol{z}}_j - \boldsymbol{z}) \right].$$

[1]Straightly speaking, $\boldsymbol{\Sigma} \stackrel{\text{def}}{=} \mathbb{E}\left[ (\widehat{\boldsymbol{Z}} - \boldsymbol{Z})^T (\widehat{\boldsymbol{Z}} - \boldsymbol{Z}) \right]$ is not the conventional covariance matrix because denoisers are not necessarily unbiased, i.e., $\mathbb{E}[\widehat{\boldsymbol{Z}}] \neq \boldsymbol{Z}$.

To express $\Sigma_{ij}$ in terms of $\text{MSE}_i$ and $\text{MSE}_j$, we notice that

$$\begin{aligned}\mathbb{E}\left[\|\widehat{\boldsymbol{z}}_i - \widehat{\boldsymbol{z}}_j\|^2\right] &= \mathbb{E}\left[\|\widehat{\boldsymbol{z}}_i - \boldsymbol{z} + \boldsymbol{z} - \widehat{\boldsymbol{z}}_j\|^2\right] \\ &= \mathbb{E}\,\|\widehat{\boldsymbol{z}}_i - \boldsymbol{z}\|^2 + \mathbb{E}\,\|\widehat{\boldsymbol{z}}_j - \boldsymbol{z}\|^2 + \dots \\ &\quad - 2\mathbb{E}\left[(\widehat{\boldsymbol{z}}_i - \boldsymbol{z})^T(\widehat{\boldsymbol{z}}_j - \boldsymbol{z})\right].\end{aligned}$$

Rearranging the terms we can write $\Sigma_{ij}$ as

$$\Sigma_{ij} = \frac{\text{MSE}_i + \text{MSE}_j - \mathbb{E}\left[\|\widehat{\boldsymbol{z}}_i - \widehat{\boldsymbol{z}}_j\|^2\right]}{2}. \tag{4.4}$$

Therefore, when we do not have the true $\text{MSE}_i$ and $\text{MSE}_j$ but only the estimates $\widetilde{\text{MSE}}_i$ and $\widetilde{\text{MSE}}_j$, (4.4) provides a convenient way to construct $\Sigma_{ij}$ because $\mathbb{E}\left[\|\widehat{\boldsymbol{z}}_i - \widehat{\boldsymbol{z}}_j\|^2\right]$ does not require the ground truth.

### 4.2.2 Solving $(P_1)$

The optimization problem in $(P_1)$ is a quadratic minimization over a unit simplex. It is known that such problem does not have a closed form solution. Iterative algorithms are available, e.g., using general purpose semi-definite programming such as CVX [106, 107], or using projected gradients [108, 109]. However, since $(P_1)$ has a simple structure, efficient algorithms can be derived.

Our algorithm is an accelerated gradient method following from the work of Jaggi [110]. We briefly describe the algorithm for completeness. Let

$$f(\boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w} \tag{4.5}$$

be the objective function, and

$$\Omega \stackrel{\text{def}}{=} \{\boldsymbol{w} \mid \boldsymbol{w}^T\mathbf{1} = 1, \text{ and } \boldsymbol{w} \succeq 0\} \tag{4.6}$$

be the feasible set. The first order linear approximation at the $t$-th iterate is

$$f(\boldsymbol{y}) = f(\boldsymbol{w}^{(t)}) + \nabla f(\boldsymbol{w}^{(t)})^T(\boldsymbol{y} - \boldsymbol{w}^{(t)}), \quad \forall \boldsymbol{y} \in \Omega.$$

Thus, for any $\boldsymbol{y} \in \Omega$, $\boldsymbol{y} - \boldsymbol{w}^{(t)}$ is a feasible search direction. One choice of $\boldsymbol{y}$ is to make $\nabla f(\boldsymbol{w}^{(t)})^T\boldsymbol{y}$ minimized so that $f(\boldsymbol{y})$ has a lower cost. This leads to

$$\underset{\boldsymbol{y}\in\Omega}{\text{minimize}} \quad \nabla f(\boldsymbol{w}^{(t)})^T\boldsymbol{y}, \tag{4.7}$$

which has a linear objective function. Once $\boldsymbol{y}$ is determined, we construct a standard accelerated gradient step:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \alpha(\boldsymbol{y} - \boldsymbol{w}^{(t)}), \tag{4.8}$$

where $\alpha = \frac{2}{t+2}$ is the step size.

It remains to find out an solution for the subproblem (4.7). Note that the subproblem (4.7) is a linear programming over the unit simplex. Therefore, the solution has to lie on one of the vertices. Proposition 4.2.1 provides a full characterization. The pseudo-code is shown in Algorithm 8.

**Proposition 4.2.1** *The solution to* (4.7) *is* $\boldsymbol{y} = \boldsymbol{e}_{i^*}$*, where* $i^* = \text{argmin}_i(\nabla f(\boldsymbol{w}^{(t)}))_i$.

**Proof** Let $\boldsymbol{g} = \nabla f(\boldsymbol{w}^{(t)})$. Then it follows that

$$\boldsymbol{g}^T\boldsymbol{y} = \sum_{i=1}^{K} g_i y_i \geq g_{\min} \sum_{i=1}^{K} y_i = g_{\min},$$

where $g_{\min} = \min_i g_i$, and $\sum_{i=1}^{K} y_i = 1$ because $\boldsymbol{y} \in \Omega$. The lower bound can be attained when $\boldsymbol{y} = \boldsymbol{e}_{i^*}$, where $i^* = \text{argmin}_i \ g_i$. ■

**Example 1** *As an illustration of Algorithm 8, we compare its performance with an ADMM algorithm by Condat [108]. The reference method is CVX [106]. We repeat*

**Algorithm 8:** Algorithm to Solve $(P_1)$

1: Initialize $\boldsymbol{w}^0 = \boldsymbol{e}_1$.
2: **for** $t = 0, 1, \ldots, T_{\max}$ **do**
3: Let $i^* = \underset{i}{\operatorname{argmin}} \, (\boldsymbol{\Sigma}\boldsymbol{w}^{(t)})_i$
4: Update $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \left(\frac{2}{t+2}\right)(\boldsymbol{e}_{i^*} - \boldsymbol{w}^{(t)})$.
5: **end for**

*the experiment* 1000 *times using different random matrices* $\boldsymbol{\Sigma}$*, and take the average. As shown in Figure 4.3, Algorithm 8 converges significantly faster than [108]. In terms of runtime, Algorithm 8 takes about* 4.4 *msec, [108] takes* 13 *msec, and CVX takes* 223.1 *msec.*

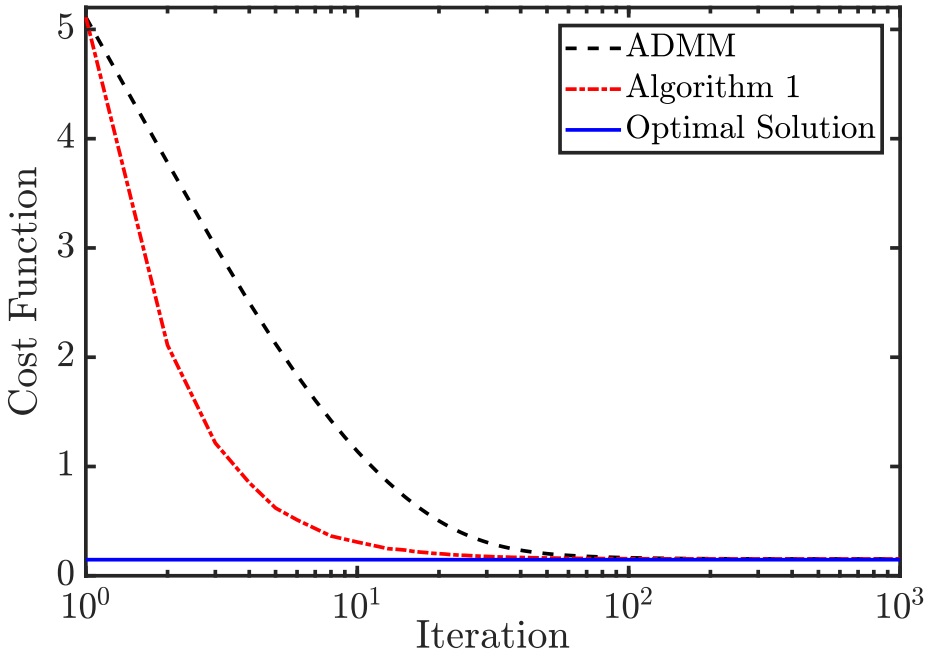



Fig. 4.3.: Comparison of Algorithm 8 and the ADMM algorithm by [108], using the optimal solution obtained by CVX [106].

### 4.2.3 Geometric Interpretation of $(P_1)$

**Uniqueness**. The uniqueness of the solution of $(P_1)$ is determined by the positive definiteness of $\boldsymbol{\Sigma}$. If $\boldsymbol{\Sigma}$ is positive definite, then $(P_1)$ is strictly convex, and hence the

optimal weight is unique. If $\mathbf{\Sigma}$ is only positive semi-definite, then there are infinitely many optimal weights. The following proposition explains this phenomenon.

**Proposition 4.2.2** *Suppose that* $\mathbf{\Sigma}$ *is positive semi-definite. Let* $\boldsymbol{w}_1^*$ *and* $\boldsymbol{w}_2^*$ *be two solutions of* $(P_1)$*. Then, for any* $0 \leq t \leq 1$*, the vector* $\boldsymbol{w}^* \stackrel{def}{=} t\boldsymbol{w}_1^* + (1-t)\boldsymbol{w}_2^*$ *is also a solution of* $(P_1)$*.*

**Proof** Let $f(\boldsymbol{w}) = \boldsymbol{w}^T\mathbf{\Sigma}\boldsymbol{w}$. Since both $\boldsymbol{w}_1^*$ and $\boldsymbol{w}_2^*$ are solutions to $(P_1)$, we have $f(\boldsymbol{w}_1^*) = f(\boldsymbol{w}_2^*)$. Also, by linearity, we have that $\mathbf{1}^T\boldsymbol{w}^* = 1$ and $\boldsymbol{w}^* \succeq 0$. Since $f$ is convex, we can show that

$$\begin{aligned} f(\boldsymbol{w}^*) &= f(t\boldsymbol{w}_1^* + (1-t)\boldsymbol{w}_2^*) \\ &\leq tf(\boldsymbol{w}_1^*) + (1-t)f(\boldsymbol{w}_2^*) = f(\boldsymbol{w}_1^*). \end{aligned}$$

But since $\boldsymbol{w}_1^*$ is an optimal solution, it is impossible for $f(\boldsymbol{w}^*) < f(\boldsymbol{w}_1^*)$. So the only possibility is $f(\boldsymbol{w}^*) = f(\boldsymbol{w}_1^*)$. This implies that $\boldsymbol{w}^*$ is also a solution. ■

The implication of Proposition 4.2.2 is that if two initial estimates $\widehat{\boldsymbol{z}}_i$ and $\widehat{\boldsymbol{z}}_j$ are identical (or scalar multiple of one and other), then $\mathbf{\Sigma}$ will have dependent columns (hence positive semi-definite). When this happens, there will be infinitely many ways of combining the two initial estimates. However, in practice this is not an issue because even if the pair $(w_i^*, w_j^*)$ is not unique, the combined estimate $w_i^*\widehat{\boldsymbol{z}}_i + w_j^*\widehat{\boldsymbol{z}}_j$ remains unique as $\widehat{\boldsymbol{z}}_i = \widehat{\boldsymbol{z}}_j$.

**Geometry**. The geometry of $(P_1)$ can be interpreted in low dimensions, e.g., Figure 4.4. In this figure, we consider a 2D case so that $\mathbf{\Sigma}$ is a $2 \times 2$ matrix. We can show that the ellipse always has its minor axis pointing to the northeast direction if the two initial estimates are positively correlated.

**Proposition 4.2.3** *Consider a two-dimensional* $\mathbf{\Sigma}$*. If* $\Sigma_{12} > 0$*, then* $\mathbf{\Sigma}$ *always has its minor axis pointing to the northeast direction and major axis to the northwest direction.*

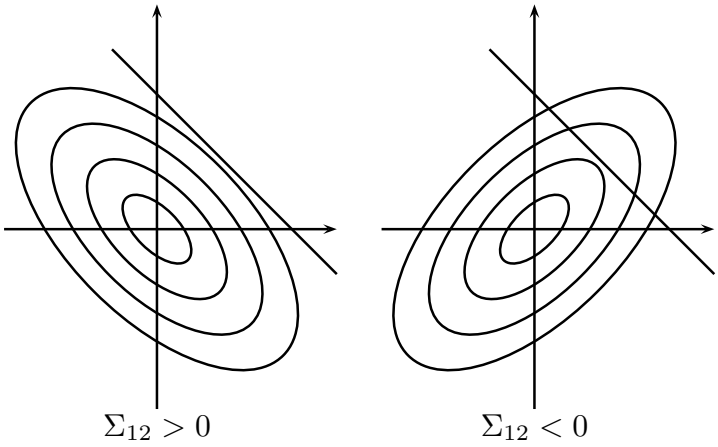



Fig. 4.4.: Geometry of the optimal weight minimization problem.

**Proof** Consider the eigen-decomposition of $\boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{U}^T$. For a $2\times2$ matrix, classical results in matrix analysis [111] shows that the eigen-value and eigen-vectors are

$$s_1 = \frac{1}{2}\left(\Sigma_{11} + \Sigma_{22} - \lambda\right), \quad s_2 = \frac{1}{2}\left(\Sigma_{11} + \Sigma_{22} + \lambda\right),$$

and

$$\boldsymbol{u}_1 = \begin{bmatrix} \frac{\Sigma_{11}-\Sigma_{22}+\lambda}{2\Sigma_{12}} \\ 1 \end{bmatrix}, \quad \boldsymbol{u}_2 = \begin{bmatrix} \frac{\Sigma_{11}-\Sigma_{22}-\lambda}{2\Sigma_{12}} \\ 1 \end{bmatrix}$$

where $\lambda = \sqrt{4\Sigma_{12}^2 + (\Sigma_{11} - \Sigma_{12})^2}$.

Note that $\lambda \geq |\Sigma_{11} - \Sigma_{22}|$ because $\Sigma_{12}^2 \geq 0$. Therefore, $s_2 \geq s_1$ and so $\boldsymbol{u}_1$ is the minor axis and $\boldsymbol{u}_2$ is the major axis. The numerator of the first entry of $\boldsymbol{u}_1$ is

$$\begin{aligned} \Sigma_{11} - \Sigma_{22} + \lambda &\geq \Sigma_{11} - \Sigma_{22} + |\Sigma_{11} - \Sigma_{22}| \\ &= \begin{cases} 2|\Sigma_{11} - \Sigma_{22}| \geq 0, & \text{if } \Sigma_{11} \geq \Sigma_{22}, \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

As a result, the numerator of the first entry of $\boldsymbol{u}_1$ is always non-negative, implying that the sign of the denominator determines the sign of the entry. Therefore, if

$\Sigma_{12} > 0$, then $\boldsymbol{u}_1$ will be pointing to the northeast direction. By orthogonality of the eigen-vectors, $\boldsymbol{u}_2$ points to the northwest direction. ■

Proposition 4.2.3 provides some insights about the solution. If $\Sigma_{12} > 0$ (which is usually the case), the major axis must point to northwest. Therefore, the solution is more likely to be at one of the two vertices. In other words, the optimal solution tends to be *sparse*. Such sparsity should come with no surprise, because the linear constraint $\boldsymbol{w}^T\mathbf{1} = 1$ is equivalent to $\|\boldsymbol{w}\|_1 = 1$ if $\boldsymbol{w} \succeq 0$.

**Remark 4.2.1** *In practice, if we only have an estimated covariance matrix $\widetilde{\boldsymbol{\Sigma}}$, there is no guarantee that $\widetilde{\boldsymbol{\Sigma}}$ is positive semi-definite. (Symmetry can be preserved by constructing the off-diagonals using* (4.4)*.) When $\widetilde{\boldsymbol{\Sigma}}$ is not positive semi-definite, we project $\widetilde{\boldsymbol{\Sigma}}$ onto its closest positive semi-definite matrix by solving*

$$\boldsymbol{\Sigma} = \underset{\boldsymbol{S} \succeq 0}{\mathit{argmin}} \ \|\boldsymbol{S} - \widetilde{\boldsymbol{\Sigma}}\|_F^2. \tag{4.9}$$

*The solution to* (4.9) *is the truncated singular value decomposition where negative singular values of $\widetilde{\boldsymbol{\Sigma}}$ are set to 0.*

### 4.2.4 Optimal MSE Lower Bound

We derive the MSE lower bound of $(P_1)$. To do so, we consider a relaxed optimization by removing the non-negativity constraint:

$$\begin{aligned} \underset{\boldsymbol{w}}{\text{minimize}} \quad & \boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w} \\ \text{subject to} \quad & \boldsymbol{w}^T\mathbf{1} = 1. \end{aligned} \tag{$P_2$}$$

Clearly, the feasible set of $(P_2)$ includes the feasible set of $(P_1)$, and so the MSE obtained by solving $(P_2)$ must be a lower bound of the MSE obtained by solving $(P_1)$.

More precisely, if we let $\widehat{\boldsymbol{w}}$ be the optimal weight vector obtained by $(P_1)$, and $\boldsymbol{w}^*$ be that obtained by $(P_2)$, then

$$\mathbb{E}\left[ \left\| \widehat{\boldsymbol{Z}}\widehat{\boldsymbol{w}} - \boldsymbol{z} \right\|^2 \right] \geq \mathbb{E}\left[ \left\| \widehat{\boldsymbol{Z}}\boldsymbol{w}^* - \boldsymbol{z} \right\|^2 \right]. \tag{4.10}$$

Let us analyze the right hand side of (4.10). The optimization in $(P_2)$ is a standard linear equality constrained quadratic minimization. Closed-form solution can be derived via the standard Lagrangian approach by defining:

$$\mathcal{L}(\boldsymbol{w}, \lambda) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w} - \lambda(\boldsymbol{w}^T\mathbf{1} - 1). \tag{4.11}$$

The first order KKT conditions state that

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = 0, \qquad \boldsymbol{w}^T\mathbf{1} = 1,$$

where the first condition is equivalent to

$$\boldsymbol{\Sigma}\boldsymbol{w} - \lambda\mathbf{1} = 0, \quad \text{or} \quad \boldsymbol{w} = \lambda\boldsymbol{\Sigma}^\dagger\mathbf{1}, \tag{4.12}$$

where $\boldsymbol{\Sigma}^\dagger$ denotes the pseudo-inverse of a symmetric positive semi-definite matrix $\boldsymbol{\Sigma}$. If $\boldsymbol{\Sigma}$ is positive definite, then $\boldsymbol{\Sigma}^\dagger = \boldsymbol{\Sigma}^{-1}$ and (4.12) can be written as $\boldsymbol{w} = \lambda\boldsymbol{\Sigma}^{-1}\mathbf{1}$. Substituting (4.12) into the constraint, we have that

$$\mathbf{1}^T\left(\lambda\boldsymbol{\Sigma}^\dagger\mathbf{1}\right) = 1 \quad \Rightarrow \quad \lambda = \frac{1}{\mathbf{1}^T\boldsymbol{\Sigma}^\dagger\mathbf{1}}. \tag{4.13}$$

Substituting (4.13) into (4.12), we prove the following.

**Proposition 4.2.4** *The solution to* $(P_2)$ *is given by*

$$\boldsymbol{w}^* = \frac{\boldsymbol{\Sigma}^\dagger\mathbf{1}}{\mathbf{1}^T\boldsymbol{\Sigma}^\dagger\mathbf{1}}, \tag{4.14}$$

*where* $\mathbf{\Sigma}^\dagger$ *denotes the pseudo-inverse of the symmetric positive semi-definite matrix* $\mathbf{\Sigma}$.

Given the optimal weight vector $\boldsymbol{w}^*$, we can determine the corresponding mean squared error:

$$\mathbb{E}\left[ \left\| \widehat{\boldsymbol{Z}}\boldsymbol{w}^* - \boldsymbol{z} \right\|^2 \right] = (\boldsymbol{w}^*)^T \mathbf{\Sigma} \boldsymbol{w}^* = \frac{1}{\mathbf{1}^T \mathbf{\Sigma}^\dagger \mathbf{1}}. \tag{4.15}$$

Since the weight $\boldsymbol{w}^*$ provides a lower bound on the MSE, in particular if we consider a weight vector $\boldsymbol{e}_k = [0, \ldots, 1, \ldots, 0]^T$ (i.e., the $k$-th standard basis vector), we must have

$$\mathrm{MSE}_k = \boldsymbol{e}_k^T \mathbf{\Sigma} \boldsymbol{e}_k \geq (\boldsymbol{w}^*)^T \mathbf{\Sigma} \boldsymbol{w}^* = \frac{1}{\mathbf{1}^T \mathbf{\Sigma}^\dagger \mathbf{1}}. \tag{4.16}$$

The inequality holds because $\boldsymbol{e}_k$ is one of the feasible vectors but $\boldsymbol{w}^*$ is the optimal solution. Therefore, an optimally combined estimate using $\boldsymbol{w}^*$ has to be no worse than any initial estimate.

**Remark 4.2.2** *The MSE lower bound result presented here is more general than the previous result by Odell et al. [97] which only considered* $K = 2$. *When* $K = 2$, *we have*

$$w_1^* = \frac{\Sigma_{22} - \Sigma_{12}}{\Sigma_{11} + \Sigma_{22} - 2\Sigma_{12}}, \quad \textit{and} \quad w_2^* = 1 - w_1^*, \tag{4.17}$$

*which is the same as Equation 2 of Table 3 in [97].* [2]

### 4.2.5 Perturbation in $\Sigma$

We conclude this section by discussing the perturbation issue when we use an estimated covariance matrix $\widetilde{\mathbf{\Sigma}}$ instead of $\mathbf{\Sigma}$. To facilitate the discussion, we define two weight vectors:

[2] In Equation 2 of Table 3 in [97], there is a typo of the numerator which should be corrected as $m_{22} - m_{12}$.

$$\widetilde{\boldsymbol{w}} = \underset{\boldsymbol{v}\in\Omega}{\operatorname{argmin}} \ \boldsymbol{v}^T\widetilde{\boldsymbol{\Sigma}}\boldsymbol{v}, \quad \text{and} \quad \boldsymbol{w} = \underset{\boldsymbol{v}\in\Omega}{\operatorname{argmin}} \ \boldsymbol{v}^T\boldsymbol{\Sigma}\boldsymbol{v}. \tag{4.18}$$

That is, $\widetilde{\boldsymbol{w}}$ is the optimal weight vector found according to the estimated covariance matrix $\widetilde{\boldsymbol{\Sigma}}$, and $\boldsymbol{w}$ is the optimal weight vector found according to the true covariance matrix $\boldsymbol{\Sigma}$. Correspondingly, we define their combined estimates as

$$\widetilde{\boldsymbol{z}} = \widehat{\boldsymbol{Z}}\widetilde{\boldsymbol{w}}, \quad \text{and} \quad \widehat{\boldsymbol{z}} = \widehat{\boldsymbol{Z}}\boldsymbol{w}. \tag{4.19}$$

The following proposition summarizes the perturbation result.

**Proposition 4.2.5** *Assume that* $\widetilde{\boldsymbol{\Sigma}} \succ 0$ *and* $\boldsymbol{\Sigma} \succ 0$. *Then,*

$$\mathbb{E}\|\widetilde{\boldsymbol{z}} - \widehat{\boldsymbol{z}}\|^2 \leq \mathbb{E}\|\widehat{\boldsymbol{z}} - \boldsymbol{z}\|^2(2\Delta + \Delta^2), \tag{4.20}$$

*where*

$$\Delta \stackrel{def}{=} \|\widetilde{\boldsymbol{\Sigma}}\boldsymbol{\Sigma}^{-1} - \widetilde{\boldsymbol{\Sigma}}^{-1}\boldsymbol{\Sigma}\|_2.$$

**Proof** The proof is given in the Appendix B. Our proof simplifies the multi-block concept of [98]. We also utilize the generalized Rayleigh quotient idea to obtain the bound. ■

The implication of Proposition 4.2.5 can be seen from the two terms on the right hand side of (4.20). First, $\mathbb{E}\|\widehat{\boldsymbol{z}}-\boldsymbol{z}\|^2$ measures the bias between the oracle combination $\widehat{\boldsymbol{z}}$ and the ground truth $\boldsymbol{z}$. That it is an upper bound in (4.20) implies that the perturbed estimate is upper limited by the bias. The second term $\Delta$ measures the closeness between the oracle covariance $\boldsymbol{\Sigma}$ and the estimated covariance $\widetilde{\boldsymbol{\Sigma}}$. If $\boldsymbol{\Sigma}\widetilde{\boldsymbol{\Sigma}}^{-1} = \boldsymbol{I}$, then $\Delta = 0$ and so the perturbation is minimized. In practice, if $\widetilde{\boldsymbol{\Sigma}}$ can be estimated in $n$ random trials and if $\boldsymbol{\Sigma}\widetilde{\boldsymbol{\Sigma}}_n^{-1} \xrightarrow{p} \boldsymbol{I}$ as $n \to \infty$, then we can also show that $\Delta \xrightarrow{p} 0$. (See Section 4.3 for an example using the SURE.)

## 4.3 MSE Estimator

The key to make $(P_1)$ success is an accurate covariance matrix $\mathbf{\Sigma}$. Estimating the covariance matrix requires estimating the mean squared error (MSE). In this section we discuss a neural network solution.

### 4.3.1 Why not SURE?

In image processing, perhaps the most popular approach for estimating MSE is the Stein's Unbiased Risk Estimator (SURE). (See, e.g., [99, 112] for illustrations, [113] for a Monte-Carlo version, and [114] for a recent work using SURE in deep neural network.) As its name suggested, SURE is an unbiased estimator of the true MSE, i.e., the estimator will approach to the true MSE as the number of samples grows.

While SURE-based estimators work well in ideal situations, it also has many shortcomings:

- Large Variance. SURE only provide *average performance* guarantee. For Monte-Carlo SURE, there is another level of randomness due to the Monte-Carlo scheme. Therefore, given a single noisy image, SURE can be inaccurate, especially for non-linear denoises such as BM3D.

- Clipped Noise. SURE is designed to handle additive i.i.d. Gaussian noise. However, most real images are clipped to $[0, 1]^N$, and most neural network denoisers clip the noise during training. If the observed image is clipped, then SURE will fail. See Figure 4.5 for an example. For more discussions of clipped noise, see [115].

- Beyond Denoisers. While SURE is a good choice for image denoising problems, one has to re-derive the SURE equations for different forward models, e.g., deblurring or super-resolution. This severely limits the generality of the present optimal combination framework.

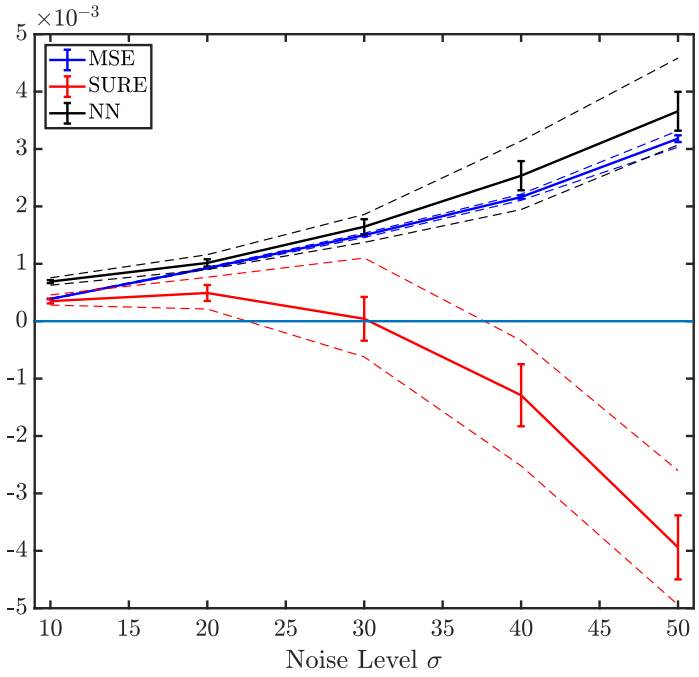



Fig. 4.5.: Clipped Noise Example. Compare SURE and the proposed neural network (NN) on estimating the MSE. In this experiment, we use BM3D to denoise the `cameraman` image. The noise level changes from $\sigma = 10$ to $\sigma = 50$. The observed images are clipped to $[0, 1]^N$. The error bars are computed using 50 random trials of the i.i.d. Gaussian noise realizations.

### 4.3.2 Neural Network MSE Estimator

Our proposed solution is a deep neural network based MSE estimator. Using deep neural networks for image quality assessment is an active research topic [116–120]. However, the existing neural network based image quality assessment methods are tailored to predict the human visual system responses when seeing an image. A pure MSE estimator, to the best of our knowledge, does not exist.

The proposed neural network based MSE estimator is shown in Figure 4.6. There are two unique features of the network. First, the input to the network is a pair of images $(\boldsymbol{y}, \widehat{\boldsymbol{z}}_k)$, i.e., the noisy observation and the $k$-th denoised image. Using both $\boldsymbol{y}$ and $\widehat{\boldsymbol{z}}_k$ is reminiscent to the SURE approach, as $\boldsymbol{y}$ provides noise statistics that cannot be obtained from $\widehat{\boldsymbol{z}}_k$ alone.

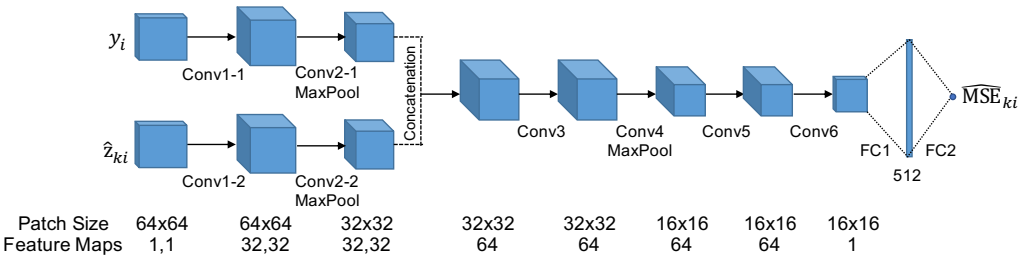



Fig. 4.6.: Network structure of a proposed MSE Estimator.

Second, instead of feeding the entire image into the network, we partition the image into non-overlapping patches of size $64 \times 64$. That is, if we denote the MSE of the $i$-th patch of the $k$-th denoiser as $\widetilde{\text{MSE}}_{k,i} \overset{\text{def}}{=} \widetilde{\text{MSE}}(\boldsymbol{y}_i, \widehat{\boldsymbol{z}_{k,i}})$, then the overall MSE of the $k$-th denoiser is

$$\widetilde{\text{MSE}}_k = \frac{1}{M} \sum_{i=1}^{M} \widetilde{\text{MSE}}(\boldsymbol{y}_i, \widehat{\boldsymbol{z}_{k,i}}),$$

where $\boldsymbol{y}_i$ is the $i$-th patch of $\boldsymbol{y}$, $\widehat{\boldsymbol{z}}_{k,i}$ is the $i$-th patch of $\widehat{\boldsymbol{z}}_k$, and $M$ is the number of non-overlapping patches in the image. Partitioning the image into small patches reduces the breath and depth of the neural network.

The network consists of 8 convolutional layers, 3 maxpool layers and 2 fully connected layers. The inputs $\boldsymbol{y}_i$ and $\widehat{\boldsymbol{z}}_{k,i}$ separately pass through two convolutional layers, and then concatenate and pass over four convolutional layers. The convoultional layers use $3 \times 3$ kernels with zero-padding and the leaky rectifier activation function (LReLU) [121]. The scale of our LReLU is 0.2, i.e., $\max(x, 0.2x)$. We apply maxpool layer with $2 \times 2$ kernel every two convoultional layer. Fully connected layers use ReLU and dropout regularization of ratio 0.5. The cost function is the $L_1$-loss, defined as

$$L = \left| \text{MSE}_{k,i} - \widetilde{\text{MSE}}_{k,i} \right| \tag{4.21}$$

where $\text{MSE}_{k,i}$ is the true MSE of $i$-th block of the $k$-th denoiser. For implementation, we use ADAM optimizer [122] with learning rate $\alpha = 0.0001$.

The training data we use is the 300 Training and Validation images in BSD500. For each image, we randomly extracted 32 patches of size $64 \times 64$ and generate 6

variations by flipping horizontally and vertically and rotating at 0°, 90°, 180° and 270°. The noise level is $\sigma \in [1, 60]$, with clipping to $[0, 1]^N$. To prepare denoised images for training the networks, we use five pre-trained REDNets [14] at noise levels $\widehat{\sigma} = 10, 20, 30, 40, 50$. Therefore, for every noisy input we generate multiple denoised images, and every denoised image forms an input-output pair with the ground truth MSE. We trained the MSE estimator network with 100 epochs for around 7 hours.

### 4.3.3 Comparison with SSDA

Readers familiar with the image denoising literature may ask about the difference between the proposed method and the AMC-SSDA method by Agostinelli et al. [102]. The AMC-SSDA method is an end-to-end neural network for denoising images of different noise types, e.g., salt-pepper, Gaussian, and Poisson. We are not interested in this problem because it is unnatural to have an image denoising problem where the noise type is totally blind. In contrast, it is more common to have multiple denoisers for different noise levels (Section 4.5.1), different image classes (Section 4.5.2), and different denoiser types (Section 4.5.3).

There are other differences. First, the SSDA has a set of fixed neural network denoisers. CsNet can adapt any initial denoisers, including both deterministic or learning-based. Second, the weight prediction of the AMC-SSDA is done using a neural network which does not have any optimality guarantee. CsNet, however, is provably optimal. Additionally, CsNet estimates the MSE (which is a scalar) from an image. This is easier than estimating the weight vector in AML-SSDA. Third, CsNet can be generalized to other estimation problems such as deblurring and super-resolution. The AMC-SSDA, however, has limited generalization capability because the initial estimators are limited to the SSDA structure.


Fig. 4.7.: Examples showing the effectiveness of the booster in improving the details and contrast of the combined result. See Section 4.5.3 for experiment details.

## 4.4 Booster Network

In our proposed CsNet, besides the convex optimization algorithm and the MSE estimator, there is a third component known as the booster. The booster is used to improve the combined estimates by enhancing the contrast and to recover lost details. To provide readers a quick preview of the booster, we show a few examples in Figure 4.7.

### 4.4.1 What is a Booster?

The concept of boosting can be traced back to as early as the 70's, when Tukey [123] suggested a "twicing procedure". In machine learning, the same concept was studied by Bühlmann and Yu [124]. The essential step of boosting is simple: Given a current estimate $\widehat{\boldsymbol{z}}^{(t)}$ and the observation $\boldsymbol{y}$, we construct a mapping $\mathcal{B}: \mathbb{R}^N \to \mathbb{R}^N$ (usually another denoising algorithm), and then define the next estimate $\widehat{\boldsymbol{z}}^{(t+1)}$ in

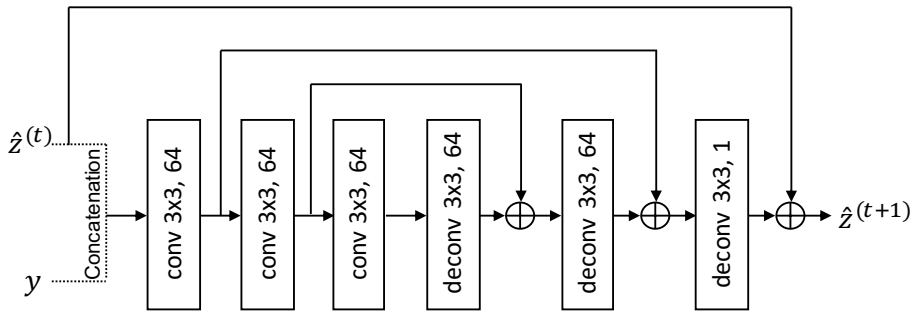



Fig. 4.8.: Network structure of the proposed booster network. The network contains 3 convolutional layers followed by 3 deconvolutional layers. Convolutional and deconvolutional layers consists of residual neural network blocks. Skip connections are used to enforce symmetry of the network. This network is repeated three times ($t = 1, 2, 3$).

terms of $\widehat{\boldsymbol{z}}^{(t)}$, $\boldsymbol{y}$ and $\mathcal{B}$ with the goal to improve the MSE. In Tukey's "twicing", the relationship between $\widehat{\boldsymbol{z}}^{(t)}$ and $\widehat{\boldsymbol{z}}^{(t+1)}$ is

$$\widehat{\boldsymbol{z}}^{(t+1)} = \mathcal{B}(\boldsymbol{y} - \widehat{\boldsymbol{z}}^{(t)}) + \widehat{\boldsymbol{z}}^{(t)}. \tag{4.22}$$

Thus, if $\mathcal{B}$ is a denoiser, then $\mathcal{B}(\boldsymbol{y}-\widehat{\boldsymbol{z}}^{(t)})$ is the filtered version of the residue. As shown in [125], MSE is not monotonically decreasing as $t \to \infty$ because of the bias-variance trade-off. However, with proper monitoring such as cross-validation, MSE can be minimized by stopping the boosting procedure before saturation. (See additional discussion for the image denoising problem in [126].)

In the image denoising literature, the above idea of boosting has been studied in multiple places such as [125–127]. There are several variations, e.g., Osher's iterative regularization [128], and Romano and Elad's SOS [129]. In all these boosting methods, the idea is the take the noisy input and the estimate $\widehat{\boldsymbol{z}}^{(t)}$ to recursively update the estimate.

### 4.4.2 Deep Learning based Booster

Our proposed neural network booster is motivated by the above examples of classical boosters. The specific network architecture is shown in Figure 4.8. Instead of

using a deterministic function $\mathcal{B}$, we use a multi-layer neural network as the building block of the booster. We then cascade the building blocks to form an overall booster.

Referring to Figure 4.8, if we denote the $t$-th building block as $\mathcal{B}_t$, then the input-output relationship of $\mathcal{B}_t$ is

$$\widehat{\boldsymbol{z}}^{(t+1)} = \mathcal{B}_t(\boldsymbol{y}, \widehat{\boldsymbol{z}}^{(t)}) + \widehat{\boldsymbol{z}}^{(t)}. \tag{4.23}$$

Clearly, (4.23) is a generalization of (4.22) as $\mathcal{B}_t$ now becomes a nonlinear mapping trained from the data. Also, when cascading a sequence $\{\mathcal{B}_t\}$, we generalize (4.22) by allowing each $\mathcal{B}_t$ to have its own network weights.

The architecture of the $t$-th building block $\mathcal{B}_t$ consists of 3 convolutional layers followed by 3 deconvolutional layers, each using kernels of size $3 \times 3$. The input to the network is the pair $(\boldsymbol{y}, \widehat{\boldsymbol{z}}^{(t)})$, which is concatenated to form a common input. The convolutional layers are used to smooth out the noisy input $\boldsymbol{y}$, whereas the deconvolutional layers are used to recover the sharp details. Skip connections are used to ensure that signals are not attenuated as it passes through the layers. Note that we purposely add a skip connection from the input $\widehat{\boldsymbol{z}}^{(t)}$ to the output $\widehat{\boldsymbol{z}}^{(t+1)}$ to mimic the addition in (4.22). We cascade $\mathcal{B}_t$ for $t = 1, \ldots, T$, where $T$ is typically small ($T = 3$).

The training data we use is the 300 train and validation images in BSD500. We extract 32 patches of size $64 \times 64$ from each training dataset. For each patch we generate 6 variations by flipping horizontally and vertically and rotating at 0°, 90°, 180° and 270°. The cost function we use in training the booster network is the standard $L_1$-loss:

$$L = \|\boldsymbol{z} - \widehat{\boldsymbol{z}}^{(T)}\|_1 \tag{4.24}$$

where $\text{MSE}_{k,i}$ is the true MSE of $i$-th block of the $k$-th denoiser. During the training, we use ADAM optimizer with learning rate $10^{-4}$. We trained booster network with 100 epochs for 12 hours.

### 4.4.3 Performance of Booster

The effectiveness of the booster can be seen in Figure 4.7, where we show a few examples taken from the BSD500 dataset. In this example, we consider a neural network denoiser trained at five different noise levels (See Section 4.5.3 for experiment details).

As we see in Figure 4.7, the booster is doing particularly well for two types of improvements. The first type of improvement is the recovery of the fine details. For example, in the Swam image we can recover the lines on the feather; in the House image we can recover branches of the tree. These are also reflected in the PSNR. The second type of improvement is the contrast enhancement. For example, before boosting the House image we see that the background sky has a gray-ish intensity. However, after boosting the background sky has a brighter background.

## 4.5 Experiments

We build our neural networks using Tensorflow and are run on Intel(R) Core(TM) i5-4690K CPU 3.50GHz with an Nvidia Titan-X GPU, except DnCNN which is downloaded from the author's website.

### 4.5.1 Experiment 1: Noise-Level Mismatch

There are two objectives of this experiment. First, we want to evaluate the effectiveness of CsNet in interpolating denoising performance when the initial denoisers are not trained for every noise level. Second, we want to compare the performance of CsNet with existing blind denoisers such as [16] because these denoisers can handle multiple noise level.

Regarding the initial denoisers, we use the 300 training and validation images in BSD500 to train five initial denoisers $\mathcal{D}_1, \ldots, \mathcal{D}_5$ using two neural network denoisers: DnCNN [16] and REDNet [14]. For each denoiser, the denoising strength is set as

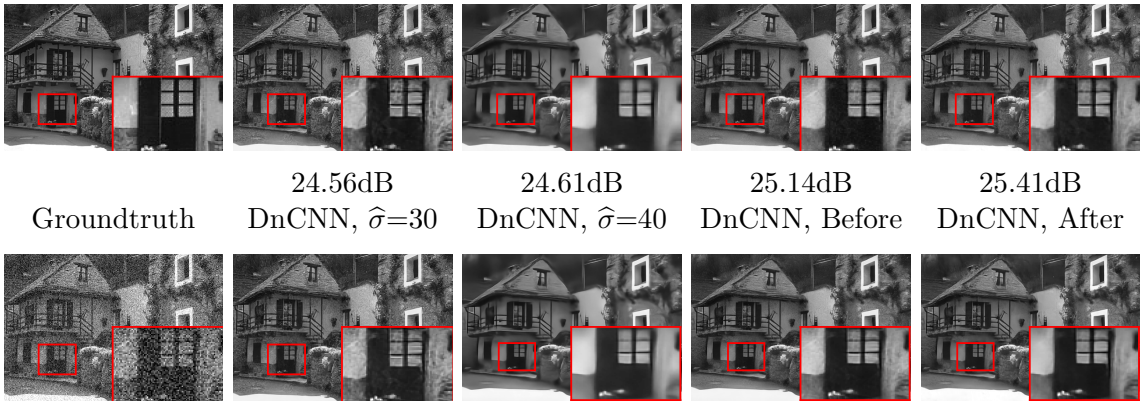



Fig. 4.9.: Example of Noise-level mismatch. The image is `House` (size 321×481) from BSD500. The actual noise level is $\sigma = 35$. Before and After means Before Booster and After Booster, respectively. DnCNN, Before and After uses five DnCNN initial denoisers, and REDNet, Before and After uses five REDNet initial denoisers.

Table 4.1.: Example of Noise-level mismatch. The average PSNRs of REDNet ($\widehat{\sigma} = 10, 20, 30, 40, 50$), Blind REDNet with 50 layers and ConsensusNet on 200 test images from BSD500.

| $\sigma$ | RED (10) | RED (20) | RED (30) | RED (40) | RED (50) | Before (NN) | After (NN) | Before (ora) | After (ora) | RED Blind |
|---|---|---|---|---|---|---|---|---|---|---|
| 10 | **34.14** | 30.69 | 28.25 | 26.83 | 25.86 | **34.08** | 33.92 | **34.14** | 33.91 | 33.77 |
| 15 | 28.43 | **30.75** | 28.30 | 26.84 | 25.85 | 31.32 | **31.79** | 31.39 | **31.80** | 31.61 |
| 20 | 24.43 | **30.35** | 28.36 | 26.84 | 25.84 | 30.31 | **30.46** | 30.35 | **30.48** | 30.13 |
| 25 | 21.84 | 27.00 | **28.42** | 26.85 | 25.81 | 28.90 | **29.31** | 28.93 | **29.31** | 28.99 |
| 30 | 19.96 | 23.42 | **28.21** | 26.84 | 25.77 | 28.20 | **28.52** | 28.22 | **28.52** | 28.06 |
| 35 | 18.49 | 21.05 | 26.20 | **26.80** | 25.71 | 27.26 | **27.78** | 27.28 | **27.78** | 27.27 |
| 40 | 17.29 | 19.34 | 23.26 | **26.63** | 25.64 | 26.65 | **27.21** | 26.68 | **27.22** | 26.58 |
| 45 | 16.27 | 18.01 | 20.97 | **25.57** | 25.53 | 25.95 | **26.69** | 25.98 | **26.70** | 25.95 |
| 50 | 15.40 | 16.91 | 19.21 | 23.40 | **25.35** | 25.40 | **26.26** | 25.44 | **26.27** | 25.37 |

one of the values $\widehat{\sigma} = 10, 20, 30, 40,$ and 50. When testing, we use a noise level of $\sigma \in [10, 50]$.

The results of this experiment are shown in Table 4.1 and Figure 4.9. Table 4.1 shows the comparison with REDNet as initial denoisers, whereas Figure 4.9 shows a visual comparison of an image in the BSD500 dataset. To illustrate the behavior of

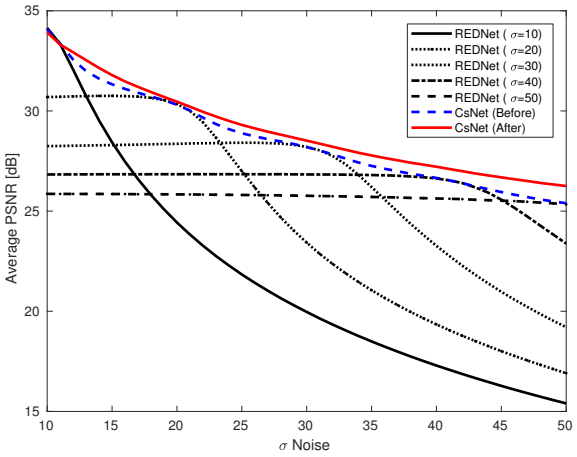



Fig. 4.10.: Noise-level mismatch. Graphical illustration of Table 4.1. The red curve indicates the performance before boosting.

CsNet of all the noise levels, we show in Figure 4.10 a PSNR plot as a function of $\sigma$. This is a comparison between individual REDNets and the CsNet before the boosting step. There are two observations in this experiment.

First, for each $\sigma$, the best performing REDNet is the one with $\widehat{\sigma}$ right above $\sigma$. This result is consistent with the suggestion made in [16]. However, CsNet is able to boost the performance by an average of 0.45dB for noise levels that are originally not trained for, i.e., $\sigma = 15, 25, 35, 45$.

Second, compared to blind REDNet, we observe that CsNet generally has a similar performance before boosting, and better after boosting. This suggests that instead of training a blind denoiser, one can train a set of weak denoisers and use CsNet to combine the results. The advantage of doing so, besides reducing the training cost, is that CsNet allows us to plug-in any off-the-shelf image denoiser as the initial estimators whereas blind denoiser is a fixed network.

### 4.5.2 Experiment 2: Different Image Classes

The objective of this experiment is to evaluate the performance of CsNet when the initial denoisers are trained for different image classes. To this end, we fix the type of initial denoisers as REDNet, and train three different REDNets using three classes

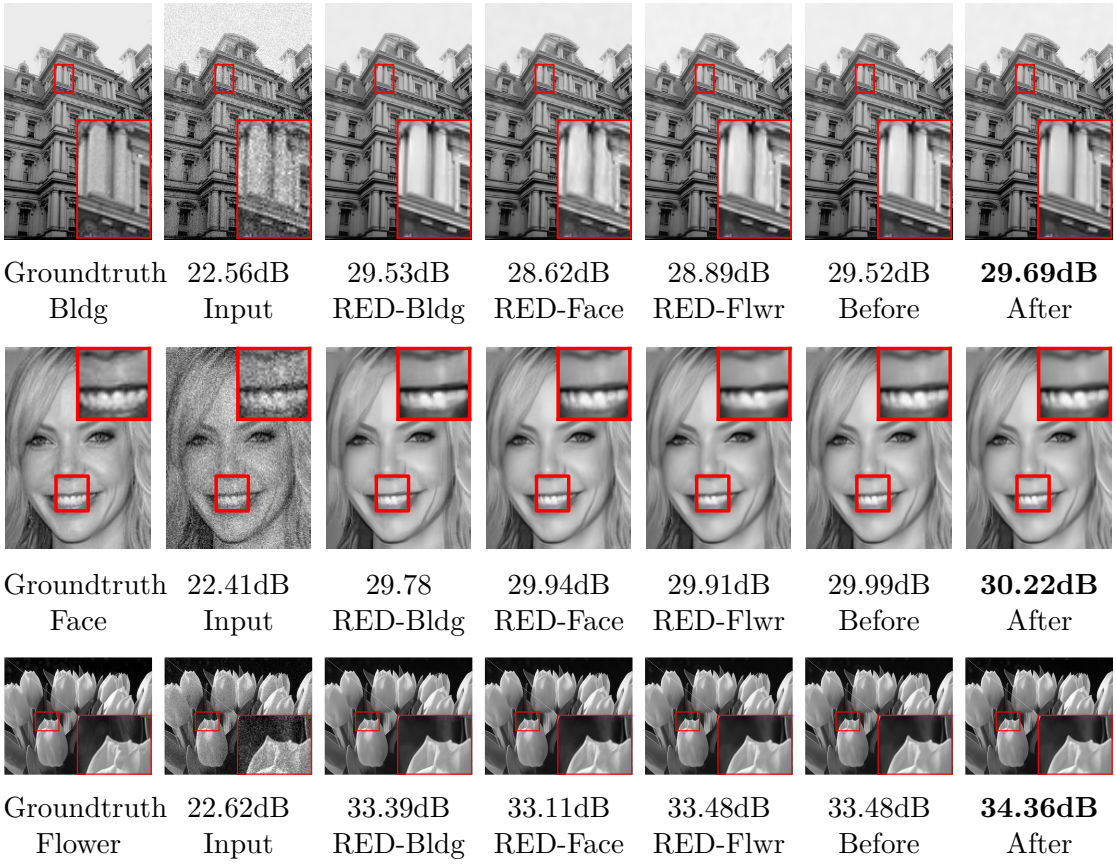



Fig. 4.11.: Image denoising for `Building`, `Face` and `Flower` classes. While class-specific REDNet has good performance when classes match, CsNet is able to select the best denoiser. Testing images are from ImageNet.

Table 4.2.: Example of different image classes. Class-specific REDNets have better performance than BM3D, DnCNN (generic) and REDNet (generic). CsNet selects the best class. We use 10 images from ImageNet for testing.

| | RED (Bldg) | RED (Face) | RED (Flwr) | Before (est) | After (est) | Before (ora) | After (ora) | BM3D (Gen) | DnCNN (Gen) | RED (Gen) |
|---|---|---|---|---|---|---|---|---|---|---|
| Bldg | **30.40** | 28.96 | 29.35 | 30.33 | **30.41** | 30.40 | **30.47** | 29.30 | 29.77 | 29.77 |
| Face | 30.19 | **30.39** | 30.34 | 30.45 | **30.74** | 30.51 | **30.80** | 29.97 | 30.28 | 30.29 |
| Flwr | 31.09 | 30.95 | **31.31** | 31.32 | **31.50** | 31.38 | **31.54** | 30.42 | 31.15 | 31.18 |

of images: `Flower`, `Face` and `Building`. We have experimented with other initial denoisers such as DnCNN, but the results are similar. In training the initial denoisers, we manually select 200 class-specific images for each class from the ImageNet [130].

We fix the noise level as $\sigma = 20$ to eliminate the complication of having uncertainty in both noise levels and image classes.

The result of this experiment is shown in Table 4.2 with a few representative examples in Figure 4.11. We observe that denoisers trained with generic database such as DnCNN and REDNet perform worse than class-specific denoisers. For example, in the `Building` image, DnCNN (generic) and REDNet (generic) attain 29.7722dB and 29.7743dB respectively. A REDNet trained with `Building` class has a PSNR of 30.39dB, approximately 0.7dB above the generic REDNet. For `Face` and `Flower` classes, the same observation can be found, although the gap is less substantial. One reason is that for `Building` class, the vertical and horizontal features learned by the network are less common in generic images.

### 4.5.3 Experiment 3: Different Denoiser Types

The objective of this experiment is to evaluate CsNet for different types of initial denoisers. To this end, we consider four denoisers running at specific noise levels $\widehat{\sigma}$ that match with the actual noise level $\sigma$. These denoisers are BM3D [20], DnCNN [16], REDNet [14] and FFDNet [18]. We use the original implementation by the authors for DnCNN and FFDNet, and build our own REDNet.

The results of this experiment are shown in Table 4.3. Among the four denoisers, FFDNet and REDNet have comparable performance at the top, followed by DnCNN and then BM3D. For the five noise levels we tested, CsNet consistently improves the performance. The PSNR gain with respect to the best denoiser is less significant for small $\sigma$, but becomes more substantial for large $\sigma$. One reason is that for high noise the initial denoisers tend to oversmooth. The boosting of the CsNet is thus effective. Figure 4.12 shows a visual comparison on the `Bear` image. In this image, BM3D actually performs better than DnCNN. The proposed CsNet can pick this best estimate (24.79dB), and boost the PSNR to 25.76dB.

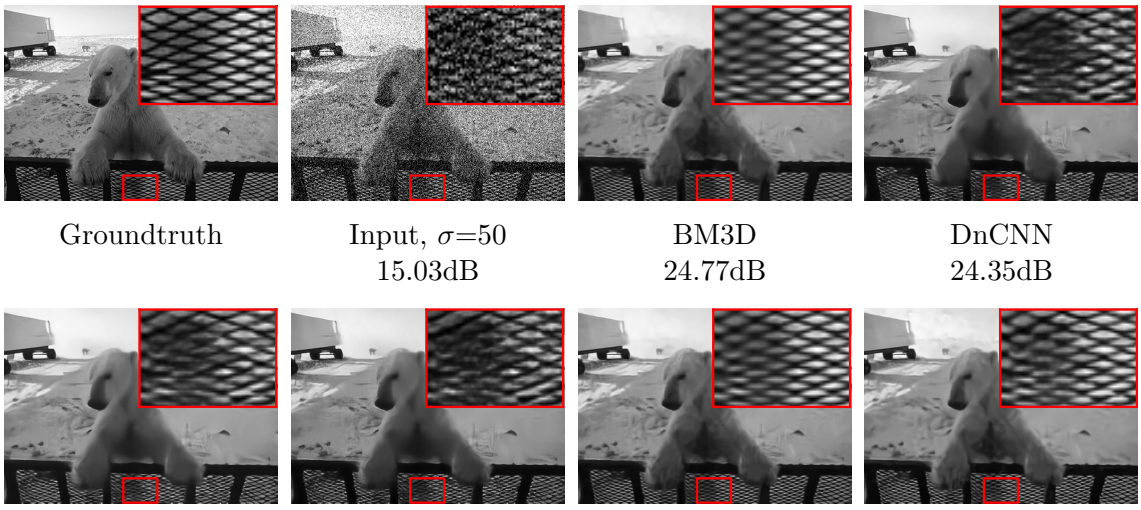



Fig. 4.12.: Example of different denoiser type. The ConsensusNet is used to integrate BM3D [20], DnCNN [16], REDNet [14], and FFDNet [18]. The testing image is `Bear` (size 321×481) from BSD500.

Table 4.3.: Example of different denoiser type. We integrate BM3D [20], DnCNN [16], REDNet [14], and FFDNet [18], and show CsNet before and after boosting. We use 200 images from BSD500 for testing.

| $\sigma$ | BM3D [20] | DnCNN [16] | FFDNet [18] | REDNet [14] | Before Boost (est) | After Boost (est) | Before Boost (ora) | After Boost (ora) |
|---|---|---|---|---|---|---|---|---|
| 10 | 33.56 | 34.10 | 33.94 | **34.12** | **34.14** | 33.89 | **34.16** | 33.90 |
| 20 | 29.73 | 30.33 | 30.27 | **30.34** | 30.37 | **30.48** | 30.40 | **30.51** |
| 30 | 27.68 | 28.17 | 28.18 | **28.20** | 28.23 | **28.52** | 28.28 | **28.55** |
| 40 | 26.22 | 26.60 | **26.65** | 26.62 | 26.68 | **27.19** | 26.72 | **27.21** |
| 50 | 24.99 | 25.34 | **25.35** | 25.35 | 25.40 | **26.16** | 25.44 | **26.18** |

### 4.5.4 Limitations and Extensions

The effectiveness of CsNet is mainly dominated by the accuracy of the MSE estimate. If the noise is truly i.i.d. Gaussian (i.e., unclipped) and the noise level is low, then a deterministic MSE estimator such as SURE would be the ideal candidate. When noise level increases and when noise becomes clipped, then a neural network

based MSE estimator is a better option. If the images are large and complex, we can partition the image into sub-regions and use CsNet to handle each region separately. The bottleneck, again, is the accuracy in estimating the MSE. In the presence of MSE uncertainty, one solution is to consider regularization to $(P_1)$. Possible choices of regularization include forcing similar weights for denoisers that are known to perform similarly. We leave the discussion of such regularization to future work.

When training the neural networks we choose to use the $L_1$ metric, for it gives slightly better visual quality then the usual $L_2$ metric. We do not heavily tune this metric because it is not the focus of the chapter. For readers who are concerned about the loss function, we refer to [131] for some recent empirical findings on the topic.

The advantage of CsNet relative to other class-aware neural network denoisers is that we allow combination of multiple denoisers. Typical class-aware denoisers, e.g., [15], rely on semantic classifiers to greedily select only one denoiser. As we demonstrated in Section 4.5.2, a combination of the denoisers is better than the best of the individuals.

CsNet is a general framework for combining estimators. That is, one is not limited to applying CsNet to image denoising problems, although we use denoising as a demonstration. A straight forward extension of CsNet is to combine multiple deblurring algorithms, or to combine multiple image super-resolution algorithms. In complex imaging scenarios where no single method performs uniformly better than the others, CsNet offers a solution to integrate individual weak estimators.

## 4.6 Conclusion

We present an optimal framework called the Consensus Neural Network (CsNet) to combine multiple weak image denoisers. CsNet consists of three major components. Starting with a set of initial image denoisers, CsNet first uses a novel deep neural network to estimate the MSE. The deep neural network is more robust than the traditional estimators such as SURE for estimating the MSE. Once the MSE

is estimated, CsNet solves a convex optimization problem. The optimality of the CsNet is guaranteed by the convex formulation. Finally, the combined estimate is boosted using a new deep neural network image booster. Experimental results confirm the effectiveness of CsNet, where it shows superior performance compared to other state-of-the-art denoising algorithms on tasks including: overcoming noise level mismatch, combining denoisers for different image classes, and combining different denoiser types.

# 5. IMAGE RECONSTRUCTION FOR QUANTA IMAGE SENSORS USING DEEP NEURAL NETWORKS

In this chapter, we propose deep neural networks for reconstructing images for Quanta Image Sensors. This work is based on a paper to appear in International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2018 [132].

## 5.1 Introduction

Quanta Image Sensor (QIS) is a new type of image sensor envisioned to supersede CMOS and CCD [133]. Having a very small full-well capacity ($1-250$ photoelectrons) and single-photon sensitivity, QIS is perceived as an ideal candidate for compensating the deterioration of signal-to-noise ratio in small pixels. The sensor has an extremely high readout rate (10k fps as in [134], and 156k fps in [135]), and can potentially be made for very high spatial resolution [133, 136]. However, the QIS data is binary: A pixel has a value 1 if the photon count exceeds certain threshold, and has a value 0 if the photon count is below the threshold. As a result, non-traditional image reconstruction algorithms are need to recover the images, as illustrated in Figure 5.1.

Existing image reconstruction methods for QIS are largely based on maximum-likelihood (ML) or maximum a-posteriori (MAP) estimation. These optimizations are done using gradient descent [137], dynamic programming [138] or ADMM [139], which are all time consuming. A significantly faster algorithm is the Transform-Denoise method by Chan et al. [24], where the authors use the variance stabilizing transform (VST) to convert the truncated Poisson random variables to Gaussian, and then apply denoising algorithms for smoothing. In this chapter, we propose a deep neural network approach for QIS image reconstruction. As shown in Figure 5.2,

the neural network has better performance than Transform-Denoise by a substantial margin.

Using deep neural networks for image restoration problems is relatively new but has a strong momentum [140–146]. In [25], the authors proposed a neural network to unroll the ISTA iteration with a sparsity prior. However, sparsity priors are generally inferior to discriminative priors learned directly by the neural networks [142]. A simple QIS reconstruction network is proposed by Rojas et al. [26], where they presented a two-layer neural network to learn the Transform-Denoise pipeline in [24]. However, despite the speed-up offered by the network, the PSNR performance is worse than Transform-Denoise using BM3D as the denoiser.

The key contribution of this chapter is a new deep neural network based solution for QIS image reconstruction. Different from [25] which assumes a sparsity prior, our network learns the denoiser directly; And compared to [26], our network has a significantly deeper layer to learn the transformation. We present two designs: one mimics the entire Transform-Denoise pipeline, and the other one substitutes part of the Transform-Denoise pipeline. We show that both networks has significantly better performance than the existing Transform-Denoise method.

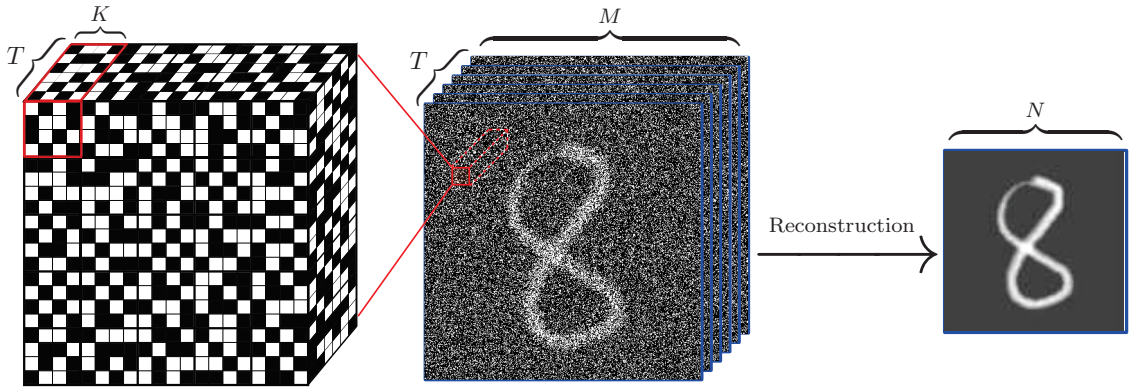
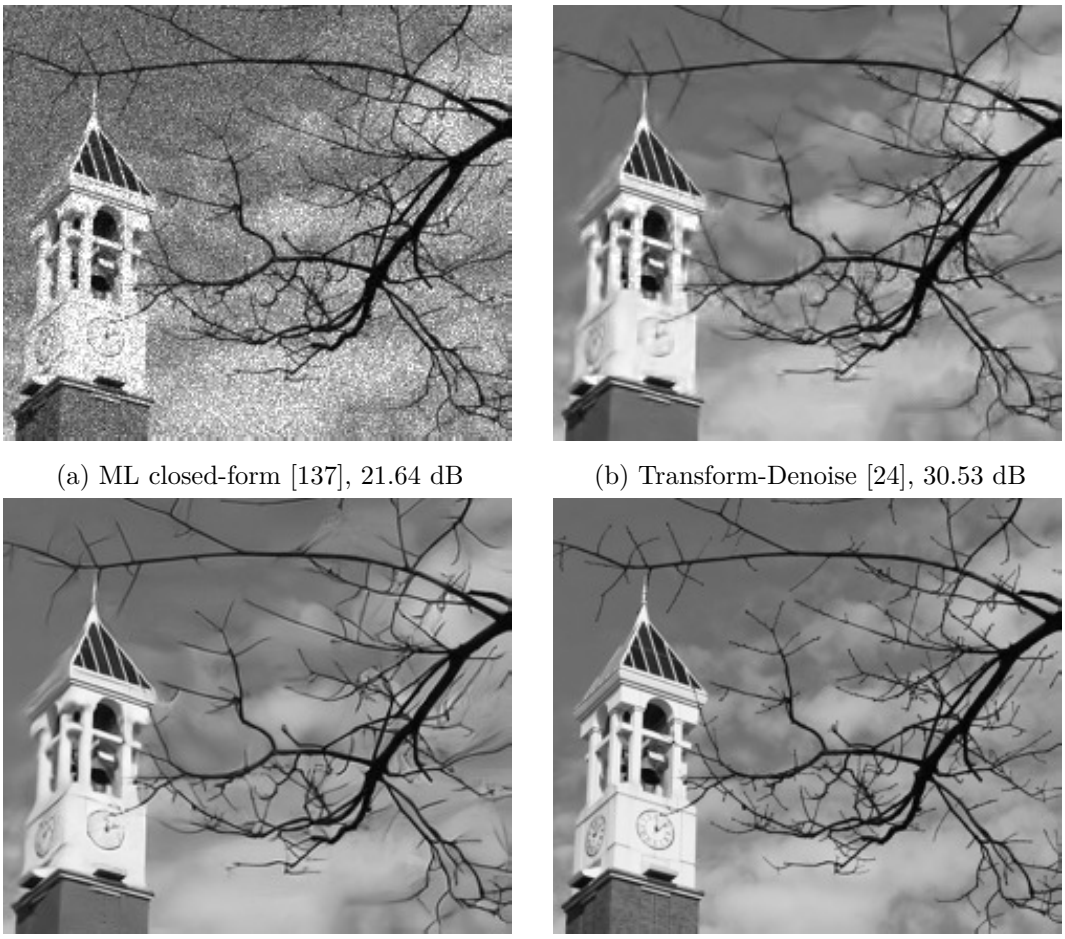



Fig. 5.1.: Image reconstruction of QIS. Given the binary bit planes, the algorithm estimates the gray-scale image shown on the right.



(a) ML closed-form [137], 21.64 dB

(b) Transform-Denoise [24], 30.53 dB

(c) Proposed Method, 31.45 dB

(d) Ground Truth

Fig. 5.2.: Image Reconstruction using ML [137], TD [24], and our proposed RED-Net method.

## 5.2 QIS Imaging Model

In this section we provide an overview of the QIS imaging model. A pictorial illustration is shown in Figure 5.3. We shall focus on a few important highlights of the model. Readers interested in the details can refer to [137], [24] or [147].

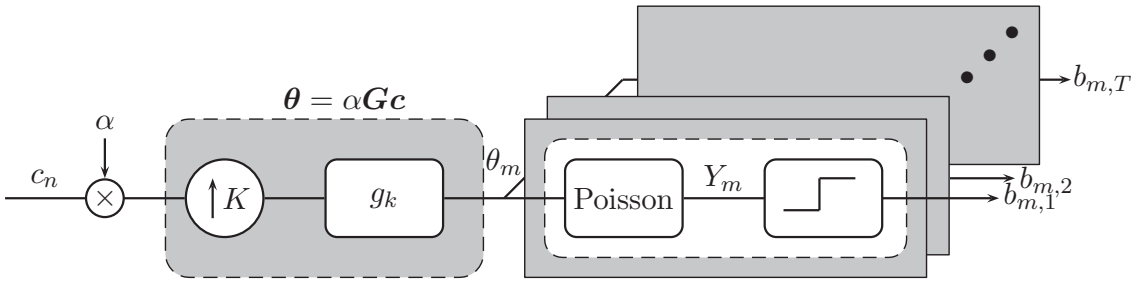



Fig. 5.3.: Image formation process of QIS.

### 5.2.1 Spatial-Temporal Oversampling

We model the incoming light intensity as a vector $\boldsymbol{c} = [c_0, \ldots, c_{N-1}]^T$. We assume that $c_n$ is normalized to the range $[0, 1]$ for all $n$, and use a constant $\alpha > 0$ to model the gain factor.

QIS uses $M \gg N$ jots to *oversample* $\boldsymbol{c}$. The ratio $K \stackrel{\text{def}}{=} M/N$ is the spatial oversampling factor. The oversampling process is modeled by an up-sampling operator and a lowpass filter $\{g_k\}$ as shown in Figure 5.3. Mathematically, we define the output of the oversampling process as

$$\boldsymbol{\theta} = \alpha \boldsymbol{G}\boldsymbol{c}, \tag{5.1}$$

where $\boldsymbol{\theta} = [\theta_0, \ldots, \theta_{M-1}]^T$ denotes the light intensity sampled at the $M$ jots, and the matrix $\boldsymbol{G} \in \mathbb{R}^{M \times N}$ is a matrix capturing the upsampling and the lowpass filter $\{g_k\}$.

The lowpass filter $\{g_k\}$ can be arbitrary, e.g., B-spline as mentioned in [137]. However, for efficient reconstruction we shall assume that the filter is box-car. Physically, by using a box-car filter we implicitly assume that the incident light is focused on each jot, which is reasonable to some extent because QIS is equipped with micro-lenses to focus incident light. If $\{g_k\}$ deviates from the box-car but we still use box-car for reconstruction, we say that there is model mismatch, which will be studied in Section 5.4.

### 5.2.2 Truncated Poisson Process

The oversampled signal $\boldsymbol{\theta}$ generates a sequence of Poisson random variables according to the distribution

$$\mathbf{P}(Y_{m,t} = y_{m,t}) = \frac{\theta_m^{y_{m,t}} e^{-\theta_m}}{y_{m,t}!}, \tag{5.2}$$

where $m \in \{0, 1, \ldots, M-1\}$ and $t \in \{0, 1, \ldots, T-1\}$ denote the $m$-th jot and the $t$-th independent measurement in time, respectively. Denoting $q \in \mathbb{N}$ as the quantization threshold, the final observed binary measurement $B_{m,t}$ is a truncation of $Y_{m,t}$, i.e., $B_{m,t} = 1$ when $Y_{m,t} \geq q$, and $B_{m,t} = 0$ otherwise. Hence, the distribution of $B_{m,t}$ is

$$\mathbf{P}(B_{m,t} = b_{m,t}) = \begin{cases} \Psi_q(\theta_m), & \text{if } b_{m,t} = 0, \\ 1 - \Psi_q(\theta_m), & \text{if } b_{m,t} = 1. \end{cases} \tag{5.3}$$

where $\Psi_q : \mathbb{R}^+ \to [0, 1]$ is the upper incomplete Gamma function [148].

The goal of image reconstruction is to reconstruct the underlying image $\boldsymbol{c}$ from the binary measurements $\mathcal{B} = \{B_{m,t} \mid m = 0, \ldots, M-1, \text{and } t = 0, \ldots, T-1\}$ as shown in Figure 5.1. With the box-car kernel assumption, one can show that the ML solution has a closed-form [24]:

$$\hat{c}_n = \frac{K}{\alpha} \Psi_q^{-1} \left( 1 - \frac{S_n}{L} \right), \tag{5.4}$$

where $S_n \stackrel{\text{def}}{=} \sum_{t=0}^{T-1} \sum_{k=0}^{K-1} B_{Kn+k,t}$ is the spatial-temporal binning of the binary measurements, and $L \stackrel{\text{def}}{=} KT$ is the combined spatial-temporal oversampling factor.

### 5.2.3 Transform-Denoise Approach

Our proposed deep neural network shares some similarity with the Transform-Denoise in [24]. In Transform-Denoise, the key observation is that the random variable $S_n$ in (5.4) is binomial. The binomial random variable in QIS has spatially varying

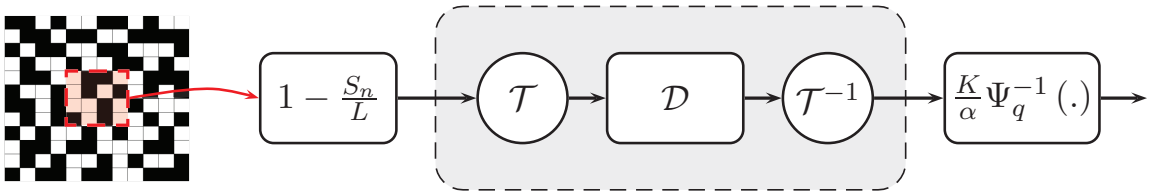



Fig. 5.4.: Transform-Densoise [24]: We apply a pair of transforms $(\mathcal{T}, \mathcal{T}^{-1})$ and a Gaussian denoiser $\mathcal{D}$ for QIS image reconstruction.

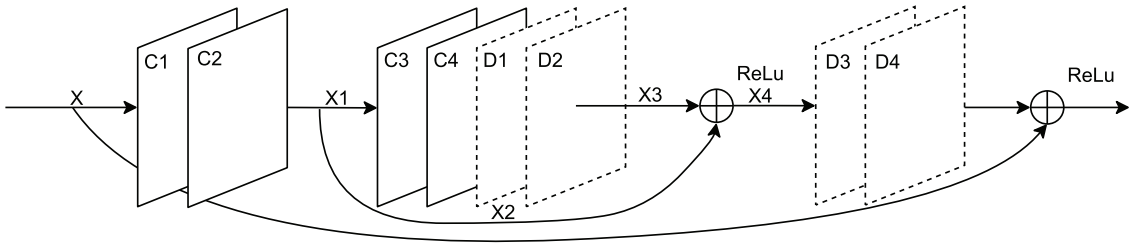



Fig. 5.5.: The proposed QISNet consists of 15 convolutional layers followed by 15 deconvolutional layers.

variance. Thus, one needs to stabilize its variance using variance stabilizing transform (VST). The VST used in Transform-Denoise is the Anscombe binomial transform [149]:

$$Z_n = \mathcal{T}(S_n) \overset{\text{def}}{=} \sqrt{L + \frac{1}{2}} \sin^{-1}\left(\sqrt{\frac{S_n + \frac{3}{8}}{L + \frac{3}{4}}}\right). \tag{5.5}$$

After VST, standard Gaussian denoisers can be used to smooth the image. The final result is obtained by an inverse VST. The overall Transform-Denoise pipeline is shown in Figure 5.4.

## 5.3 Proposed Method

### 5.3.1 Network Structure

The structure of our proposed neural network is shown in Figure 5.5. We call our network the QISNet. On the network level, QISNet has the same structure as the very deep Residual Encoder-Decoder Network "RED-Net" architecture [150], which

was originally proposed for denoising. In this network structure, there is a sequence of $N$ convolutional layers and $N$ deconvolutional layers. The convolutional layers extract the features from the input image, and the deconvolutional layers recover the details lost during the convolutional steps. As mentioned in [151], the deconvolutional layers are necessary for image restoration tasks because the convolutional layers tend to oversmooth the image.

What makes QISNet different from RED-Net is that RED-Net cannot be directly applied to the QIS image reconstruction problem as RED-Net is designed for i.i.d. Gaussian noise. The QIS data, as discussed, is binary following from the truncated Poisson distribution. Therefore, in order to apply the network to QIS, modifications are needed.

Our modification is based on the Transform-Denoise pipeline. The insight is that while individual bits of the QIS data follow a truncated Poisson distribution, the average of the bits within a small spatial-temporal block $1 - \frac{S_n}{L}$ is a Binomial random variable. If we further assume that the blocks do not overlap, then $1 - \frac{S_n}{L}$ can be regarded as an noisy pixel where the distribution is independent (but not identical) Binomial. As a result, if we feed $1 - \frac{S_n}{L}$ into the network, then a denoising network will be sufficient.

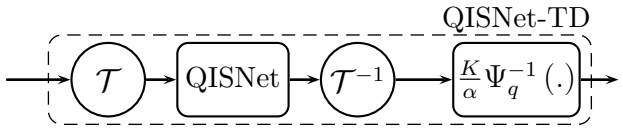
### 5.3.2 Two Designs for QISNet

Knowing that the input data to the QIS image reconstruction is independent Binomial, we can now design different combinations of the networks for the reconstruction task. Here we present two designs.

The first design is to use the neural network to replace the Gaussian denoiser in Transform-Denoise. We call this design QISNet-TD (See Figure 5.5). The idea of QISNet-TD is that since the performance of Transform-Denoise depends heavily on the denoiser, we should use a good denoiser. However, we cannot simply put a pre-trained Gaussian noise network denoiser for this task because the pipeline involves

other components. We train the network while forcing it to learn the presence of $\mathcal{T}$, $\mathcal{T}^{-1}$ and $\frac{K}{\alpha}\Psi_q^{-1}(\cdot)$.

The second design is to use the QISNet to replace the entire Transform-Denoise pipeline (See Figure 5.6). This design is slightly more aggressive as we ask the neural network to learn the denoiser, the nonlinear functions $\mathcal{T}$ and $\mathcal{T}^{-1}$, and $\frac{K}{\alpha}\Psi_q^{-1}(\cdot)$. The difference between QISNet-TD and QISNet is the transforms $\mathcal{T}$ and $\mathcal{T}^{-1}$ (and the nonlinear function $\frac{K}{\alpha}\Psi_q^{-1}(\cdot)$ which is less important here). The inverse transform $\mathcal{T}^{-1}$ is the algebraic inverse, which is a biased inverse transformation. As $L$ grows, the bias of $\mathcal{T}$ will cause the estimate to deviate from its ideal value. Therefore, as one may expect, QISNet-TD performs worse than QISNet in general. We will demonstrate this behavior in the experiment section.



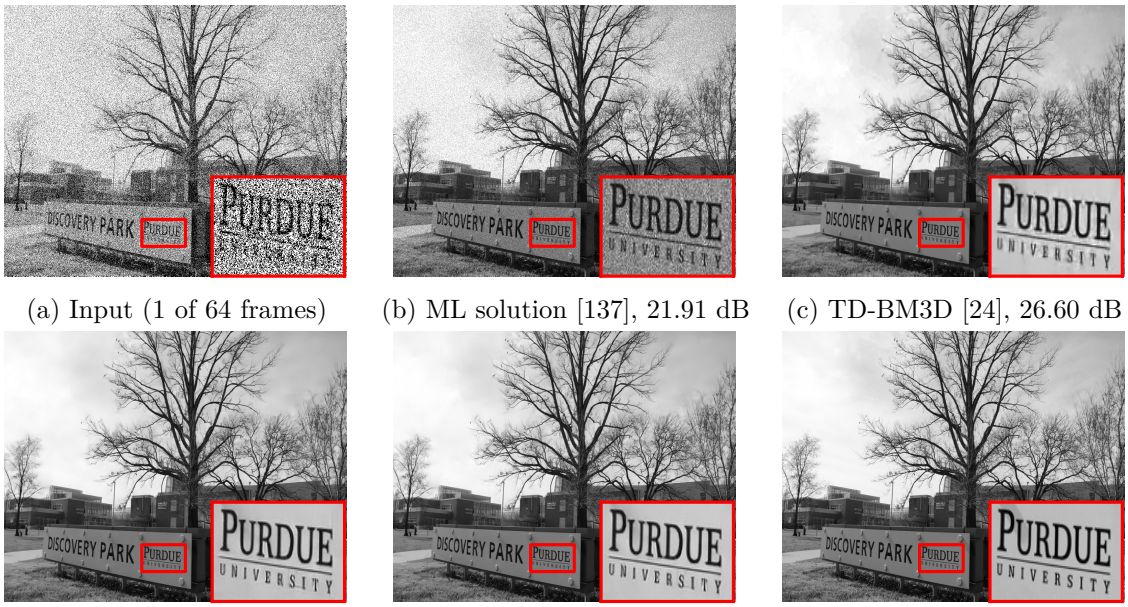

(a) QISNet-TD: Embeds QISNet into the TD framework.


(b) QISNet: Use QISNet to learn the entire framework.

Fig. 5.6.: The two proposed designs.

### 5.3.3 Training and Parameters

We implement both QISNet-TD and QISNet using 15 convolutional and 15 deconvolutional layers. Each layer uses $3 \times 3$ kernels, and 64 feature maps. The network nonlinearity is obtained using ReLu. The training dataset consists of 2000 images selected from the Pascal VOC 2008 dataset [152]. 128 patches of size $50 \times 50$ are randomly extracted from each image. The inputs used to train the networks are $1 - \frac{S_n}{L}$, which are images with Binomial "'noise". The ground truths are the clean images. The loss function is $L_2$-loss, which is optimized using Adam optimizer with a learning rate of 0.0001. The training converges to a local minimum [150] and it takes 8 hours




(a) Input (1 of 64 frames) (b) ML solution [137], 21.91 dB (c) TD-BM3D [24], 26.60 dB

(d) QISNet-TD, 27.35 dB (e) QISNet, 27.36 dB (f) Ground Truth

Fig. 5.7.: Reconstructed Images and their PSNR for $L = 64$.

using NVidia Geforce GTX TITAN GPU. For parameters, we set $q = 1$, $\alpha = 2K^2$, and $T = 16$.

## 5.4 Experiments

We synthesize QIS data from 77 images captured using a Canon EOS Rebel T6i camera. The images are captured on Purdue campus, which are guaranteed to be different from the Pascal VOC 2008 dataset used for training.

### 5.4.1 Reconstruction Quality

We compare the proposed networks with the Transform-Denoise using BM3D [24] and the classical MLE approach [137]. We study two cases: $K = 1$ and $K = 2$. Since $T = 16$, these correspond to $L = K^2T = 16$ and $L = 64$, respectively.

The results of the experiments are shown in Table 5.1. In this table, we divide the study into two parts. The first part is the "Match" experiment, where during

the QIS data synthesis we assume that the lowpass filter $g_k$ is box-car. It is called "Match" because the variable $S_n$ also assumes a box-car filter.

We observe that while TD-BM3D [24] offers almost 10dB improvement over MLE [137], the proposed networks give additional improvements. QISNet performs as good as than QISNet-TD for small $L$ (27.41dB). For large $L$, QISNet is better (30.62dB with 30.51dB). This suggests that QISNet is indeed able to learn the transforms $(\mathcal{T}, \mathcal{T}^{-1})$ with sufficient amount of data. Visually, the results in Figure 5.7 show that the neural networks reconstruct more details.

### 5.4.2 Model Mismatch in G

The second part of the experiment is the "Mismatch" case. Here, by mismatch we meant that the box-car filter used in calculating $S_n$ does not match with the lowpass filter used for generating the QIS data. Note that if the lowpass filter $g_k$ is not box-car, one has to use an iterative algorithm such as gradient descent [137] or ADMM [139] to do the reconstruction. Iterative algorithms are not preferred as they are practically slow. Thus it is important to see how well the neural networks can tolerate the model mismatch.

Table 5.1.: PSNR in dB for $L = 16$ and $L = 64$

| | Method | Mismatch | | | Match |
|---|---|---|---|---|---|
| | | Linear | Quad | Cubic | Box-Car |
| $L = 16$ | MLE | 15.74 | 15.69 | 15.64 | 15.84 |
| | TD-BM3D | 25.67 | 25.44 | 25.23 | 26.40 |
| | QISNet-TD | 26.38 | 26.04 | 25.74 | **27.41** |
| | QISNet | **26.39** | **26.05** | **25.76** | 27.40 |
| $L = 64$ | MLE | 19.94 | 19.93 | 19.92 | 21.12 |
| | TD-BM3D | 25.45 | 25.40 | 25.33 | 29.90 |
| | QISNet-TD | 25.51 | 25.47 | 25.39 | 30.51 |
| | QISNet | **25.57** | **25.52** | **25.45** | **30.62** |

The results of this part of the experiment are shown in Table 5.1. Our proposed QISNet-TD and QISNet are trained assuming box-car functions. As we can see

from the table, as the mismatch becomes worse (from linear to cubic splines), the reconstruction PSNR also drops. However, the PSNR drop in the neural network approaches are not worse than Transform-Denoise. In fact, for all the mismatch filters, the networks still produce better reconstruction quality. One thing to note, however, is that if we know the lowpass filter, we can easily re-train the network to adapt to the filter. Transform-Denoise does not have this flexibility.

## 5.5 Conclusion

We proposed deep neural networks for reconstructing images for Quanta Image Sensors. Our networks can replace the existing Transform-Denoise pipeline, while offering better image reconstruction results. Practically, we anticipate that the networks can eventually be put on neuromorphic chips for better speed and performance.

# 6. CONCLUSION

## 6.1 Summary

### Matrix and Tensor Factorization

In Chapter 2, we presented a computationally-attractive and memory-efficient tensor factorization algorithm by addressing the exploiting problem of the Khatri-Rao product. Also, we discussed a strategy for distributing the computations across multiple machines. In Chapter 3, we studied four variants of ADMM algorithm for nonnegative matrix factorization in the presense of noise by variable split strategy. We found from the experiments that the multiple-variable split method performs worse than the single-variable split method, the half quadratic penalty and the algorithm-induced priors.

### Deep Neural Network for Image Denoising

In Chapter 4, we presented an optimal framework to combine multiple, weak image denoisers. The framework consists of three steps: Estimate the MSE using deep neural network; optimally combine the images from multiple denoisers via convex formulation; boost the combined image. The experimental results support our framework as they show superior performance compared to other state-of-the-art denoising algorithms. In Chapter 5, we proposed applying deep neural networks to Quanta Image Sensors for image reconstruction. Our deep neural network replaces the existing Transform-Denoise pipeline and offers better image reconstruction results.

## 6.2 Future Work

There are notable limitations to both the tensor factorization and the deep neural network models in this thesis. First, our distributed tensor factorization model suffered a substantial increase in communication time on more than 8 nodes. To reduce the communication time, an algorithm should avoid synchronization which forces computationally completed nodes to wait until all other nodes are finished. To solve this problem, we will further pursue an asynchronous tensor factorization method using a queue per each node. We expect this method will work well in terms of computational efficiency and accuracy.

As for the deep neural network model, we used noisy images clipped to values between 0 and 1 as input for our Consensus Neural Network (CsNet). To imitate reality, we should use unclipped noisy images that include negative values. Therefore, our next set of experiments includes unclipped noisy images and an updated deep neural network model that is capable of handling negative values.

# APPENDICES

# A. APPENDIX OF DFACTO

## A.1 Definitions of Standard Matrix Products

**Definition A.1.1** *The Kronecker product* $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{mp \times nq}$ *of matrices* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *and* $\mathbf{B} \in \mathbb{R}^{p \times q}$ *is defined as*

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{1,1}\mathbf{B} & a_{1,2}\mathbf{B} & \dots & a_{1,n}\mathbf{B} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m,1}\mathbf{B} & a_{m,2}\mathbf{B} & \dots & a_{m,n}\mathbf{B} \end{bmatrix}. \tag{A.1}$$

**Definition A.1.2** *The Khatri-Rao product* $\mathbf{A} \odot \mathbf{B} \in \mathbb{R}^{mp \times n}$ *of matrices* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *and* $\mathbf{B} \in \mathbb{R}^{p \times n}$ *is given by the Kronecker product of the corresponding columns of the two matrices:*

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_{:,1} \otimes \mathbf{b}_{:,1} & \mathbf{a}_{:,2} \otimes \mathbf{b}_{:,2} & \dots & \mathbf{a}_{:,n} \otimes \mathbf{b}_{:,n} \end{bmatrix}. \tag{A.2}$$

**Definition A.1.3** *The Hadamard product* $\mathbf{A} * \mathbf{B} \in \mathbb{R}^{n \times m}$ *of two conforming matrices* $\mathbf{A} \in \mathbb{R}^{n \times m}$ *and* $\mathbf{B} \in \mathbb{R}^{n \times m}$ *is given by*

$$\mathbf{A} * \mathbf{B} = \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,2}b_{1,2} & \dots & a_{1,m}b_{1,m} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n,1}b_{n,1} & a_{n,2}b_{n,2} & \dots & a_{n,m}b_{n,m} \end{bmatrix} \tag{A.3}$$

**Definition A.1.4** *The outer product* $\mathbf{a} \circ \mathbf{b}$ *of vectors* $\mathbf{a} \in \mathbb{R}^m$ *and* $\mathbf{b} \in \mathbb{R}^n$ *is given by a matrix* $\mathbf{M} \in \mathbb{R}^{m \times n}$ *such that*

$$m_{i,j} = a_i b_j. \tag{A.4}$$

*The definition can be extended to tensors by defining the outer product* $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$ *of three vectors* $\mathbf{a} \in \mathbb{R}^m$, $\mathbf{b} \in \mathbb{R}^n$, *and* $\mathbf{c} \in \mathbb{R}^p$ *as a tensor* $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{m \times n \times p}$ *with*

$$m_{i,j,k} = a_i b_j c_k. \tag{A.5}$$

**Definition A.1.5** *Given a matrix* $\mathbf{A} \in \mathbb{R}^{n \times m}$, *the linear operator* $vec(\mathbf{A})$ *yields a vector* $\mathbf{x} \in \mathbb{R}^{nm}$, *which is obtained by stacking the columns of* $\mathbf{A}$*:*

$$vec(\mathbf{A}) = \mathbf{x} = \begin{bmatrix} \mathbf{a}_{:,1} \\ \mathbf{a}_{:,2} \\ \vdots \\ \mathbf{a}_{:,n} \end{bmatrix}. \tag{A.6}$$

*Observe that*

$$x_{i+(j-1)n} = a_{i,j}. \tag{A.7}$$

*On the other hand, given a vector* $\mathbf{x} \in \mathbb{R}^{nm}$, *the operator* $unvec_{(n,m)}(\mathbf{x})$ *yields a matrix* $\mathbf{A} \in \mathbb{R}^{n \times m}$*:*

$$unvec_{(n,m)}(\mathbf{x}) = \mathbf{A} = \begin{bmatrix} \mathbf{a}_{:,1} & \mathbf{a}_{:,2} & \dots & \mathbf{a}_{:,n} \end{bmatrix}. \tag{A.8}$$

The Kronecker product satisfies the following well known relationship (see *e.g.*, proposition 7.1.9 of [68]):

$$vec(\mathbf{ABC}) = \left(\mathbf{C}^\top \otimes \mathbf{A}\right) vec(\mathbf{B}). \tag{A.9}$$

The Khatri-Rao product satisfies (see *e.g.*, chapter 2 of [36]):

$$\left(\mathbf{A} \odot \mathbf{B}\right)^\top \left(\mathbf{A} \odot \mathbf{B}\right) = \mathbf{A}^\top \mathbf{A} * \mathbf{B}^\top \mathbf{B}. \tag{A.10}$$

Plugging this into the definition of the Moore-Penrose pseudo-inverse [68] immediately shows that

$$(\mathbf{A} \odot \mathbf{B})^{\dagger} = \left(\mathbf{A}^{\top}\mathbf{A} * \mathbf{B}^{\top}\mathbf{B}\right)^{-1} (\mathbf{A} \odot \mathbf{B})^{\top}. \tag{A.11}$$

### A.1.1 An Example of Flattening Tensors

Let $\mathcal{X}$ be a $3 \times 4 \times 3$ tensor with frontal slices

$$\begin{bmatrix} 1 & 1 & 4 & 2 \\ 3 & 4 & 5 & 3 \\ 5 & 0 & 5 & 1 \end{bmatrix} \begin{bmatrix} 4 & 5 & 5 & 1 \\ 1 & 1 & 1 & 4 \\ 1 & 1 & 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 & 4 \\ 4 & 1 & 5 & 1 \\ 5 & 2 & 4 & 1 \end{bmatrix}, \text{ then}$$

$$\mathbf{X}^1 = \left[\begin{array}{cccc|cccc|cccc} 1 & 1 & 4 & 2 & 4 & 5 & 5 & 1 & 1 & 0 & 2 & 4 \\ 3 & 4 & 5 & 3 & 1 & 1 & 1 & 4 & 4 & 1 & 5 & 1 \\ 5 & 0 & 5 & 1 & 1 & 1 & 0 & 3 & 5 & 2 & 4 & 1 \end{array}\right]$$

$$\mathbf{X}^2 = \left[\begin{array}{ccc|ccc|ccc} 1 & 4 & 1 & 3 & 1 & 4 & 5 & 1 & 5 \\ 1 & 5 & 0 & 4 & 1 & 1 & 0 & 1 & 2 \\ 4 & 5 & 2 & 5 & 1 & 5 & 5 & 0 & 4 \\ 2 & 1 & 4 & 3 & 4 & 1 & 1 & 3 & 1 \end{array}\right]$$

$$\mathbf{X}^3 = \left[\begin{array}{ccc|ccc|ccc|ccc} 1 & 3 & 5 & 1 & 4 & 0 & 4 & 5 & 5 & 2 & 3 & 1 \\ 4 & 1 & 1 & 5 & 1 & 1 & 5 & 1 & 0 & 1 & 4 & 3 \\ 1 & 4 & 5 & 0 & 1 & 2 & 2 & 5 & 4 & 4 & 1 & 1 \end{array}\right]$$

## A.2 Review of ALS

In this section, we will introduce the CANDECOMP / PARAFAC (CP) decomposition model, and the ALS algorithm. The CP decomposition is a multi-way tensor factorization model. Given a tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, the $R$-rank CP decomposition of $\mathcal{X}$ is given by three matrices $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, and $\mathbf{C} \in \mathbb{R}^{K \times R}$ such that

$$\mathcal{X} \approx \sum_{r=1}^{R} \lambda_r \cdot \mathbf{a}_{:,r} \circ \mathbf{b}_{:,r} \circ \mathbf{c}_{:,r}. \tag{A.12}$$

Note that the columns of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are normalized to have unit length. The CP decomposition is computed by solving

$$\min_{\hat{\mathcal{X}}} \ \mathcal{X} - \hat{\mathcal{X}} \qquad \text{with} \quad \hat{\mathcal{X}} = \sum_{r=1}^{R} \lambda_r \cdot \mathbf{a}_{:,r} \circ \mathbf{b}_{:,r} \circ \mathbf{c}_{:,r}. \tag{A.13}$$

The most popular method to solve the above problem is the Alternating Least Squares (ALS) algorithm [37]. The basic idea here is to fix all the matrices except one, and solve a least squares problem. Fixing $\mathbf{B}$ and $\mathbf{C}$ and rewriting (A.13), this amounts to setting

$$\hat{\mathbf{A}} \leftarrow \operatorname*{argmin}_{\hat{\mathbf{A}}} \ \mathbf{X}^1 - \hat{\mathbf{A}} \left(\mathbf{C} \odot \mathbf{B}\right)^\top \tag{A.14}$$

The optimal solution of (A.14) can be rewritten using (A.11) as

$$\hat{\mathbf{A}} = \mathbf{X}^1 \left( \left(\mathbf{C} \odot \mathbf{B}\right)^\top \right)^\dagger \tag{A.15}$$

$$= \mathbf{X}^1 \left(\mathbf{C} \odot \mathbf{B}\right) \left(\mathbf{C}^\top \mathbf{C} * \mathbf{B}^\top \mathbf{B}\right)^{-1}. \tag{A.16}$$

We obtain $\mathbf{A}$ by normalizing the columns of $\hat{\mathbf{A}}$. The ALS procedure repeats analogously to find $\hat{\mathbf{B}}$ and $\hat{\mathbf{C}}$ until a stopping criterion is met. The general CP-ALS algorithm is summarized in Algorithm 9.

**Algorithm 9:** CP-ALS algorithm

**Input:** $\mathbf{X}^1$, $\mathbf{X}^2$, $\mathbf{X}^3$
**Initialize:** $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$
**while** *stopping criterion not met* **do**
  $\mathbf{M}_1 \leftarrow \mathbf{X}^1 (\mathbf{C} \odot \mathbf{B})$
  $\mathbf{A} \leftarrow \mathbf{M}_1 \left(\mathbf{C}^\top \mathbf{C} * \mathbf{B}^\top \mathbf{B}\right)^{-1}$
  Normalize columns of $\mathbf{A}$
  $\mathbf{M}_2 \leftarrow \mathbf{X}^2 (\mathbf{A} \odot \mathbf{C})$
  $\mathbf{B} \leftarrow \mathbf{M}_2 \left(\mathbf{A}^\top \mathbf{A} * \mathbf{C}^\top \mathbf{C}\right)^{-1}$
  Normalize columns of $\mathbf{B}$
  $\mathbf{M}_3 \leftarrow \mathbf{X}^3 (\mathbf{B} \odot \mathbf{A})$
  $\mathbf{C} \leftarrow \mathbf{M}_3 \left(\mathbf{B}^\top \mathbf{B} * \mathbf{A}^\top \mathbf{A}\right)^{-1}$
  Normalize columns of $\mathbf{C}$
**end**

In tensor factorization, occasionally the problem of overfitting occurs. Thus, we add regularization terms to the objective function. Accordingly, we obtain the following new objective function:

$$\min_{\hat{\mathcal{X}}} \ \left\| \mathcal{X} - \hat{\mathcal{X}} \right\| + \frac{1}{2} \lambda \left( \|\mathbf{A}\|^2 + \|\mathbf{B}\|^2 + \|\mathbf{C}\|^2 \right) \quad \text{with} \quad \hat{\mathcal{X}} = \sum_{r=1}^{R} \lambda_r \cdot \mathbf{a}_{:,r} \circ \mathbf{b}_{:,r} \circ \mathbf{c}_{:,r}. \tag{A.17}$$

Then, the optimal solution of (A.17) becomes

$$\hat{\mathbf{A}} = \mathbf{X}^1 (\mathbf{C} \odot \mathbf{B}) \left(\mathbf{C}^\top \mathbf{C} * \mathbf{B}^\top \mathbf{B} + \lambda \mathbf{I}\right)^{-1}. \tag{A.18}$$

### A.3 Review of GD

In this section, we will introduce the GD algorithm using CANDECOMP / PARAFAC (CP) decomposition model introduced in Section A.2. This algorithm uses the same objective function as CP-ALS except for normalization. Thus, we solve

$$\min_{\hat{\mathbf{x}}} \sum_{i,j,k} \frac{1}{2} \left(x_{i,j,k} - \hat{x}_{i,j,k}\right)^2 \quad \text{s.t. } \hat{\boldsymbol{\mathcal{X}}} = \sum_{r=1}^{R} \mathbf{a}_{:,r} \circ \mathbf{b}_{:,r} \circ \mathbf{c}_{:,r} \tag{A.19}$$

We can rewrite the equation in (A.19) as

$$f = \frac{1}{2} \left\| \mathbf{X}^1 - \mathbf{A} \left(\mathbf{C} \odot \mathbf{B}\right)^\top \right\|^2. \tag{A.20}$$

Next, the gradient of (A.20) with respect to $\mathbf{A}$ can be presented as

$$\frac{\partial}{\partial \mathbf{A}} f = -\mathbf{X}^1 \left(\mathbf{C} \odot \mathbf{B}\right) + \mathbf{A} \left(\mathbf{C}^\top \mathbf{C} * \mathbf{B}^\top \mathbf{B}\right). \tag{A.21}$$

In GD, the gradient of $f$ will be written as

$$\nabla f = \begin{bmatrix} vec\left(\frac{\partial}{\partial \mathbf{A}} f\right) \\ vec\left(\frac{\partial}{\partial \mathbf{B}} f\right) \\ vec\left(\frac{\partial}{\partial \mathbf{C}} f\right) \end{bmatrix}. \tag{A.22}$$

Then, we can compute the factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ with $\hat{f} = f - \alpha \nabla f$. The general CP-GD algorithm is summarized in Algorithm 10.

We add regularization terms to the objective function to solve the problem of overfitting. The new objective function is now

$$\min_{\hat{\mathbf{x}}} \sum_{i,j,k} \frac{1}{2} \left(x_{i,j,k} - \hat{x}_{i,j,k}\right)^2 \frac{1}{2} \lambda \left(\|\mathbf{A}\|^2 + \|\mathbf{B}\|^2 + \|\mathbf{C}\|^2\right) \quad \text{s.t. } \hat{\boldsymbol{\mathcal{X}}} = \sum_{r=1}^{R} \mathbf{a}_{:,r} \circ \mathbf{b}_{:,r} \circ \mathbf{c}_{:,r} \tag{A.23}$$

**Algorithm 10:** CP-OPT algorithm

**Input:** $\mathbf{X}^1$, $\mathbf{X}^2$, $\mathbf{X}^3$
**Initialize:** $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$
**while** *stopping criterion not met* **do**
  $\mathbf{M}_1 \leftarrow \mathbf{X}^1 (\mathbf{C} \odot \mathbf{B})$
  $\nabla \mathbf{A} \leftarrow -\mathbf{M}_1 + \mathbf{A} \left(\mathbf{C}^\top \mathbf{C} * \mathbf{B}^\top \mathbf{B}\right)$
  $\mathbf{M}_2 \leftarrow \mathbf{X}^2 (\mathbf{A} \odot \mathbf{C})$
  $\nabla \mathbf{B} \leftarrow -\mathbf{M}_2 + \mathbf{B} \left(\mathbf{A}^\top \mathbf{A} * \mathbf{C}^\top \mathbf{C}\right)$
  $\mathbf{M}_3 \leftarrow \mathbf{X}^3 (\mathbf{B} \odot \mathbf{A})$
  $\nabla \mathbf{C} \leftarrow -\mathbf{M}_3 + \mathbf{C} \left(\mathbf{B}^\top \mathbf{B} * \mathbf{A}^\top \mathbf{A}\right)$
  Calculate Step Size $\alpha$
  $\mathbf{A} \leftarrow \mathbf{A} - \alpha \nabla \mathbf{A}$
  $\mathbf{B} \leftarrow \mathbf{B} - \alpha \nabla \mathbf{B}$
  $\mathbf{C} \leftarrow \mathbf{C} - \alpha \nabla \mathbf{C}$
**end**

Then, the gradient of (A.23) with respect to $\mathbf{A}$ becomes

$$\frac{\partial}{\partial \mathbf{A}} f = -\mathbf{X}^1 (\mathbf{C} \odot \mathbf{B}) + \mathbf{A} \left(\mathbf{C}^\top \mathbf{C} * \mathbf{B}^\top \mathbf{B} + \lambda \mathbf{I}\right). \tag{A.24}$$

### A.4 Illustrative Example for tensor factorization

We illustrate the differences between our algorithm for computing $\mathbf{M} := \mathbf{X}^1 (\mathbf{C} \odot \mathbf{B})$ vs the algorithms proposed by [2] and [3] on the following example: Consider $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{2\times 3\times 3}$ and let

$$\mathbf{X}^1 = \left[\begin{array}{ccc|ccc|ccc} 1 & 0 & 6 & 0 & 4 & 7 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 8 & 0 & 5 & 9 \end{array}\right] \quad \text{and} \quad \mathbf{X}^2 = \left[\begin{array}{ccc|ccc} 1 & 0 & 2 & 0 & 3 & 0 \\ 0 & 4 & 0 & 0 & 0 & 5 \\ 6 & 7 & 0 & 0 & 8 & 9 \end{array}\right].$$

Moreover, let

$$\mathbf{B} = \begin{bmatrix} 3 & 1 \\ 1 & 1 \\ 2 & 3 \end{bmatrix} \text{ and } \mathbf{C} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 3 \end{bmatrix}.$$

[2] propose to store the above tensor as

$$\boldsymbol{b}^{\mathbf{x}} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{bmatrix} \text{ and } \mathbf{S}^{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 2 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \\ 0 & 2 & 1 \\ 1 & 2 & 1 \\ 1 & 2 & 2 \end{bmatrix},$$

where $\boldsymbol{b}^{\mathcal{X}}$ denotes the vector of non-zero entries of $\mathcal{X}$, while $\mathbf{S}^{\mathcal{X}}$ denotes the corresponding vector of indices. The algorithm proposed in Sections 3.2.4 and 3.2.7 of [2] first computes

$$\mathbf{m}_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{bmatrix} * \begin{bmatrix} 3 \\ 3 \\ 3 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 1 \\ 1 \\ 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 18 \\ 8 \\ 5 \\ 12 \\ 28 \\ 32 \\ 18 \end{bmatrix}.$$

The above Hadamard product involves three vectors namely $\boldsymbol{b}_{\mathcal{X}}$, a vector formed by repeating entries of $\mathbf{B}_{:,1}$ based on $\mathbf{S}^{\mathcal{X}}_{:,2}$, and a vector formed by repeating entries of $\mathbf{C}_{:,1}$ based on $\mathbf{S}^{\mathcal{X}}_{:,3}$. Similarly, we compute the vector below but by using $\boldsymbol{b}^{\mathcal{X}}$ and repeated entries from $\mathbf{B}_{:,2}$ and $\mathbf{C}_{:,2}$ respectively:

$$\mathbf{m}_2 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 3 \\ 3 \\ 3 \\ 3 \end{bmatrix} * \begin{bmatrix} 2 \\ 3 \\ 1 \\ 1 \\ 3 \\ 2 \\ 1 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \\ 3 \\ 4 \\ 15 \\ 36 \\ 21 \\ 24 \\ 81 \end{bmatrix}.$$

Finally, we use

$$\mathbf{S}^{\mathbf{x}}_{:,1} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

to sum the appropriate entries of $\mathbf{m}_1$ and $\mathbf{m}_2$ to form $\mathbf{M}$:

$$\mathbf{M} = \begin{bmatrix} 3+6+8+12+28 & 2+6+4+36+21 \\ 18+5+32+18 & 3+15+24+81 \end{bmatrix} = \begin{bmatrix} 57 & 69 \\ 73 & 123 \end{bmatrix}.$$

The algorithm uses 2 $\Omega^{\mathbf{x}}$ extra storage and 5 $\Omega^{\mathbf{x}}$ flops to compute one column of $\mathbf{M}$. On the other hand, the algorithm of [3] computes $\mathbf{M}$ as follows:

$$\begin{aligned} \mathbf{N}_1 &= \mathbf{X}^1 * \left( \mathbf{1}_I \odot (\mathbf{c}_{:,0} \otimes \mathbf{1}_J)^\top \right) \\ &= \begin{bmatrix} 1 & 0 & 6 & 0 & 4 & 7 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 8 & 0 & 5 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 6 & 0 & 8 & 14 & 2 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 16 & 0 & 5 & 9 \end{bmatrix}. \end{aligned}$$

Here $\mathbf{1}_n$ denotes a vector of size $n$ with all entries set to one. Similarly, if $bin\left(\mathbf{X}^1\right)$ denotes an indicator matrix for the non-zero entries of $\mathbf{X}^1$, then

$$\begin{aligned}\mathbf{N}_2 &= bin\left(\mathbf{X}^1\right) * \left(\mathbf{1}_I \odot \left(\mathbf{1}_K \otimes \mathbf{b}_{:,0}\right)^\top\right) \\ &= \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 \\ 3 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 \end{bmatrix} \\ &= \begin{bmatrix} 3 & 0 & 2 & 0 & 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 2 & 0 & 1 & 2 \end{bmatrix}.\end{aligned}$$

Finally we compute $\mathbf{N}_3 = \mathbf{N}_1 * \mathbf{N}_2$ via

$$\mathbf{N}_3 = \begin{bmatrix} 3 & 0 & 12 & 0 & 8 & 28 & 6 & 0 & 0 \\ 0 & 0 & 0 & 18 & 0 & 32 & 0 & 5 & 18 \end{bmatrix}$$

to obtain

$$\mathbf{m}_{:,1} = \mathbf{N}_3 \, \mathbf{1}_{JK} = \begin{bmatrix} 57 \\ 73 \end{bmatrix}.$$

To compute the second column of $\mathbf{M}$ we use

$$\begin{aligned}\mathbf{N}_1 &= \mathbf{X}^1 * \left(\mathbf{1}_I \odot \left(\mathbf{c}_{:,1} \otimes \mathbf{1}_J\right)^\top\right) \\ &= \begin{bmatrix} 1 & 0 & 6 & 0 & 4 & 7 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 8 & 0 & 5 & 9 \end{bmatrix} * \begin{bmatrix} 2 & 2 & 2 & 1 & 1 & 1 & 3 & 3 & 3 \\ 2 & 2 & 2 & 1 & 1 & 1 & 3 & 3 & 3 \end{bmatrix} \\ &= \begin{bmatrix} 2 & 0 & 12 & 0 & 4 & 7 & 6 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 8 & 0 & 15 & 27 \end{bmatrix}.\end{aligned}$$

$$\begin{aligned}\mathbf{N}_2 &= bin\left(\mathbf{X}^1\right) * \left(\mathbf{1}_I \odot \left(\mathbf{1}_K \otimes \mathbf{b}_{:,1}\right)^\top\right) \\ &= \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 3 & 1 & 1 & 3 & 1 & 1 & 3 \\ 1 & 1 & 3 & 1 & 1 & 3 & 1 & 1 & 3 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 3 & 0 & 1 & 3 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 3 & 0 & 1 & 3 \end{bmatrix}.\end{aligned}$$

Finally we compute $\mathbf{N}_3 = \mathbf{N}_1 * \mathbf{N}_2$ via

$$\mathbf{N}_3 = \begin{bmatrix} 2 & 0 & 36 & 0 & 4 & 21 & 6 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 24 & 0 & 15 & 81 \end{bmatrix}$$

and then compute

$$\mathbf{m}_{:,1} = \mathbf{N}_3 \, \mathbf{1}_{JK} = \begin{bmatrix} 69 \\ 123 \end{bmatrix}.$$

The algorithm uses $\max\left(J + \ \Omega^{\mathcal{X}} \ , K + \ \Omega^{\mathcal{X}} \right)$ extra storage and $5 \ \Omega^{\mathcal{X}}$ flops to compute one column of $\mathbf{M}$.

In contrast, our algorithm computes $\mathbf{M}$ as follows:

$$\mathbf{m}_{:,0} = unvec_{(2,3)} \left( \left[\begin{array}{ccc} 1 & 0 & 6 \\ 0 & 4 & 7 \\ 2 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 3 & 0 & 8 \\ 0 & 5 & 9 \end{array}\right] \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix} \right)^{\top} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 15 & 18 & 6 \\ 0 & 25 & 23 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 57 \\ 73 \end{bmatrix}$$

$$\mathbf{m}_{:,1} = unvec_{(2,3)} \left( \left[\begin{array}{ccc} 1 & 0 & 6 \\ 0 & 4 & 7 \\ 2 & 0 & 0 \\ \hline 0 & 0 & 0 \\ 3 & 0 & 8 \\ 0 & 5 & 9 \end{array}\right] \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix} \right)^{\top} \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 19 & 25 & 2 \\ 0 & 27 & 32 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 69 \\ 123 \end{bmatrix}.$$

Our algorithm only requires $nnzc(\mathbf{X}^2)$ extra storage space and $2 \ \Omega^{\mathcal{X}}$ flops for computing $\mathbf{M}$.

## A.5 The application of DFacTo - Joint Matrix Completion and Tensor Factorization

Generally, matrix completion is used when predicting how users will rate items based on data of how these users have previously rated other items. Occasionally, however, the accuracy of prediction from matrix completion is poor because matrix completion only uses prior information on the user, item, and rating. Thus, we suggest a joint matrix completion and tensor factorization model. In this model, we add a word count tensor $\mathcal{X}$ with user-item-word dimensions to the previous rating matrix $\mathbf{Y}$. This model is similar to [71]; but instead of sharing just one dimension (item), we introduce a model that shares both the user and item dimensions. Also, while [71] applies joint tensor completion and matrix factorization, we suggest using joint matrix completion and tensor factorization.

Our joint model can be computed by solving

$$\min_{\hat{\mathbf{x}},\hat{\mathbf{Y}}} \sum_{(i,j)\in\Omega^{\mathbf{Y}}} \frac{1}{2}\left(y_{i,j}-\hat{y}_{i,j}\right)^2 + \mu \sum_{i,j,k} \frac{1}{2}\left(x_{i,j,k}-\hat{x}_{i,j,k}\right)^2 + \lambda \frac{1}{2}\left(\|\mathbf{A}\|^2+\|\mathbf{B}\|^2+\|\mathbf{C}\|^2\right) \tag{A.25}$$

$$\text{s.t. } \hat{\mathcal{X}} = \sum_{r=1}^{R} \mathbf{a}_{:,r}\circ\mathbf{b}_{:,r}\circ\mathbf{c}_{:,r}, \hat{\mathbf{Y}} = \sum_{r=1}^{R} \mathbf{a}_{:,r}\circ\mathbf{b}_{:,r}$$

We can rewrite the equation in (A.25) as

$$f = \frac{1}{2}\sum_{j\in\Omega^{\mathbf{Y}}_{i,:}} \left(y_{i,j}-\mathbf{a}_{i,:}\mathbf{b}_{j,:}^{\top}\right)^2 + \frac{1}{2}\mu\sum_{j}\left(x^1_{i,j}-\mathbf{a}_{i,:}\left(\mathbf{C}\odot\mathbf{B}\right)^{\top}_{j,:}\right)^2 + \frac{1}{2}\lambda\mathbf{a}_{i,:}\mathbf{a}_{i,:}^{\top}. \tag{A.26}$$

Next, the gradient of (A.26) with respect to $\mathbf{a}_{i,:}$ can be presented as

$$\frac{\partial}{\partial\mathbf{a}_{i,:}} f = -\left[\mathbf{y}_{i,:}\mathbf{B}+\mu\,\mathbf{x}^1_{i,:}\left(\mathbf{C}\odot\mathbf{B}\right)\right] + \mathbf{a}_{i,:}\left[\sum_{j\in\Omega^{\mathbf{Y}}}\mathbf{b}_{j,:}^{\top}\mathbf{b}_{j,:}+\mu\,\mathbf{C}^{\top}\mathbf{C}*\mathbf{B}^{\top}\mathbf{B}+\lambda\mathbf{I}\right]. \tag{A.27}$$

The two optimization methods we use to solve the minimization problem in this chapter are the Gradient Descent (GD) and the Alternative Least Squares (ALS).

In GD, the gradient of $f$ will be written as

$$\nabla f = \begin{bmatrix} vec\left(\frac{\partial}{\partial \mathbf{A}} f\right) \\ vec\left(\frac{\partial}{\partial \mathbf{B}} f\right) \\ vec\left(\frac{\partial}{\partial \mathbf{C}} f\right) \end{bmatrix}. \tag{A.28}$$

And each $vec(\cdot)$ of (A.28) will be computed by the gradient of $f$ in (A.27) that corresponds to $\mathbf{a}_{j,:}$, $\mathbf{b}_{j,:}$ and $\mathbf{c}_{k,:}$, respectively because

$$vec\left(\frac{\partial}{\partial \mathbf{A}} f\right) = \begin{bmatrix} \left(\frac{\partial}{\partial \mathbf{a}_{1,:}} f\right)^\top \\ \left(\frac{\partial}{\partial \mathbf{a}_{2,:}} f\right)^\top \\ \vdots \\ \left(\frac{\partial}{\partial \mathbf{a}_{I,:}} f\right)^\top \end{bmatrix}.$$

Then, we can compute the factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ with $\hat{f} = f - \alpha \nabla f$.

On the other hand, in ALS, setting (A.27) to zero shows that the optimal solution of (A.26) is given by

$$\hat{\mathbf{a}}_{i,:} = \left[\mathbf{y}_{i,:}\mathbf{B} + \mu\, \mathbf{x}^1_{i,:}\,(\mathbf{C} \odot \mathbf{B})\right] \left[\sum_{j \in \Omega^{\mathbf{Y}}} \mathbf{b}^\top_{j,:}\mathbf{b}_{j,:} + \mu\, \mathbf{C}^\top\mathbf{C} * \mathbf{B}^\top\mathbf{B} + \lambda\mathbf{I}\right]^{-1}.$$

In both cases, we will use DFacTo, which we suggested in Section 2.3, to avoid the intermediate data explosion problem of $\mathbf{X}^1(\mathbf{C} \odot \mathbf{B})$.

### A.5.1 Experimental Evaluation

We evaluate the joint tensor factorization and matrix completion model on a subset of datasets from Table 2.1. Arguably, our experimental evaluation is very preliminary, but promising. The experimental setup is as follows: We split each

dataset into train, test, and validation. We randomly select 60% of review, rating pairs and designate them as training data. We then select 20% of the remaining review, rating pairs, discard the reviews, remove users or items which do not occur in the training data, and use it for validation. A similar procedure is used to generate the test dataset. Cellartracker and RateBeer datasets contain ratings which are not in a 0 to 5 scale. For consistency, we normalize these ratings to be in 0 to 5. Our evaluation metric is the mean square error which is given by $\sum (i,j) \in \Omega^{\mathbf{Y}}(y_{i,j} - \hat{y}_{i,j})$, were $y_{i,j}$ is a test rating and $\hat{y}_{i,j}$ is the rating predicted by our model.

We train our model with $\mu \in \{10^2, 10^1, ..., 10^{-9}, 10^{-10}\}$ and $\lambda \in \{100, 10, 1, 0.1, 0.01\}$, evaluate its performance on the validation set, and pick the best model based on its mean square error. We use this model to predict on the test dataset and report average mean square error. In Tables A.1 and A.2, we show the MSEs from both the matrix completion and our joint model using GD and ALS. For GD, the method of backtracking line search was used.

Table A.1.: Best Test MSE of single matrix completion and joint matrix completion and tensor factorization model after 500 iterations using Gradient Descent.

| Dataset | Matrix Completion | | Joint (MC + TF) | | |
|---|---|---|---|---|---|
| | $\lambda$ | Test MSE | $\mu$ | $\lambda$ | Test MSE |
| Yelp Phoenix | 10 | 3.133650 | $10^{-6}$ | 0.1 | 1.481320 |
| Cellartracker | 1 | 1.506590 | $10^{-7}$ | 1 | 0.927066 |
| Beeradvocate | 1 | 0.603431 | $10^{-7}$ | 0.1 | 0.459174 |
| Ratebeer | 0.01 | 0.390188 | $10^{-9}$ | 1 | 0.389653 |

Table A.2.: Best Test MSE of single matrix completion and joint matrix completion and tensor factorization model after 500 iterations using ALS.

| Dataset | Matrix Completion | | Joint (MC + TF) | | |
|---|---|---|---|---|---|
| | $\lambda$ | Test MSE | $\mu$ | $\lambda$ | Test MSE |
| Yelp Phoenix | 1 | 2.904320 | 1 | 1 | 1.944050 |
| Cellartracker | 1 | 1.148010 | 100 | 0.01 | 0.363496 |
| Beeradvocate | 0.1 | 0.465695 | 10 | 0.1 | 0.373827 |
| Ratebeer | 0.1 | 0.355989 | 0.1 | 1 | 0.318692 |

The results show that our joint model produces better MSEs than matrix completion across all datasets and methods. All in all, our joint model improves the accuracy of prediction when compared to matrix completion.

# B. PROOF OF PROPOSITION 4.2.5

**Proof** First, we show that

$$\begin{aligned}\mathbb{E}\|\widetilde{\boldsymbol{z}}-\widehat{\boldsymbol{z}}\|^2 &\stackrel{\text{def}}{=} \mathbb{E}\|\widehat{\boldsymbol{Z}}\widetilde{\boldsymbol{w}}-\widehat{\boldsymbol{Z}}\boldsymbol{w}\|^2 = \mathbb{E}\|\widehat{\boldsymbol{Z}}\widetilde{\boldsymbol{w}}-\boldsymbol{z}+\boldsymbol{z}-\widehat{\boldsymbol{Z}}\boldsymbol{w}\|^2 \\ &= \mathbb{E}\|(\widehat{\boldsymbol{Z}}\widetilde{\boldsymbol{w}}-\boldsymbol{Z}\widetilde{\boldsymbol{w}})-(\widehat{\boldsymbol{Z}}\boldsymbol{w}-\boldsymbol{Z}\boldsymbol{w})\|^2 \\ &= \mathbb{E}\|(\widehat{\boldsymbol{Z}}-\boldsymbol{Z})(\widetilde{\boldsymbol{w}}-\boldsymbol{w})\|^2 = (\widetilde{\boldsymbol{w}}-\boldsymbol{w})^T\boldsymbol{\Sigma}(\widetilde{\boldsymbol{w}}-\boldsymbol{w}).\end{aligned}$$

The term $(\widetilde{\boldsymbol{w}}-\boldsymbol{w})^T\boldsymbol{\Sigma}(\widetilde{\boldsymbol{w}}-\boldsymbol{w})$ can be upper bounded by

$$\begin{aligned}(\widetilde{\boldsymbol{w}}-\boldsymbol{w})^T\boldsymbol{\Sigma}(\widetilde{\boldsymbol{w}}-\boldsymbol{w}) &= \widetilde{\boldsymbol{w}}^T\boldsymbol{\Sigma}\widetilde{\boldsymbol{w}}-\boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w}-2(\widetilde{\boldsymbol{w}}-\boldsymbol{w})^T\boldsymbol{\Sigma}\boldsymbol{w} \\ &\leq \widetilde{\boldsymbol{w}}^T\boldsymbol{\Sigma}\widetilde{\boldsymbol{w}}-\boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w}.\end{aligned}$$

The last inequality holds because the function $f(\boldsymbol{w})=\boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w}$ attains its first order optimality at $\boldsymbol{w}$ when

$$\nabla f(\boldsymbol{w})^T(\widetilde{\boldsymbol{w}}-\boldsymbol{w})\geq 0.$$

Therefore,

$$\begin{aligned}\widetilde{\boldsymbol{w}}^T\boldsymbol{\Sigma}\widetilde{\boldsymbol{w}}-\boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w} &= \widetilde{\boldsymbol{w}}^T\boldsymbol{\Sigma}\widetilde{\boldsymbol{w}}-\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{w}}+\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{w}}-\boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w} \\ &\leq \widetilde{\boldsymbol{w}}^T\boldsymbol{\Sigma}\widetilde{\boldsymbol{w}}-\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{w}}+\boldsymbol{w}^T\widetilde{\boldsymbol{\Sigma}}\boldsymbol{w}-\boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w} \\ &= \widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{w}}\left(\frac{\widetilde{\boldsymbol{w}}^T\boldsymbol{\Sigma}\widetilde{\boldsymbol{w}}}{\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{w}}}-1\right)+\boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w}\left(\frac{\boldsymbol{w}^T\widetilde{\boldsymbol{\Sigma}}\boldsymbol{w}}{\boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w}}-1\right) \\ &\leq (\widetilde{\boldsymbol{w}}^T\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{w}}+\boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w})\delta,\end{aligned}$$

where

$$\delta = \max\left( \frac{\widetilde{\boldsymbol{w}}^T \boldsymbol{\Sigma} \widetilde{\boldsymbol{w}}}{\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{\Sigma}} \widetilde{\boldsymbol{w}}} - 1 \ , \ \frac{\boldsymbol{w}^T \widetilde{\boldsymbol{\Sigma}} \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{\Sigma} \boldsymbol{w}} - 1 \right) \tag{B.1}$$

We can also show that

$$\boldsymbol{w}^T \widetilde{\boldsymbol{\Sigma}} \boldsymbol{w} \leq \boldsymbol{w}^T \boldsymbol{\Sigma} \boldsymbol{w}(1+\delta)$$

Continue the calculation, we have

$$\begin{aligned}(\widetilde{\boldsymbol{w}}^T \widetilde{\boldsymbol{\Sigma}} \widetilde{\boldsymbol{w}} + \boldsymbol{w}^T \boldsymbol{\Sigma} \boldsymbol{w})\delta &\leq (\boldsymbol{w}^T \widetilde{\boldsymbol{\Sigma}} \boldsymbol{w} + \boldsymbol{w}^T \boldsymbol{\Sigma} \boldsymbol{w})\delta \\ &\leq (\boldsymbol{w}^T \boldsymbol{\Sigma} \boldsymbol{w})(2\delta + \delta^2)\end{aligned}$$

This implies that

$$\mathbb{E}\|\widetilde{\boldsymbol{z}} - \widehat{\boldsymbol{z}}\|^2 \leq \mathbb{E}\|\widehat{\boldsymbol{z}} - \boldsymbol{z}\|^2 (2\delta + \delta^2).$$

It remains to derive an upper bound on $\delta$. To this end, we consider the generalized Rayleigh quotient of two positive definite matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. It is known that [153]

$$\max_{\boldsymbol{w} \neq 0} \frac{\boldsymbol{w}^T \boldsymbol{A} \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{B} \boldsymbol{w}} = \lambda_{\max}\left( \boldsymbol{B}^{-\frac{1}{2}} \boldsymbol{A} \boldsymbol{B}^{-\frac{1}{2}} \right).$$

Therefore,

$$\begin{aligned}\frac{\boldsymbol{w}^T \widetilde{\boldsymbol{\Sigma}} \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{\Sigma} \boldsymbol{w}} - 1 \ &\leq \max_{\boldsymbol{w} \neq 0} \ \frac{\boldsymbol{w}^T \widetilde{\boldsymbol{\Sigma}} \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{\Sigma} \boldsymbol{w}} - 1 \ = \max_{\boldsymbol{w} \neq 0} \ \frac{\boldsymbol{w}^T (\widetilde{\boldsymbol{\Sigma}} - \boldsymbol{\Sigma}) \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{\Sigma} \boldsymbol{w}} \\ &= \max_i \ \lambda_i \left( \boldsymbol{\Sigma}^{-\frac{1}{2}} (\widetilde{\boldsymbol{\Sigma}} - \boldsymbol{\Sigma}) \boldsymbol{\Sigma}^{-\frac{1}{2}} \right) \ ,\end{aligned}$$

where $\lambda_i(\boldsymbol{A})$ denotes the $i$-th eigen-value of the matrix $\boldsymbol{A}$. With some additional algebra we can show that

$$\begin{aligned}\max_i \ \lambda_i\left(\boldsymbol{\Sigma}^{-\frac{1}{2}}(\widetilde{\boldsymbol{\Sigma}}-\boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-\frac{1}{2}}\right) &= \max_i \ 1-\lambda_i\left(\boldsymbol{\Sigma}^{-\frac{1}{2}}\widetilde{\boldsymbol{\Sigma}}\boldsymbol{\Sigma}^{-\frac{1}{2}}\right)\\ &= \max_i \ 1-\lambda_i\left(\boldsymbol{\Sigma}^{-1}\widetilde{\boldsymbol{\Sigma}}\right)\\ &\leq \max_i \ \frac{1}{\lambda_i\left(\boldsymbol{\Sigma}^{-1}\widetilde{\boldsymbol{\Sigma}}\right)}-\lambda_i\left(\boldsymbol{\Sigma}^{-1}\widetilde{\boldsymbol{\Sigma}}\right)\ ,\end{aligned}$$

where the last inequality holds because for any $t \geq 0$, $|1-t| \leq |t-\frac{1}{t}|$. By recalling the definition of the matrix operator norm, we have that

$$\frac{\boldsymbol{w}^T\widetilde{\boldsymbol{\Sigma}}\boldsymbol{w}}{\boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w}}-1 \ \leq \ \left\|\boldsymbol{\Sigma}\widetilde{\boldsymbol{\Sigma}}^{-1}-\boldsymbol{\Sigma}^{-1}\widetilde{\boldsymbol{\Sigma}}\right\|_2 \stackrel{\text{def}}{=} \Delta.$$

Substituting this result into (B.1), and by symmetry, we complete the proof. ■

# REFERENCES

# REFERENCES


[1] M. Ciznicki, K. Kurowski, and A. Plaza, "Graphics processing unit implementation of jpeg2000 for hyperspectral image compression," *J. Appl. Remote Sens.*, vol. 6, no. 1, Jun 2012.

[2] B. W. Bader and T. G. Kolda, "Efficient matlab computations with sparse and factored tensors," *SIAM Journal on Scientific Computing*, vol. 30, no. 1, pp. 205–231, 2007.

[3] U. Kang, E. E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: scaling tensor analysis up by 100 times - algorithms and discoveries," in *kdd*, 2012, pp. 316–324.

[4] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in Neural Information Processing Systems*, vol. 13, 2001, pp. 556–562.

[5] J. Kim and H. Park, "Fast nonnegative matrix factorization: An activeset-like method and comparisons," *SIAM Journal on Scientific Computing*, vol. 33, no. 6, pp. 3261–3281, 2011.

[6] B. Cao, D. Shen, J.-T. Sun, X. Wang, Q. Yang, and Z. Chen, "Detect and track latent factors with online nonnegative matrix factorization," in *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, ser. IJCAI'07, 2007, pp. 2689–2694.

[7] Y.-H. Fung, C.-H. Li, and W. K. Cheung, "Online discussion participation prediction using non-negative matrix factorization," in *Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops*, ser. WI-IATW '07, 2007, pp. 284–287.

[8] 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*, vol. 23, no. 7, pp. 1087–1099, July 2012.

[9] C.-J. Lin, "Projected gradient methods for nonnegative matrix factorization," *Neural Comput.*, vol. 19, no. 10, pp. 2756–2779, Oct. 2007.

[10] J. Yu, D. Liu, D. Tao, and H. S. Seah, "Complex object correspondence construction in two-dimensional animation," *IEEE Transactions on Image Processing*, vol. 20, no. 11, pp. 3257–3269, Nov 2011.

[11] Y. Zhang, "An alternating direction algorithm for nonnegative matrix factorization," Rice, Tech. Rep., 2010.

[12] H. C. Burger, C. J. Schuler, and S. Harmeling, "Image denoising: Can plain neural networks compete with bm3d?" in *Proc. IEEE Comput Soc Conf Comput Vis Pattern Recognit*, Jun. 2012, pp. 2392–2399.

[13] J. Xie, L. Xu, and E. Chen, "Image denoising and inpainting with deep neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 341–349.

[14] X. Mao, C. Shen, and Y.-B. Yang, "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections," in *Adv Neural Inf Process Syst 29*, 2016, pp. 2802–2810.

[15] T. Remez, O. Litany, R. Giryes, and A. M. Bronstein, "Deep class-aware image denoising," in *Proc. Int Conf Sampling Theory and Applications (SampTA)*, Jul. 2017, pp. 138–142.

[16] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Trans Image Process*, vol. 26, no. 7, pp. 3142–3155, Jul. 2017.

[17] S. Lefkimmiatis, "Non-local color image denoising with convolutional neural networks," in *Proc. IEEE Comput Soc Conf Comput Vis Pattern Recognit*, Jul. 2017.

[18] K. Zhang, W. Zuo, and L. Zhang, "FFDNet: Toward a fast and flexible solution for CNN based image denoising," Oct. 2017, available online at: https://arxiv.org/abs/1710.04026.

[19] A. Buades, B. Coll, and J. M. Morel, "A non-local algorithm for image denoising," in *Proc. IEEE Comput Soc Conf Comput Vis Pattern Recognit*, vol. 2, Jun. 2005, pp. 60–65 vol. 2.

[20] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *IEEE Trans Image Process*, vol. 16, no. 8, pp. 2080–2095, Aug. 2007.

[21] A. Beck and M. Teboulle, "Fast gradient-based algorithms for constrained total variation image denoising and deblurring problems," *IEEE Trans Image Process*, vol. 18, no. 11, pp. 2419–2434, Nov. 2009.

[22] D. Zoran and Y. Weiss, "From learning models of natural image patches to whole image restoration," in *Proc. Int Conf Computer Vis*, Nov 2011, pp. 479–486.

[23] S. Gu, L. Zhang, W. Zuo, and X. Feng, "Weighted nuclear norm minimization with application to image denoising," in *Proc. IEEE Comput Soc Conf Comput Vis Pattern Recognit*, Jun. 2014, pp. 2862–2869.

[24] S. H. Chan, O. A. Elgendy, and X. Wang, "Images from bits: Non-iterative image reconstruction for quanta image sensors," *MDPI Sensors*, vol. 16, no. 11, p. 1961, 2016.

[25] T. Remez, O. Litany, and A. Bronstein, "A picture is worth a billion bits: Real-time image reconstruction from dense binary threshold pixels," in *Proc. 2016 IEEE Int. Conf. Comp. Photography (ICCP)*, May 2016, pp. 1–9.

[26] R. A. Rojas, W. Luo, V. Murray, and Y. M. Lu, "Learning optimal parameters for binary sensing image reconstruction algorithms," in *Proc. IEEE Int. Conf. Image Process. (ICIP'17)*, Sep. 2017, pp. 2791–2795.

[27] J. H. Choi and S. Vishwanathan, "Dfacto: Distributed factorization of tensors," in *Advances in Neural Information Processing Systems 27*, 2014, pp. 1296–1304.

[28] G. H. Golub and C. F. V. Loan, *Matrix Computations*. JHU Press, 2012.

[29] R. A. Horn and C. R. Johnson, *Matrix Analysis*, 2nd ed. New York, NY, USA: Cambridge University Press, 2012.

[30] P. Q. Hoang and P. P. Vaidyanathan, "Non-uniform multirate filter banks: theory and design," in *IEEE International Symposium on Circuits and Systems,*, May 1989, pp. 371–374 vol.1.

[31] M. Vetterli and J. Kovačevic, *Wavelets and Subband Coding*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995.

[32] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[33] J. A. Schouten, *Ricci Calculus - An introduction in the latest methods and problems in multi-dimmensional differential geometry*. Springer Verlag, 1924.

[34] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, Sep 1966. [Online]. Available: https://doi.org/10.1007/BF02289464

[35] E. Sanchez and B. R. Kowalski, "Generalized rank annihilation factor analysis," *Analytical Chemistry*, vol. 58, no. 2, pp. 496–499, 1986. [Online]. Available: https://doi.org/10.1021/ac00293a054

[36] A. Smilde, R. Bro, and P. Geladi, *Multi-way Analysis with Applications in the Chemical Sciences*. John Wiley and Sons, Ltd, 2004.

[37] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[38] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 177–187.

[39] J. McAuley and J. Leskovec, "Hidden Factors and Hidden Topics: Understanding Rating Dimensions with Review Text," in *Proceedings of the 7th ACM Conference on Recommender Systems*, 2013, pp. 165–172. [Online]. Available: http://doi.acm.org/10.1145/2507157.2507163

[40] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver, "Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering," in *Proceeedings of the* $4^{\text{th}}$ *ACM Conference on Recommender Systems (RecSys)*, 2010.

[41] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. H. Jr., and T. Mitchell, "Toward an architecture for never-ending language learning," in *In Proceedings of the Conference on Artificial Intelligence (AAAI)*, 2010.

[42] A. Cichocki, R. Zdunek, A. H. Phan, and S.-i. Amari, *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. Wiley Publishing, 2009.

[43] J. D. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, Sep 1970. [Online]. Available: https://doi.org/10.1007/BF02310791

[44] R. Harshman, "Foundations of the parafac procedure: models and conditions for an 'exploratory' multimodal factor analysis," in *UCLA Working Papers in Phonetics*, 1970, pp. 1–84.

[45] M. Lundy and A. Mees, "Convergence of an annealing algorithm," *Mathematical Programming*, vol. 34, no. 1, pp. 111–124, Jan 1986. [Online]. Available: https://doi.org/10.1007/BF01582166

[46] L. D. Lathauwer, B. D. Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[47] ——, "Independent component analysis and (simultaneous) third-order tensor diagonalization," *IEEE Transactions on Signal Processing*, vol. 49, no. 10, pp. 2262–2271, Oct 2001.

[48] L. D. Lathauwer, "Decompositions of a higher-order tensor in block terms? part i: Lemmas for partitioned matrices," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 3, pp. 1022–1032, 2008.

[49] ——, "Decompositions of a higher-order tensor in block terms? part ii: Definitions and uniqueness," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 3, pp. 1033–1066, 2008.

[50] C. A. Andersson and R. Bro, "Improving the speed of multi-way algorithms:: Part i. tucker3," *Chemometrics and Intelligent Laboratory Systems*, vol. 42, no. 1, pp. 93 – 103, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169743998000100

[51] L. Elden and B. Savas, "A newton?grassmann method for computing the best multilinear rank-$(r_1, r_2, r_3)$ approximation of a tensor," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 2, pp. 248–271, 2009. [Online]. Available: https://doi.org/10.1137/070688316

[52] I. V. Oseledets, D. V. Savostianov, and E. E. Tyrtyshnikov, "Tucker dimensionality reduction of three-dimensional arrays in linear time," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 3, pp. 939–956, 2008. [Online]. Available: https://doi.org/10.1137/060655894

[53] F. L. Hitchcock, "The expression of a tensor or a polyadic as a sum of products," *J. Math. Phys.*, vol. 6, pp. 164–189, 1927.

[54] J. B. Kruskal, "Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics," *Linear Algebra Appl.*, vol. 18, pp. 95–138, 1977.

[55] ——, "Rank, decomposition, and uniqueness for 3-way and n-way arrays," in *Multiway Data Analysis*. Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., 1989, pp. 7–18.

[56] V. de Silva and L.-H. Lim, "Tensor rank and the ill-posedness of the best low-rank approximation problem," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1084–1127, Sep. 2008. [Online]. Available: http://dx.doi.org/10.1137/06066518X

[57] J. M. F. ten Berge, J. de Leeuw, and P. M. Kroonenberg, "Some additional results on principal components analysis of three-mode data by means of alternating least squares algorithms," *Psychometrika*, vol. 52, no. 2, pp. 183–191, Jun 1987. [Online]. Available: https://doi.org/10.1007/BF02294233

[58] H. A. Kiers, "An alternating least squares algorithm for parafac2 and three-way dedicom," *Computational Statistics & Data Analysis*, vol. 16, no. 1, pp. 103 – 118, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016794739390247Q

[59] P. M. Kroonenberg and J. de Leeuw, "Principal component analysis of three-mode data by means of alternating least squares algorithms," *Psychometrika*, vol. 45, no. 1, pp. 69–97, Mar 1980. [Online]. Available: https://doi.org/10.1007/BF02293599

[60] J. Jian-hui, W. Hai-long, L. Yang, and Y. Ru-qin, "Three-way data resolution by alternating slice-wise diagonalization (asd) method," *Journal of Chemometrics*, vol. 14, no. 1, pp. 15–36, 2000.

[61] P. Comon, X. Luciani, and A. L. F. de Almeida, "Tensor decompositions, alternating least squares and other tales," pp. 393–405, Aug. 2009.

[62] G. Tomasi, "Use of the properties of the khatri-rao product for the computation of jacobian, hessian, and gradient of the parafac model under matlab," 2005, private communication.

[63] J. Berg, J. D. Leeuw, and P. Kroonenberg, "Some additional results on principal components analysis of three-mode data by means of alternating least squares algorithms," *Psychometrika*, vol. 52, pp. 183–191, 1987.

[64] M. J. Reynolds, A. Doostan, and G. Beylkin, "Randomized alternating least squares for canonical tensor decompositions: Application to a pde with random data," *SIAM J. Scientific Computing*, vol. 38, 2016.

[65] S. Zhe, K. Zhang, P. Wang, K.-c. Lee, Z. Xu, Y. Qi, and Z. Gharamani, "Distributed flexible nonlinear tensor factorization," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. USA: Curran Associates Inc., 2016, pp. 928–936. [Online]. Available: http://dl.acm.org/citation.cfm?id=3157096.3157200

[66] E. Acar, D. M. Dunlavy, and T. G. Kolda, "A scalable optimization approach for fitting canonical tensor decompositions," *Journal of Chemometrics*, vol. 25, no. 2, pp. 67–86, February 2011.

[67] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge Univ Press, 1990.

[68] D. S. Bernstein, *Matrix Mathematics*. Princeton University Press, 2005.

[69] M. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[70] A. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, 1999.

[71] E. Acar, T. G. Kolda, and D. M. Dunlavy, "All-at-once optimization for coupled matrix and tensor factorizations," in *MLG'11: Proceedings of Mining and Learning with Graphs*, August 2011.

[72] D. D. Lee and H. S. Seung, "Learning the parts of objects by nonnegative matrix factorization," *Nature*, vol. 401, pp. 788–791, Oct. 1999.

[73] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Computer Society*, vol. 42, pp. 30–37, Aug. 2009.

[74] M. Craig, "Minimum-volume transforms for remotely sensed data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 32, pp. 542–552, May 1994.

[75] V. P. Pauca, J. Piper, and R. J. Plemmons, "Nonnegative matrix factorization for spectral data analysis," *Linear Algebra and its Applications*, vol. 416, pp. 29–47, Jul. 2006.

[76] D. L. Zhang, P. Wang, M. N. Slipchenko, D. Ben-Amotz, A. M. Weiner, and J. X. Cheng, "Quantitative vibrational imaging by hyperspectral stimulated raman scattering microscopy and multivariate curve resolution analysis," *Analytical Chemistry*, vol. 85, pp. 98–106, Jan. 2013.

[77] R. Tauler, B. Kowalski, and S. Fleming, "Multivariate curve resolution applied to spectral data from multiple runs of an industrial process," *Analytical Chemistry*, vol. 65, pp. 2040–2047, Aug. 1993.

[78] H. Kim and H. Park, "Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis," *Bioinformatics*, vol. 23, pp. 1495–1502, Apr. 2007.

[79] D. Donoho and V. Stodden, "When does non-negative matrix factorization give a correct decomposition into parts?" in *Advances in Neural Information Processing*, 2003.

[80] C.-S. Liao, J. H. Choi, D. Zhang, S. H. Chan, and J.-X. Cheng, "Denoising stimulated raman spectroscopic images by total variation minimization," *Journal of Physical Chemistry C*, vol. 119, pp. 19 397–19 403, Jul. 2015.

[81] S. H. Chan, R. Khoshabeh, K. B. Gibson, P. E. Gill, and T. Q. Nguyen, "An augmented lagrangian method for total variation video restoration," *IEEE Transactions on Image Processing*, vol. 20, pp. 3097–3111, Nov. 2011.

[82] T. Zhang, B. Fang, W. Liu, Y. Y. Tang, G. He, and J. Wen, "Total variation norm-based nonnegative matrix factorization for identifying discriminant representation of image patterns," *Neurocomputing*, vol. 71, pp. 1824–1831, Jun. 2008.

[83] N. Seichepine, S. Essid, C. Fevotte, and O. Cappe, "Piecewise constant nonnegative matrix factorization," in *IEEE International Conference on Acoustic, Speech and Signal Processing*, May 2014, pp. 6721–6725.

[84] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, pp. 1–122, Nov. 2010.

[85] Y. Xu, W. Yin, Z. Wen, and Y. Zhang, "An alternating direction algorithm for matrix completion with nonnegative factors," *Journal of Frontiers of Mathematics in China*, vol. 7, pp. 365–384, Apr. 2011.

[86] D. Krishnan and R. Fergus, "Fast image deconvolution using hyper-laplacian priors," in *Advances in Neural Information Processing Systems*, Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, Eds. Curran Associates, Inc., 2009, pp. 1033–1041.

[87] D. Geman and C. Yang, "Nonlinear image recovery with half-quadratic regularization," *Image Processing, IEEE Transactions on*, vol. 4, pp. 932–946, Jul. 1995.

[88] S. V. Venkatakrishnan, C. A. Bouman, and B. Wohlberg, "Plug-and-play priors for model based reconstruction," in *IEEE global conference signal and information processing*, Dec. 2013, pp. 945–948.

[89] S. H. Chan, "Algorithm induced prior for image restoration," Feb. 2016, submitted, IEEE Signal Process. Letters.

[90] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *Image Processing, IEEE Transactions on*, vol. 16, pp. 2080–2095, Aug. 2007.

[91] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, and J. Chanussot, "Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, pp. 354–379, 2012.

[92] J. H. Choi, O. Elgendy, and S. Chan, "Integrating disparate sources of experts for robust image denoising," 2018, submitted to IEEE Transaction on Image Processing (TIP).

[93] F. A. Graybill and R. B. Deal, "Combining unbiased estimators," *Biometrics*, vol. 15, no. 4, pp. 543–550, Dec 1959.

[94] E. Samuel-Cahn, "Combining unbiased estimators," *The American Statistician*, vol. 48, no. 1, pp. 34–36, Feb 1994.

[95] D. B. Rubin and S. Weisberg, "The variance of a linear combination of independent estimators using estimated weights," *Biometrika*, vol. 62, no. 3, pp. 708– 709, Dec 1975.

[96] T. Keller and I. Olkin, "Combining correlated unbiased estimators of the mean of a normal distribution," United States Department of Agriculture, National Agricultural Statistics Service, NASS Research Reports 234919, Mar 2002. [Online]. Available: https://EconPapers.repec.org/RePEc:ags:unasrr:234919

[97] P. Odell, D. Dorsett, D. Young, and J. Igwe, "Estimator models for combining vector estimators," *Mathematical and Computer Modelling*, vol. 12, no. 12, pp. 1627 – 1642, 1989. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0895717789903385

[98] F. Lavancier and P. Rochet, "A general procedure to combine estimators," *Comput Stat Data Anal*, vol. 94, pp. 175–192, Feb 2016.

[99] T. Blu and F. Luisier, "The SURE-LET approach to image denoising," *IEEE Trans Image Process*, vol. 16, no. 11, pp. 2778–2786, Nov. 2007.

[100] K. N. Chaudhury and K. Rithwik, "Image denoising using optimally weighted bilateral filters: A sure and fast approach," in *Proc. IEEE Int Conf Image Process*, Sep. 2015, pp. 108–112.

[101] J. Jancsary, S. Nowozin, and C. Rother, "Loss-specific training of non-parametric image restoration models: A new state of the art," in *Computer Vision – ECCV 2012*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 112–125.

[102] F. Agostinelli, M. R. Anderson, and H. Lee, "Adaptive multi-column deep neural networks with application to robust image denoising," in *Adv Neural Inf Process Syst 26*. Curran Associates, Inc., 2013, pp. 1493–1501. [Online]. Available: http://papers.nips.cc/paper/5030-adaptive-multi-column-deep-neural-networks-with-application-to-robust-image-denoising.pdf

[103] K. Zhang, W. Zuo, S. Gu, and L. Zhang, "Learning deep CNN denoiser prior for image restoration," in *Proc. IEEE Comput Soc Conf Comput Vis Pattern Recognit*, 2017.

[104] S. H. Chan, X. Wang, and O. A. Elgendy, "Plug-and-Play ADMM for image restoration: Fixed-point convergence and applications," *IEEE Trans Computational Imaging*, vol. 3, no. 1, pp. 84–98, Mar. 2017.

[105] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans Image Process*, vol. 13, no. 4, pp. 600–612, April 2004.

[106] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1," http://cvxr.com/cvx, Mar. 2014.

[107] M. C. Grant and S. P. Boyd, "Graph implementations for nonsmooth convex programs," in *Recent Advances in Learning and Control*, ser. Lecture Notes in Control and Information Sciences. Springer-Verlag Limited, 2008, pp. 95–110.

[108] L. Condat, "Least-squares on the simplex for multispectral unmixing," Feb 2017, gIPSA-lab, Univ. Grenoble Alpes, F-38000 Grenoble, France.

[109] J. Mairal, "Optimization with first-order surrogate functions," in *Proc. Int Conf on Machine Learning*, 2013, pp. 783–791.

[110] M. Jaggi, "Convex optimization without projection steps," Dec 2011, available online at: https://arxiv.org/abs/1108.1170.

[111] C.-A. Deledalle, L. Denis, S. Tabti, and F. Tupin, "Closed-form expressions of the eigen decomposition of 2 x 2 and 3 x 3 Hermitian matrices," Université de Lyon, Research Report, 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01501221

[112] F. Luisier, T. Blu, and M. Unser, "A new SURE approach to image denoising: Interscale orthonormal wavelet thresholding," *IEEE Trans Image Process*, vol. 16, no. 3, pp. 593–606, Mar. 2007.

[113] S. Ramani, T. Blu, and M. Unser, "Monte-carlo Sure: A black-box optimization of regularization parameters for general denoising algorithms," *IEEE Trans Image Process*, vol. 17, no. 9, pp. 1540–1554, Sep. 2008.

[114] S. Cha and T. Moon, "Neural adaptive image denoiser," in *Proc. IEEE Int Conf Acoust Speech Signal Process (ICASSP'18)*, 2018. [Online]. Available: http://sigport.org/2825

[115] A. Foi, "Clipped noisy images: Heteroskedastic modeling and practical denoising," *Signal Processing*, vol. 89, no. 12, pp. 2609 – 2629, 2009, special Section: Visual Information Analysis for Security. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165168409001996

[116] L. Kang, P. Ye, Y. Li, and D. Doermann, "Convolutional neural networks for no-reference image quality assessment," in *Proc. IEEE Comput Soc Conf Comput Vis Pattern Recognit*, Washington, DC, USA, 2014, pp. 1733–1740.

[117] Y. Li, L. M. Po, L. Feng, and F. Yuan, "No-reference image quality assessment with deep convolutional neural networks," in *Proc. IEEE Int Conf Digital Signal Processing (DSP)*, Oct 2016, pp. 685–689.

[118] S. Bosse, D. Maniry, T. Wiegand, and W. Samek, "A deep neural network for image quality assessment," in *Proc. IEEE Int Conf Image Process (ICIP)*, Sept 2016, pp. 3773–3777.

[119] J. Kim, H. Zeng, D. Ghadiyaram, S. Lee, L. Zhang, and A. C. Bovik, "Deep convolutional neural models for picture-quality prediction: Challenges and solutions to data-driven image quality assessment," *IEEE Signal Process Mag*, vol. 34, no. 6, pp. 130–141, Nov 2017.

[120] Y. Li, X. Ye, and Y. Li, "Image quality assessment using deep convolutional networks," *AIP Advances*, vol. 7, no. 12, p. 125324, 2017. [Online]. Available: https://doi.org/10.1063/1.5010804

[121] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[122] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[123] J. W. Tukey, *Exploratory data analysis*. MA: Addison-Wesley, 1977, vol. 2.

[124] P. Bühlmann and B. Yu, "Boosting with the $l_2$ loss," *J Am Stat Assoc*, vol. 98, no. 462, pp. 324–339, 2003. [Online]. Available: https://doi.org/10.1198/016214503000125

[125] M. R. Charest and P. Milanfar, "On iterative regularization and its application," *IEEE Trans Circuits and Systems for Video Technology*, vol. 18, no. 3, pp. 406–411, March 2008.

[126] H. Talebi, X. Zhu, and P. Milanfar, "How to saif-ly boost denoising performance," *IEEE Trans Image Process*, vol. 22, no. 4, pp. 1470–1485, Apr 2013.

[127] M. R. Charest, M. Elad, and P. Milanfar, "A general iterative regularization framework for image denoising," in *Proc. 40th Annual Conf Information Sciences and Systems*, March 2006, pp. 452–457.

[128] S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin, "An iterative regularization method for total variation-based image restoration," *Multiscale Modeling & Simulation*, vol. 4, no. 2, pp. 460–489, 2005. [Online]. Available: https://doi.org/10.1137/040605412

[129] Y. Romano and M. Elad, "Boosting of image denoising algorithms," *SIAM J Imaging Sci*, vol. 8, no. 2, pp. 1187–1219, 2015.

[130] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proc. IEEE Comput Soc Conf Comput Vis Pattern Recognit*, 2009.

[131] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Loss functions for image restoration with neural networks," *IEEE Trans Comput. Imaging*, vol. 3, no. 1, pp. 47–57, Mar 2017.

[132] J. H. Choi, O. Elgendy, and S. Chan, "Image reconstruction for quanta image sensors using deep neural networks," Apr. 2018, to appear in International Conference on Acoustics, Speech, and Signal Processing (ICASSP).

[133] E. R. Fossum, J. Ma, S. Masoodian, L. Anzagira, and R. Zizza, "The quanta image sensor: Every photon counts," *MDPI Sensors*, vol. 16, no. 8, 2016. [Online]. Available: http://www.mdpi.com/1424-8220/16/8/1260

[134] N. A. W. Dutton, L. Parmesan, A. J. Holmes, L. A. Grant, and R. K. Henderson, "320 $\times$ 240 oversampled digital single photon counting image sensor," in *Proc. Symp VLSI Circuits*, Jun. 2014, pp. 1–2.

[135] I. M. Antolovic, S. Burri, C. Bruschini, R. Hoebe, and E. Charbon, "Nonuniformity analysis of a 65k pixel CMOS SPAD imager," *IEEE Trans. Electron Devices*, vol. 63, no. 1, pp. 57–64, Jan. 2016.

[136] S. Masoodian, J. M. D. Starkey, Y. Yamashita, and E. R. Fossum, "A 1mjot 1040fps 0.22e-rms stacked bsi quanta image sensor with cluster-parallel readout," in *Proc. Int. Image Sensor Workshop, Hiroshima, Japan.*, 2017, pp. 230–233.

[137] F. Yang, Y. M. Lu, L. Sbaiz, and M. Vetterli, "Bits from photons: Oversampled image acquisition using binary poisson statistics," *IEEE Trans. Image Process.*, vol. 21, no. 4, pp. 1421–1436, Apr. 2012.

[138] F. Yang, L. Sbaiz, E. Charbon, S. Süsstrunk, and M. Vetterli, "Image reconstruction in the gigavision camera," in *Proc. IEEE 12th Int. Conf. on Computer Vision Workshops (ICCV Workshops), 2009*, Sep. 2009, pp. 2212–2219.

[139] S. H. Chan and Y. M. Lu, "Efficient image reconstruction for gigapixel quantum image sensors," in *Proc. IEEE Global Conf. Signal and Information Processing (GlobalSIP'14)*, Dec. 2014, pp. 312–316.

[140] U. S. Kamilov and H. Mansour, "Learning optimal nonlinearities for iterative thresholding algorithms," *IEEE Signal Process. Lett.*, vol. 23, no. 5, pp. 747–751, May 2016.

[141] A. Kappeler, S. Yoo, Q. Dai, and A. K. Katsaggelos, "Super-resolution of compressed videos using convolutional neural networks," in *Proc. IEEE Int. Conf. Image Process. (ICIP'16)*, Sept 2016, pp. 1150–1154.

[142] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep CNN for image denoising," *IEEE Trans Image Process*, vol. 26, no. 7, pp. 3142–3155, July 2017.

[143] T. Meinhardt, M. Moeller, C. Hazirbas, and D. Cremers, "Learning proximal operators: Using denoising networks for regularizing inverse imaging problems," Apr. 2017, available online at: https://arxiv.org/abs/1704.03488.

[144] K. H. Jin, M. T. McCann, E. Froustey, and M. Unser, "Deep convolutional neural network for inverse problems in imaging," *IEEE Tran Image Process*, vol. 26, no. 9, pp. 4509–4522, Sept 2017.

[145] C. A. Metzler, A. Mousavi, and R. G. Baraniuk, "Learned D-AMP: Principled neural network based compressive image recovery," Nov. 2017, available online at: https://arxiv.org/abs/1704.06625.

[146] M. Iliadis, L. Spinoulas, and A. K. Katsaggelos, "Deep fully-connected networks for video compressive sensing," *Digit Signal Process*, vol. 72, pp. 9 – 18, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1051200417302130

[147] O. A. Elgendy and S. H. Chan, "Optimal threshold design for quanta image sensor," *IEEE Trans. Comput. Imaging*, vol. 4, no. 1, pp. 99–111, March 2018.

[148] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications: New York, USA, 1965.

[149] F. J. Anscombe, "The transformation of Poisson, binomial and negative-binomial data," *Biometrika*, vol. 35, no. 3-4, pp. 246–254, 1948.

[150] X. Mao, C. Shen, and Y. Yang, "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections," in *Proc. Advances in Neural Inf. Process. Syst.*, 2016.

[151] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *Proc. IEEE CS Conf Comp Vis Pattern Recognition*, Jun 2010, pp. 2528–2535.

[152] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2008 (VOC2008) Results," http://www.pascal-network.org/challenges/VOC/voc2008/workshop/index.html.

[153] S. Boyd and L. E. Ghaoui, "Method of centers for minimizing generalized eigenvalues," *Linear Algebra Appl*, vol. 188-189, pp. 63 – 111, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/002437959390465Z

VITA

# VITA

Joon Hee Choi received his B.E. in Electronics Engineering and B.B.A. in Business Administration from Sogang University in 2004; and a master's degree in 2014 and a doctoral degree in 2018 in Electrical and Computer Engineering from Purdue University. Prior to Purdue, he was an assistant manager at Samsung Electronics in South Korea from 2003 to 2012. His research interests include machine learning, image/video restoration, computer vision, and optimization algorithms for large-scale datasets.