TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Efficient Parallel Algorithms for Tensor Decompositions and Implementation in Distributed Environments

by

Christos Kolomvakis

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE MASTER OF SCIENCE OF

ELECTRICAL AND COMPUTER ENGINEERING

September 2019

THESIS COMMITTEE

Professor Athanasios P. Liavas, *Thesis Supervisor*
Professor George N. Karystinos
Associate Professor Vasilis Samoladas

Abstract

Tensors are mathematical objects that have recently become popular in the fields of machine learning and signal processing. This is due to their ability to describe multi—way relations and dependencies. For a complete data set, where we have no missing entries, tensor decompositions can help us model the data set, while others give us the potential to perform compression. For a data set with missing entries, we can use decompositions to infer missing values. Tensors however, suffer from the curse of dimensionality. The number of elements increases exponentially with respect to the order of a tensor. As a result, it is necessary to develop fast and scalable algorithms to perform these decompositions.

In this thesis, we focus on the CP decomposition, or PARAFAC decomposition. We begin this work by introducing some definitions from linear algebra, followed by matrix least squares problems. These chapters are succeded by an introduction to tensor algebra, and a presentation of the PARAFAC decomposition. The problem we deal with in this thesis is the speeding of a matrix multiplication during the decomposition algorithm. This product is known in the bibliography as the matricized–tensor Khatri-Rao product (MTTKRP) and constitutes the computational bottleneck of the algorithm. To this end, we use a data structure called a dimension tree. The tree stores intermediate results that can be reused in the decomposition algorithm, in order to speed up the MTTKRP.

Finally, we describe a parallel implementation in a distrbuted environment of this algorithm, and present results for various tensors and a range of processors.

Acknowledgements

As I am near the end of my undergraduate studies, I would like to thank several people that have supported me through this period of my life.

First, I would like to thank Professor Athanasios Liavas for the chance to work on this thesis and for his guidance.

I want to also thank my lab mates for their advice and friendship: Paris Karakasis, Giannis Papagiannakos and Ioanna Siaminou.

I would also like to thank my parents and my brothers Nick, Stelios and George for their support and encouragement.

I would like to offer special thanks to my cousins Nick K. and Aphrodite K.

Last but not least, I would like to thank my friends for their emotional support and the fun times we spent together. To name a few: Spiros A., Michalis Ch., Konstantinos G., Michalis M., Michalis M., and Dimitris T.

Table of Contents

List of Abbreviations

ADMM	Alternating Direction Method of Multipliers
ALS	Alternating Least Squares
AO	Alternating Optimization
AOO	All-at-Once Optimization
BSUM	Block Successive Upper bound Minimization
CANDECOMP	Canonical Decomposition
CPD	Canonical Polyadic Decomposition
EVD	Eigenvalue Decomposition
KKT	Karush-Kuhn-Tucker
i.i.d.	independent and identically distributed
LS	Least Squares
MNLS	Matrix Nonnegative Least Squares
MPI	Message Passing Interface
MTTKRP	Matricized Tensor Khatri - Rao Product
MU	Multiplicative Update
NMF	Nonnegative Matrix Factorization
NNLS	Nonnegative Least Squares
NTF	Nonnegative Tensor Factorization
PARAFAC	Parallel Factor Analysis

RFE	Relative Factor Error
SVD	Singular Value Decomposition
TTV	Tensor Times Vector

List of Figures

List of Tables

Chapter 1

Introduction

Tensors are mathematical objects that have recently gained great popularity due to their ability to model multiway data dependencies [2], [3], [1], [4]. Tensor factorization (or decomposition) into latent factors is very important for numerous tasks, such as feature selection, dimensionality reduction, compression, data visualization and interpretation. Tensor factorizations are usually computed as solutions of optimization problems [2], [3]. The Canonical Decomposition or Canonical Polyadic Decomposition (CANDECOMP or CPD), also known as Parallel Factor Analysis (PARAFAC), and the Tucker Decomposition are the two most widely used tensor factorization models. In this work, we focus on unconstrained and nonnegative PARAFAC. We call the latter for simplicity, Nonnegative Tensor Factorization (NTF).

Alternating Optimization (AO), All–at–Once Optimization (AOO), and Multiplicative Updates (MUs) are among the most commonly used techniques for NTF [3], [5]. Recent work for constrained tensor factorization/completion includes, among others, [6], [7], [8], and [9].

In [6], several NTF algorithms and a detailed convergence analysis have been developed. A general framework for joint matrix/tensor factorization/completion has been developed in [7]. In [8], an Alternating Direction Method of Multipliers (ADMM) algorithm for NTF has been derived, and an architecture for its parallel implementation has been outlined. However, the convergence properties of the algorithm in ill–conditioned cases are not favorable, necessitating additional research towards their improvement. In [9], the authors consider constrained matrix/tensor factorization/completion problems. They adopt the AO framework as outer loop and use the ADMM for solving the inner constrained optimization problem for one matrix factor conditioned on the rest. The ADMM offers significant flexibility, due to its ability to efficiently handle a wide range of constraints.

In [10], two parallel algorithms for unconstrained tensor factorization/completion have been developed and results concerning the speedup attained by their Message Passing Interface (MPI) implementations on a multi–core system have been reported. Related work on parallel algorithms for sparse tensor decomposition includes [11] and [12].

1.1 Purpose

In this work, we focus on dense unconstrained and NTF problems. Our aim is to derive an efficient algorithms, suitable for parallel implementation. We adopt the AO framework and solve each matrix least–squares (MNLS) problem; depending on the constraints, we either solve an unconstrained matrix least–squares problem, or a nonnegative least-squares problem via a first–order optimal (Nesterov–type) algorithm for L–smooth μ–strongly convex problems.[1] Then, we describe in detail an MPI implementation of the AO algorithms and measure the speedup attained in a multi–core environment.

1.2 Notation

Vectors, matrices, and tensors are denoted by small, capital, and calligraphic capital bold letters, respectively; for example, $\mathbf{x}$, $\mathbf{X}$, and $\boldsymbol{\mathcal{X}}$. Its elements are denoted by small nonbold letters and a set of indices, for example, $x_{i,j}$. For a matrix or a tensor, we may also denote an element by $[\mathbf{X}]_{i,j}$ and $[\boldsymbol{\mathcal{X}}]_{i,j,k}$, respectively, for convenience. We denote the column of a matrix $\mathbf{X}$ as $\mathbf{x}_i$.

Sets are denoted by blackboard bold capital letters; for example, $\mathbb{U}$. $\mathbb{R}$ denotes the set of real numbers. $\mathbb{R}_+^{I\times J\times K}$ denotes the set of $(I \times J \times K)$ real nonnegative tensors, while $\mathbb{R}_+^{I\times J}$ denotes the set of $(I \times J)$ real nonnegative matrices. $\|\cdot\|_F$ denotes the Frobenius norm of the tensor or matrix argument, $\mathbf{I}$ denotes the identity matrix of appropriate dimensions, and $(\mathbf{A})_+$ denotes the projection of matrix $\mathbf{A}$ onto the set of element–wise nonnegative matrices.

The outer product of vectors $\mathbf{a} \in \mathbb{R}^{I\times 1}$, $\mathbf{b} \in \mathbb{R}^{J\times 1}$, and $\mathbf{c} \in \mathbb{R}^{K\times 1}$ is the rank-one tensor $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c} \in \mathbb{R}^{I\times J\times K}$ with elements $[\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}]_{i,j,k} = \mathbf{a}_i\mathbf{b}_j\mathbf{c}_k$. The Khatri-Rao (columnwise Kronecker) product of compatible matrices $\mathbf{A}$ and $\mathbf{B}$ is denoted as $\mathbf{A} \odot \mathbf{B}$, the Kronecker product is denoted as $\mathbf{A} \otimes \mathbf{B}$ and the Hadamard (elementwise) product is denoted as $\mathbf{A} \circledast \mathbf{B}$.

Finally, inequality $\mathbf{A} \succeq \mathbf{B}$ means that matrix $\mathbf{A} - \mathbf{B}$ is positive semidefinite, and by $\mathbf{X} \geq \mathbf{0}$, we denote a matrix $\mathbf{X}$ which has nonnegative elements.

1.3 Thesis Outline

The thesis is organized as follows:

[1] We note that a closely related algorithm for the solution of MNLS problems has been used in [13] and [14]; we explain in detail later the performance improvement offered by our approach.

- In Chapter 2, we introduce some linear algebra background, which is needed in order to explain the matrix least-squares problems that we aim to solve. This background also serves as basis for several tensor algebra definitions, discussed in chapter 4.
- In Chapter 3, we explain the matrix–least squares problems that we will encounter in this thesis. The main focus of this chapter is the nonnegative case, as there is no closed form solution, and we must resort to iterative algorithms.
- In Chapter 4, we present an introduction to tensors. We begin with several definitions, and continue with a description of some tensor operations. The chapter concludes with a reference to matrix and tensor rank decompositions, noting a key difference between the two.
- Chapter 5 is dedicated to tensor rank decomposition. We present a data structure called a dimension tree. Its aim is to store quantities that can be re-used for later calculations. We explain how a dimension tree improves the performance of an ALS algorithm and we also point out the repeating patterns in calculating the factors.
- In Chapter 6, we describe a parallel implementation of the ALS algorithm presented in chapter 5. Parallelism is distributed and is carried out by using MPI.
- In Chapter 7, we present experimental results for the unconstrained and NTF problems. We present speedup plots as well as the exact times.
- The thesis concludes with Chapter 8, which suggests possible future work.

Chapter 2

Linear Algebra Preliminaries

2.1 Introduction

We begin by presenting several rudimentary definitions of linear algebra.

2.2 Definitions

Definition 2.1 *A vector space over a field* $\mathbb{F}$ *is a set* $\mathbb{V}$ *equipped with two binary operations. Elements in* $\mathbb{V}$ *are called vectors, while elements in* $\mathbb{F}$ *are called scalars. The binary operations are vector addition and scalar multiplication.*

For this thesis, we will refer to real vector spaces, i.e., vector spaces, where the field $\mathbb{F}$ is the field of real numbers. Vectors can either be row or column vectors. For example:

$$\mathbf{v} = \begin{bmatrix} 1 & 3 & 5 \end{bmatrix} \tag{2.1}$$

is a row vector, while

$$\mathbf{u} = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \tag{2.2}$$

is a column vector. Both of these vectors are three dimensional vectors, and we say that $\mathbf{v}, \mathbf{u} \in \mathbb{R}^3$.

Definition 2.2 *A matrix over a field* $\mathbb{F}$ *is a rectangular array of scalars each of which is a member of* $\mathbb{F}$*.*

As with vectors, we will focus on matrices where the field $\mathbb{F}$ is the field of real numbers. An example of a matrix is

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{bmatrix}. \tag{2.3}$$

We say that this matrix has three rows and two columns and we write that $\mathbf{A} \in \mathbb{R}^{3\times 2}$. In

general, for any matrix $\mathbf{A} \in \mathbb{R}^{I \times J}$ we will refer to the elements $a_{i,i}$, with $i \in \{1, \ldots, \min\{I, J\}\}$, as elements in the main diagonal of the matrix. A matrix is called a *square matrix* if it has the same number of rows and columns. A special square matrix is the identity matrix. It is denoted as $\mathbf{I}$ and all its values are equal to zero, except for the values in the main diagonal, which are equal to one. Matrices that have nonzero values only in the main diagonal are called *diagonal matrices*.

Definition 2.3 *The* ***Euclidean norm*** *of a vector* $v \in \mathbb{R}^n$ *is defined as*

$$\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2}. \tag{2.4}$$

Definition 2.4 ***The Frobenius norm*** *of a matrix* $A \in \mathbb{R}^{m \times n}$ *is defined as*

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{i,j}^2}. \tag{2.5}$$

Definition 2.5 *Addition of vectors* $\mathbf{v}, \mathbf{u} \in \mathbb{R}^n$ *produces a third vector* $\mathbf{x} \in \mathbb{R}^n$ *which is defined as*

$$x_i = v_i + u_i, \text{ for all } i \in \{1, \ldots, n\}. \tag{2.6}$$

Definition 2.6 *Scalar multiplication of vector* $\mathbf{v} \in \mathbb{R}^n$ *with a scalar* $a \in \mathbb{R}$ *produces a vector* $\mathbf{x} \in \mathbb{R}^n$ *which is defined as*

$$x_i = av_i, \text{ for all } i \in \{1, \ldots, n\}. \tag{2.7}$$

Definition 2.7 *Addition of matrices* $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$ *produces a matrix* $\mathbf{C} \in \mathbb{R}^{m \times n}$ *which is defined as*

$$c_{i,j} = a_{i,j} + b_{i,j}, \tag{2.8}$$

for all $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$.

Definition 2.8 *The* ***transpose*** *of a vector* $\mathbf{v}$ *is denoted as* $\mathbf{v}^T$. *If* $\mathbf{v}$ *is a row vector, it produces a column vector with the same elements, and if* $\mathbf{v}$ *is a column vector, it produces a row vector with the same elements.*

Definition 2.9 *The **inner product** of vectors* $\mathbf{v}, \mathbf{u} \in \mathbb{R}^n$ *is a scalar defined as*

$$\langle \mathbf{v}, \mathbf{u} \rangle = \mathbf{v}^T \mathbf{u} = \sum_{i=1}^{n} v_i u_i. \tag{2.9}$$

Definition 2.10 *The **transpose** of a matrix* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *is denoted as* $\mathbf{A}^T$*, is in* $\mathbb{R}^{n \times m}$ *and is defined as*

$$[\mathbf{A}^T]_{i,j} = [\mathbf{A}]_{j,i}. \tag{2.10}$$

Definition 2.11 *Matrix multiplication of* $\mathbf{A} \in \mathbb{R}^{n \times m}$ *and* $\mathbf{B} \in \mathbb{R}^{m \times p}$ *produces a matrix* $\mathbf{C} \in \mathbb{R}^{m \times p}$*, which is defined elementwise as*

$$c_{i,j} = \sum_{k=1}^{m} a_{i,k} b_{k,j}, \tag{2.11}$$

for all $i \in \{1, \ldots, n\}$ *and* $j \in \{1, \ldots, p\}$*. In other words,* $c_{i,j}$ *is the inner product of the* i*–th row of* $\mathbf{A}$ *and the* j*–th column of* $\mathbf{B}$*. If* $n = 1$ *or* $p = 1$*, we have a matrix–vector multiplication. The inner product is the case where both* n *and* p *are equal to one.*

Matrix multiplication is noncommutative. If in the definition above $n = p$ also held, then in general

$$\mathbf{AB} \neq \mathbf{BA}. \tag{2.12}$$

The identity matrix $\mathbf{I}$, which was defined above, is the neutral element of matrix multiplication, regardless of whether we perform left or right multiplication

$$\mathbf{AI} = \mathbf{A} = \mathbf{IA}. \tag{2.13}$$

Definition 2.12 *The **inverse** of a square matrix* $\mathbf{A} \in \mathbb{R}^{n \times n}$ *is denoted as* $\mathbf{A}^{-1}$ *and it has the following property*

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}. \tag{2.14}$$

We note here, that the inverse of a matrix may not exist. Matrices that have an inverse are called nonsingular or invertible, while matrices that do not have an inverse are called singular.

Definition 2.13 *Let a square matrix* $\mathbf{A} \in \mathbb{R}^{n \times n}$*. A scalar* $\lambda \in \mathbb{R}$ *is called an eigenvalue of*

A *if there is a nontrivial solution to the system*

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}. \tag{2.15}$$

Such an $\mathbf{x} \in \mathbb{R}^n$ *is called an eigenvalue of matrix* $\mathbf{A}$.

Definition 2.14 *Any symmetric matrix* $\mathbf{A}$ *admits the following decomposition*

$$\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T, \tag{2.16}$$

where $\mathbf{U}, \mathbf{\Lambda} \in \mathbb{R}^{n\times n}$. *Each column of* $\mathbf{U}$ *contains an eigenvector of* $\mathbf{A}$, *while* $\mathbf{\Lambda}$ *is a diagonal matrix containing all of its eigenvalues. This decomposition is called eigenvalue decomposition (EVD), or spectral decomposition.*

Based on the spectral decomposition of a matrix, we may raise a matrix to an arbitrary power; it can be proven that

$$\mathbf{A}^k = \mathbf{U}\mathbf{\Lambda}^k\mathbf{U}^T, \tag{2.17}$$

where $k \in \mathbb{R}$.

Definition 2.15 *Any matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *admits the following decomposition*

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \tag{2.18}$$

where $\mathbf{U} \in \mathbb{R}^{m\times m}$, $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ *and* $\mathbf{V} \in \mathbb{R}^{n\times n}$. *Each column of* $\mathbf{U}$ *contains a vector called a* ***left singular vector*** *of* $\mathbf{A}$, *each column of* $\mathbf{V}$ *contains a vector called a* ***right singular vector*** *of* $\mathbf{A}$ *while* $\mathbf{\Sigma}$ *is a diagonal matrix that contains nonnegative numbers that are called* ***singular values*** *of* $\mathbf{A}$. *This decomposition is called* ***singular value decomposition*** *(****SVD****) and is a generalization of the eigenvalue decomposition for nonsquare matrices.*

Definition 2.16 *The rank of a matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$, *denoted as* $rank(\mathbf{A})$, *is the maximal number of linearly independent columns. It can be proven that the rank is also equal to the maximal number of linearly independent rows. Values of the rank range from* 1 *to* $\min(m, n)$. *If* $rank(\mathbf{A}) = \min(m, n)$, *we say that* $\mathbf{A}$ *is full rank.*

There is a link between the rank of a square matrix and the existence of its inverse. If a square matrix is full rank, then it is invertible.

Definition 2.17 *The **Moore–Penrose pseudoinverse** of a matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *is denoted as* $\mathbf{A}^\dagger$*, is in* $\mathbb{R}^{n\times m}$ *and has the following properties*

- $\mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}$,
- $\mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger = \mathbf{A}^\dagger$,
- $\left(\mathbf{A}\mathbf{A}^\dagger\right)^T = \mathbf{A}\mathbf{A}^\dagger$,
- $\left(\mathbf{A}^\dagger\mathbf{A}\right)^T = \mathbf{A}^\dagger\mathbf{A}$.

If $\mathbf{A}^T\mathbf{A}$ *is invertible, then*

$$\mathbf{A}^\dagger = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T \tag{2.19}$$

and $\mathbf{A}^\dagger$ *is a **left inverse** of* $\mathbf{A}$*, i.e.* $\mathbf{A}^\dagger\mathbf{A} = \mathbf{I}$*. If* $\mathbf{A}\mathbf{A}^T$ *is invertible, then*

$$\mathbf{A}^\dagger = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1} \tag{2.20}$$

and $\mathbf{A}^\dagger$ *is a **right inverse** of* $\mathbf{A}$*, i.e.* $\mathbf{A}\mathbf{A}^\dagger = \mathbf{I}$*.*

Definition 2.18 *Let* $\mathbf{A} \in \mathbb{R}^{N\times M}$ *and* $\mathbf{B} \in \mathbb{R}^{P\times K}$*. The **Kronecker product** (or tensor product) of* $\mathbf{A}$ *and* $\mathbf{B}$ *is defined as the matrix*

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{1,1}\mathbf{B} & \cdots & a_{1,M}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{N,1}\mathbf{B} & \cdots & a_{N,M}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{NP\times MK}. \tag{2.21}$$

Definition 2.19 *Let* $\mathbf{A} \in \mathbb{R}^{N\times M}$ *and* $\mathbf{B} \in \mathbb{R}^{P\times M}$*. The **Khatri–Rao product** of* $\mathbf{A}$ *and* $\mathbf{B}$ *is defined as the matrix*

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \cdots & \mathbf{a}_M \otimes \mathbf{b}_M \end{bmatrix} \in \mathbb{R}^{NP\times M}. \tag{2.22}$$

Definition 2.20 *Let* $\mathbf{A} \in \mathbb{R}^{N\times M}$ *and* $\mathbf{B} \in \mathbb{R}^{N\times M}$*. The **Hadamard Product** or element-wise matrix product of* $\mathbf{A}$ *and* $\mathbf{B}$ *is a matrix of size* $N \times M$*, and is defined as*

$$[\mathbf{A} \circledast \mathbf{B}]_{n,m} = a_{n,m}b_{n,m}, \tag{2.23}$$

for all $n \in \{1, \ldots, N\}$, $m \in \{1, \ldots, M\}$.

Regarding the Kronecker and Khatri–Rao products, we note that they are both associative [4]. For example, a Khatri–Rao product of three matrices can be equivalently

calculated as

$$\mathbf{A} \odot \mathbf{B} \odot \mathbf{C} = (\mathbf{A} \odot \mathbf{B}) \odot \mathbf{C} = \mathbf{A} \odot (\mathbf{B} \odot \mathbf{C}) . \tag{2.24}$$

For the latter three products, the following properties hold [1]:

$$(\mathbf{A} \otimes \mathbf{B}) (\mathbf{C} \otimes \mathbf{D}) = \mathbf{AC} \otimes \mathbf{BD}, \tag{2.25}$$

$$(\mathbf{A} \otimes \mathbf{B})^{\dagger} = \mathbf{A}^{\dagger} \otimes \mathbf{B}^{\dagger}, \tag{2.26}$$

$$(\mathbf{A} \odot \mathbf{B})^{T} (\mathbf{A} \odot \mathbf{B}) = \mathbf{A}^{T}\mathbf{A} \circledast \mathbf{B}^{T}\mathbf{B}, \tag{2.27}$$

$$(\mathbf{A} \odot \mathbf{B})^{\dagger} = ((\mathbf{A}^{T}\mathbf{A}) \circledast (\mathbf{B}^{T}\mathbf{B}))^{\dagger}(\mathbf{A} \odot \mathbf{B})^{T}. \tag{2.28}$$

Based on the associativity of these products, we can derive generalized expressions for properties (2.25), (2.27), and (2.28) for three or more operands.

Chapter 3

Matrix Least Squares Problems

In this chapter, we describe the matrix least squares problem, which will be our workhorse towards the development of efficient algorithms for tensor factorizations.

3.1 Matrix Least Squares

Let $\mathbf{X} \in \mathbb{R}^{M \times N}$ and $\mathbf{B} \in \mathbb{R}^{N \times R}$ and consider the problem

$$\min_{\mathbf{A}} f(\mathbf{A}) = \frac{1}{2}\|\mathbf{X} - \mathbf{A}\mathbf{B}^T\|_F^2. \tag{3.1}$$

This problem is known as the matrix least squares problem. The cost function is convex, therefore, the existence of a global minimizer of f, $\mathbf{A}^*$, is guaranteed. The solution $\mathbf{A}^*$ must satisfy the following linear system of equations

$$\mathbf{X}\mathbf{B} = \mathbf{A}^*\mathbf{B}^T\mathbf{B}, \tag{3.2}$$

which are known as *normal equations*. Thus, $\mathbf{A}^*$ is given by

$$\mathbf{A}^* = \mathbf{X}\mathbf{B}^\dagger = \mathbf{X}\mathbf{B}\left(\mathbf{B}^T\mathbf{B}\right)^{-1}. \tag{3.3}$$

For an extensive discussion on the computational aspects of the solution of the normal equations the reader is referred to [15].

3.2 Matrix Nonnegative LS

Let $\mathbf{X} \in \mathbb{R}^{m \times n}$, $\mathbf{A} \in \mathbb{R}_+^{m \times r}$, $\mathbf{B} \in \mathbb{R}^{n \times r}$, and consider the Matrix Nonnegative LS (MNLS) problem

$$\min_{\mathbf{A} \geq \mathbf{0}} f(\mathbf{A}) := \frac{1}{2}\|\mathbf{X} - \mathbf{A}\mathbf{B}^T\|_F^2. \tag{3.4}$$

The problem given in (3.4), is convex, however, it does not have a closed form solution. Hence, we resort to iterative methods that solve efficiently problems of this form in the

following sections. The next section focuses on L–smooth μ–strongly convex optimization problems.

3.2.1 Optimal first–order methods for L–smooth μ–strongly convex optimization problems

We consider optimization problems of smooth and strongly convex functions and briefly present results concerning their information complexity and the associated first–order optimal algorithms (for a detailed exposition see [16, Chapter 2]).

We assume that $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth (that is, differentiable up to a sufficiently high order) convex function, with gradient $\nabla f(\mathbf{x})$ and Hessian $\nabla^2 f(\mathbf{x})$. Our aim is to solve the problem

$$\min_{\mathbf{x}} f(\mathbf{x}), \tag{3.5}$$

within accuracy $\epsilon > 0$. The solution accuracy is defined as follows. If $f^* := \min_{\mathbf{x}} f(\mathbf{x})$, then point $\bar{\mathbf{x}} \in \mathbb{R}^n$ solves problem (3.5) within accuracy ϵ if $f(\bar{\mathbf{x}}) - f^* \leq \epsilon$.

Let $0 < \mu \leq L < \infty$. A smooth convex function f is called L–smooth or, using the notation of [16, p. 66], $f \in \mathcal{S}^{\infty,1}_{0,L}$, if

$$\mathbf{0} \preceq \nabla^2 f(\mathbf{x}) \preceq L\mathbf{I}, \quad \forall \mathbf{x} \in \mathbb{R}^n, \tag{3.6}$$

and L–smooth μ–strongly convex, or $f \in \mathcal{S}^{\infty,1}_{\mu,L}$, if

$$\mu\mathbf{I} \preceq \nabla^2 f(\mathbf{x}) \preceq L\mathbf{I}, \quad \forall\, \mathbf{x} \in \mathbb{R}^n. \tag{3.7}$$

The number of iterations that first-order methods need for the solution of problem (3.5), within accuracy ϵ, is $O\left(\frac{1}{\sqrt{\epsilon}}\right)$ if $f \in \mathcal{S}^{\infty,1}_{0,L}$, and $O\left(\sqrt{\frac{L}{\mu}} \log \frac{1}{\epsilon}\right)$ if $f \in \mathcal{S}^{\infty,1}_{\mu,L}$ [16, Theorem 2.2.2]. The convergence rate in the first case is *sublinear* while, in the second case, it is *linear* and determined by the condition number of the problem, $\mathcal{K} := \frac{L}{\mu}$. Thus, strong convexity is a very important property that should be exploited whenever possible.

An algorithm that achieves this complexity, and, thus, is first–order optimal, appears in Algorithm 1 (see, also [16, p. 80]). This algorithm can handle both the L-smooth case, by setting $q = 0$, and the L–smooth μ–strongly convex case, by setting $q = \frac{\mu}{L} > 0$.

If the problem of interest is the constrained problem

$$\min_{\mathbf{x} \in \mathbb{X}} f(\mathbf{x}), \tag{3.8}$$

Algorithm 1 Nesterov algorithm for L–smooth μ–strongly convex optimization problems

procedure NESTEROV_ALGORITHM($\mathbf{x}_0 \in \mathbb{R}^n$, μ, L, k)
  Set $\mathbf{y}_0 = \mathbf{x}_0$, $\alpha_0 \in (0,1)$, $q = \frac{\mu}{L}$
    $\mathbf{x}_{k+1} = \mathbf{y}_k - \frac{1}{L}\nabla f(\mathbf{y}_k)$
    $\alpha_{k+1} \in (0,1)$ from $\alpha_{k+1}^2 = (1-\alpha_{k+1})\alpha_k^2 + q\alpha_{k+1}$
    $\beta_{k+1} = \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2+\alpha_{k+1}}$
    $\mathbf{y}_{k+1} = \mathbf{x}_{k+1} + \beta_{k+1}(\mathbf{x}_{k+1} - \mathbf{x}_k)$
end procedure

where $\mathbb{X}$ is a closed convex set, then the corresponding optimal algorithm is very much alike Algorithm 1, with the only difference being in the computation of $\mathbf{x}_{k+1}$. We now have that [16, p. 90]

$$\mathbf{x}_{k+1} = \Pi_{\mathbb{X}}\left(\mathbf{y}_k - \frac{1}{L}\nabla f(\mathbf{y}_k)\right), \tag{3.9}$$

where $\Pi_{\mathbb{X}}(\cdot)$ denotes the Euclidean projection onto set $\mathbb{X}$. The convergence properties of this algorithm are the same as those of Algorithm 1. If the projection onto set $\mathbb{X}$ is easy to compute, then the algorithm is both theoretically optimal and very efficient in practice.

3.2.2 Optimal first–order methods for L–smooth μ–strongly convex MNLS problems

In this section, we present an optimal first–order algorithm for the solution of L–smooth μ–strongly convex MNLS problems. Optimal first–order methods have recently attracted great research interest because they are strong candidates and, in many cases, the only viable way for the solution of very large optimization problems.

Nesterov–type algorithm for MNLS with proximal term

In the sequel, we present a Nesterov–type algorithm for the MNLS problem with proximal term. Let $\mathbf{X} \in \mathbb{R}^{m\times n}$, $\mathbf{A} \in \mathbb{R}^{m\times r}$, $\mathbf{B} \in \mathbb{R}^{n\times r}$, and consider the problem

$$\min_{\mathbf{A}\geq\mathbf{0}} f(\mathbf{A}) := \frac{1}{2}\left\|\mathbf{X} - \mathbf{A}\mathbf{B}^T\right\|_F^2. \tag{3.10}$$

The gradient and Hessian of f, at point $\mathbf{A}$, are, respectively,

$$\nabla f(\mathbf{A}) = -\left(\mathbf{X} - \mathbf{A}\mathbf{B}^T\right)\mathbf{B} \tag{3.11}$$

Algorithm 2 Nesterov–type algorithm for MNLS with proximal term

procedure NESTEROV_MNLS($\mathbf{X} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times r}$, $\mathbf{A}_* \in \mathbb{R}^{m\times r}$)
  $L = \max(\text{eig}(\mathbf{B}^T\mathbf{B}))$
  $\mu = \min(\text{eig}(\mathbf{B}^T\mathbf{B}))$
  $\lambda = g(L, \mu)$
  $\mathbf{W} = -\mathbf{X}\mathbf{B} - \lambda\mathbf{A}_*$
  $\mathbf{Z} = \mathbf{B}^T\mathbf{B} + \lambda\mathbf{I}$
  $q = \frac{\mu+\lambda}{L+\lambda}$
  $\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$
  $\alpha_0 = 1$, $k = 0$
  while (1) **do**
    $\nabla f_{\mathtt{P}}(\mathbf{Y}_k) = \mathbf{W} + \mathbf{Y}_k\mathbf{Z}$
    if (terminating_condition is TRUE) then
      break
    else
      $\mathbf{A}_{k+1} = \left(\mathbf{Y}_k - \frac{1}{L+\lambda}\nabla f_{\mathtt{P}}(\mathbf{Y}_k)\right)_+$
      $\alpha_{k+1}^2 = (1-\alpha_{k+1})\alpha_k^2 + q\alpha_{k+1}$
      $\beta_{k+1} = \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2+\alpha_{k+1}}$
      $\mathbf{Y}_{k+1} = \mathbf{A}_{k+1} + \beta_{k+1}\left(\mathbf{A}_{k+1} - \mathbf{A}_k\right)$ $k = k+1$
    end if
  end while
  return $\mathbf{A}_k$
end procedure

and

$$\nabla^2 f(\mathbf{A}) := \frac{\partial^2 f(\mathbf{A})}{\partial \text{vec}(\mathbf{A}) \partial \text{vec}(\mathbf{A})^T} = \mathbf{B}^T\mathbf{B} \otimes \mathbf{I} \succeq \mathbf{0}. \tag{3.12}$$

Let $L := \max(\text{eig}(\mathbf{B}^T\mathbf{B}))$ and $\mu := \min(\text{eig}(\mathbf{B}^T\mathbf{B}))$. If $\mu = 0$ (for example, if $r > n$), then problem (3.10) is L-smooth. If $\mu > 0$, then problem (3.10) is L–smooth μ-strongly convex. A first–order optimal algorithm for the solution of (3.10) can be derived using the approach of Section 3.2.1. We note that [13] and [14] solved problem (3.10) using a variation of Algorithm 1, which is equivalent to Algorithm 1 with $\mu = 0$. However, if $\mu > 0$, then this algorithm is *not* first–order optimal and, as we shall see later, it performs much worse than the optimal.

We note that the values of L and μ are necessary for the development of the Nesterov-type algorithm, thus, their computation is imperative.[1]

[1] An alternative to their direct computation is to estimate L using line-search techniques and overcome the computation of μ using heuristic adaptive restart techniques [17]. However, in our case, this alternative is computationally demanding, especially for large–scale problems, and shall not be considered.

Under the AO framework, in order to avoid very ill–conditioned problems (and guarantee strong convexity), we introduce a proximal term and solve problem

$$\min_{\mathbf{A}\geq\mathbf{0}} f_{\mathtt{P}}(\mathbf{A}) := \frac{1}{2}\left\|\mathbf{X}-\mathbf{A}\mathbf{B}^T\right\|_F^2 + \frac{\lambda}{2}\|\mathbf{A}-\mathbf{A}_*\|_F^2, \tag{3.13}$$

for given $\mathbf{A}_*$ and appropriately chosen λ. We choose λ based on L and μ, and denote this functional dependence as $\lambda = g(L,\mu)$. If $\frac{\mu}{L} \ll 1$, then we may set $\lambda \approx 10\mu$, significantly improving the conditioning of the problem by putting large weight on the proximal term; however, in this case, we expect that the optimal point will be biased towards $\mathbf{A}_*$. Otherwise, we may set $\lambda \lessapprox \mu$, putting small weight on the proximal term and permitting significant progress towards the computation of $\mathbf{A}$ that satisfies approximate equality $\mathbf{X} \approx \mathbf{A}\mathbf{B}^T$ as accurately as possible.

The gradient of $f_{\mathtt{P}}$, at point $\mathbf{A}$, is

$$\nabla f_{\mathtt{P}}(\mathbf{A}) = -\left(\mathbf{X}-\mathbf{A}\mathbf{B}^T\right)\mathbf{B} + \lambda(\mathbf{A}-\mathbf{A}_*). \tag{3.14}$$

The Karush–Kuhn–Tucker (KKT) conditions for problem (3.13) are [13]

$$\nabla f_{\mathtt{P}}(\mathbf{A}) \geq \mathbf{0},\ \mathbf{A} \geq \mathbf{0},\ \nabla f_{\mathtt{P}}(\mathbf{A}) \circledast \mathbf{A} = \mathbf{0}. \tag{3.15}$$

These expressions can be used in a terminating condition. For example, we may terminate the algorithm if

$$\min_{i,j}\left([\nabla f_{\mathtt{P}}(\mathbf{A})]_{i,j}\right) > -\delta_1, \max_{i,j}\left(\left|[\nabla f_{\mathtt{P}}(\mathbf{A}) \circledast \mathbf{A}]_{i,j}\right|\right) < \delta_2, \tag{3.16}$$

for small positive real numbers δ_1 and δ_2. Of course, other criteria, based, for example, on the (relative) change of the cost function can be used in terminating conditions.

A Nesterov–type algorithm for the solution of the MNLS problem with proximal term (3.13) is given in Algorithm 2. For notational convenience, we denote Algorithm 2 as

$$\mathbf{A}_{\text{opt}} = \text{Nesterov\_MNLS}(\mathbf{X},\mathbf{B},\mathbf{A}_*).$$

Computational complexity of Algorithm 2

Quantities $\mathbf{W}$ and $\mathbf{Z}$ are computed once per algorithm call and cost, respectively, $O(mnr)$ and $O(rn^2)$ arithmetic operations. Quantities L and μ are also computed once and cost at most $O(r^3)$ operations. $\nabla f_{\mathtt{P}}(\mathbf{Y}_k)$, $\mathbf{A}_k$, and $\mathbf{Y}_k$ are updated in every iteration

with cost $O(mr^2)$, $O(mr)$, and $O(mr)$ arithmetic operations, respectively.

Chapter 4

Tensor Algebra Preliminaries

4.1 Introduction

Before introducing the algorithms used in this thesis, it is necessary to present several properties of tensors. Initially, we will give definitions regarding what a tensor is, tensor operations, and we will note a key difference between matrices and tensors.

4.2 Definitions

Definition 4.1 *A tensor is a multidimensional array. The* ***order*** *of a tensor defines the number of dimensions a tensor has.*

Based on the definition above, a third order tensor is a tensor with three indices. An illustration is shown in Figure 4.1.

A matrix is a second–order tensor and a vector is a first–order tensor. We collectively refer to tensors of order three or higher as higher–order tensors.

As in matrices, the operations of addition and subtraction are also defined for tensors, in a straightforward way. There also exist a tensor analogue of matrix multiplication and matrix–vector multiplication, which will be defined later in this chapter.

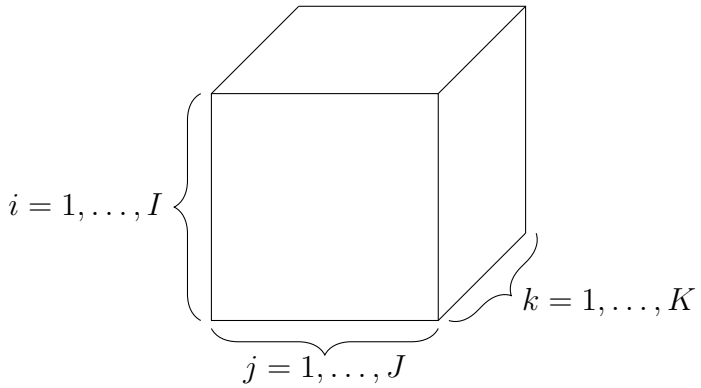

Figure 4.1: A third order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$.

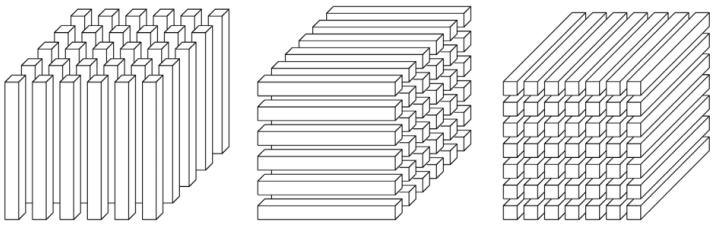

Figure 4.2: Mode–1, mode–2 and mode–3 fibers for a third order tensor, respectively. This image is derived from [1].

Definition 4.2 *The **Frobenius norm** of a tensor* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ *is defined as*

$$\|\boldsymbol{\mathcal{X}}\| = \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1,i_2,\ldots,i_N}^2}. \tag{4.1}$$

Definition 4.3 *A **fiber** of a tensor is a vector defined by fixing all indices of a tensor but one.*

As an example, let us assume a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$. $x_{:,j,k}$ defines a mode–1 fiber, $x_{i,:,k}$ defines a mode–2 fiber and $x_{i,j,:}$ defines a mode–3 fiber, as shown in the figure below.

Definition 4.4 *Let* $\mathbf{a} \in \mathbb{R}^N$, $\mathbf{b} \in \mathbb{R}^P$, *and* $\mathbf{c} \in \mathbb{R}^J$. *The **outer product** of* $\mathbf{a}$ *and* $\mathbf{b}$ *is defined as the **rank–one matrix** with elements*

$$[\mathbf{a} \circ \mathbf{b}]_{n,p} = a_n b_p, \tag{4.2}$$

for all $n \in \{1, \ldots, N\}$, $p \in \{1, \ldots, P\}$. *In the same manner, the outer product of* $\mathbf{a}$, $\mathbf{b}$ *and* $\mathbf{c}$ *is defined as the **rank–one tensor** with elements*

$$[\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}]_{n,p,j} = a_n b_p c_j, \tag{4.3}$$

for all $n \in \{1, \ldots, N\}$, $p \in \{1, \ldots, P\}$, *and* $j \in \{1, \ldots, J\}$.

Rank–1 tensors for N–th order tensors, with $N > 3$, are defined similarly.

Definition 4.5 *Let* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. *The **matricization of the tensor with respect to the** j**–th mode** (mode–j matricization) is defined as the matrix* $\mathbf{X}_{(j)} \in \mathbb{R}^{I_j \times \prod_{k=1,k\neq j}^{N} I_k}$, *where the element* $[\boldsymbol{\mathcal{X}}]_{i_1,i_2,\ldots,i_j,\ldots,i_N}$ *is mapped to* $[\mathbf{X}_{(j)}]_{i_j,k}$ *according to [1]*:

$$k = 1 + \sum_{\substack{p=1 \\ p\neq j}}^{N} (i_p - 1) J_p, \text{ where } J_p = \prod_{\substack{m=1 \\ m\neq j}}^{p-1} I_m.$$

This can be better understood through an example. Let a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$. We are interested in examining the mode–1 matricization of $\boldsymbol{\mathcal{X}}$, $\mathbf{X}_{(1)} \in \mathbb{R}^{I \times JK}$. Then, element $[\boldsymbol{\mathcal{X}}]_{i_1,i_2,i_3}$ will be mapped to element $[\mathbf{X}_{(1)}]_{i_1,k}$, where

$$k = 1 + (i_2 - 1) + (i_3 - 1)I_2, \tag{4.4}$$

since $\prod_{m=2}^{1} I_m = \prod_{m \in \emptyset} I_m = 1$ (by convention) and $J_3 = \prod_{m=2}^{2} I_m = I_2$. For the mode–2 matricization, $\mathbf{X}_{(2)} \in \mathbb{R}^{J \times IK}$, element $[\boldsymbol{\mathcal{X}}]_{i_1,i_2,i_3}$ will be mapped to element $[\mathbf{X}_{(2)}]_{i_2,k'}$, where

$$k' = 1 + (i_1 - 1) + (i_3 - 1)I_1,$$

since $J_1 = \prod_{m=1}^{0} I_m = 1$, $J_3 = \prod_{m=2}^{2} I_m = I_1$, and so on.

Definition 4.6 *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_n \cdots \times I_N}$ and $\mathbf{U} \in \mathbb{R}^{I_n \times J}$. The* **n–*mode product*** *of $\boldsymbol{\mathcal{X}}$ and $\mathbf{U}$, denoted as $\boldsymbol{\mathcal{X}} \times_n \mathbf{U}$, is a tensor of size $I_1 \times I_2 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$ and it is defined, elementwise, as*

$$(\boldsymbol{\mathcal{X}} \times_n \mathbf{U})_{i_1,\ldots,i_{n-1},j,i_{n+1},\ldots,I_N} = \sum_{i_n=1}^{I_N} x_{i_1,i_2,\ldots,i_n,\ldots,i_N} u_{i_n,j}.$$

The idea can also be expressed in terms of the mode–n matricization of $\boldsymbol{\mathcal{X}}$ since

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_n \mathbf{U} \Leftrightarrow \mathbf{Y}_{(n)} = \mathbf{U}^T \mathbf{X}_{(n)}.$$

Definition 4.7 *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_n \cdots \times I_N}$ and $\mathbf{v} \in \mathbb{R}^{I_n}$. The* **n–*mode vector product*** *of $\boldsymbol{\mathcal{X}}$ with $\mathbf{v}$ is denoted as $\boldsymbol{\mathcal{X}} \times_n \mathbf{v}$, and produces a new tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{n-1} \times 1 \times I_{n+1} \times \cdots \times I_N}$ and is defined as*

$$\boldsymbol{\mathcal{Y}}_{i_1,i_2,\ldots,i_{n-1},i_{n+1},\ldots,i_N} = \sum_{j=1}^{I_n} x_{i_1,i_2,\ldots,i_{n-1},j,i_{n+1},\ldots,i_N} v_j.$$

The new tensor $\boldsymbol{\mathcal{Y}}$ has order $(N-1)$. This operation is also called ***tensor–times–vector multiplication (TTV)****.*

Products between tensors can also be defined, but are beyond the scope of this thesis.

4.3 Rank Decompositions

In several problems, we may wish to model a data set using multilinear models. Procedures that can provide such models are called *matrix* or *tensor factorizations* or *decompositions*. If we choose to model using sums of rank–1 components, then we we have a *rank decomposition* or *rank factorization*. Some applications are referred at the end of this section. Before proceeding, we give an alternative definition for the rank of a matrix.

Definition 4.8 *Let* $\mathbf{A} \in \mathbb{R}^{m \times n}$. *The* ***rank*** *of* $\mathbf{A}$ *is the minimal number of rank–1 matrices that we need to add, in order to have an exact approximation of* $\mathbf{A}$.

The **CP rank** of a tensor, which we will refer to for the rest of this thesis as the rank of a tensor, is defined in a similar manner.

4.3.1 Matrix Case

In the matrix case, we can get a rank decomposition through the SVD. For an arbitrary matrix $\mathbf{A} \in \mathbb{R}^{I \times J}$ with $rank(\mathbf{A}) = R$ we have:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\dagger} = \sum_{i=1}^{R} \sigma_i \mathbf{u}_i \circ \mathbf{v}_i. \tag{4.5}$$

Computing the SVD has a complexity of $O(mn^2)$ [18]. The rank R is equal to the number of nonzero singular values. As a last note, we mention that the rank decomposition is not unique. If we consider an orthogonal matrix $Q \in \mathbb{R}^{I \times I}$, equation (4.5) can be rewritten as

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\dagger} = \mathbf{U}\mathbf{Q}\mathbf{Q}^{T}\mathbf{\Sigma}\mathbf{V}^{\dagger} = \widetilde{\mathbf{U}}\widetilde{\mathbf{\Sigma}}\mathbf{V}^{\dagger}. \tag{4.6}$$

Uniqueness can be only be achieved by imposing orthogonality constraints on matrices $\mathbf{U}$ and $\mathbf{V}$.

4.3.2 Tensor Case

The rank decomposition of a tensor is given by

$$\boldsymbol{\mathcal{X}} = [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!] = \sum_{r=1}^{R} \mathbf{u}_r^{(1)} \circ \cdots \circ \mathbf{u}_r^{(N)}. \tag{4.7}$$

This decomposition is called PARAFAC decomposition, or CP decomposition. Computing the rank of a tensor, in contrast to the matrix case, is an NP-hard problem [19]. As

a result, we cannot hope that we wiil always be able to find an exact approximation of the initial tensor. Bounds on the rank of tensors for several orders are listed in [4]. In practice, the rank of a tensor is determined numerically by fitting various rank–R decompositions. It has been shown, however, that, under mild conditions, a tensor rank decomposition is unique, up to a scaling and permutation ambiguity. These conditions are shown in the sequel.

Uniqueness of tensor rank decomposition

We begin with several additional definitions.

Definition 4.9 *Let a matrix* $\mathbf{A} \in \mathbb{R}^{I \times R}$. *The kruskal rank* $k_{\mathbf{A}}$ *of* $\mathbf{A}$ *is the largest integer* k *such that any* k *columns of* $\mathbf{A}$ are linearly independent. It holds that $k_{\mathbf{A}} \leq rank(\mathbf{A})$ [4].

We can now give the definition of the scaling and permutation ambiguity.

Definition 4.10 *Given a tensor* $\boldsymbol{\mathcal{X}}$ *of rank* R, *we say that its CPD is unique, up to a scaling and permutation ambiguity, if the* R *rank–*1 *terms in its decomposition are unique.*

The above definition means that for a tensor $\boldsymbol{\mathcal{X}} = [\![\tilde{\mathbf{U}}^{(0)}, \ldots, \tilde{\mathbf{U}}^{(N)}]\!]$, with $\tilde{\mathbf{U}}^{(i)} \in \mathbb{R}^{I_i \times R}$ and $i \in \{1, \ldots, N\}$, there exists a permutation matrix $\boldsymbol{\Pi}$ and diagonal scaling matrices $\boldsymbol{\Lambda}_i$, with $i \in \{1, \ldots, N\}$, such that for all $i \in \{1, \ldots, N\}$

$$\tilde{\mathbf{U}}^{(i)} = \mathbf{U}^{(i)} \boldsymbol{\Pi} \boldsymbol{\Lambda}_i, \ \prod_{i=1}^{N} \boldsymbol{\Lambda}_i, = \mathbf{I}. \tag{4.8}$$

Now we can state the uniqueness condition, for any N–th order tensor, with $N \geq 3$.

Theorem 4.1 *Let* $\boldsymbol{\mathcal{X}} = [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!] = \sum_{r=1}^{R} \mathbf{u}_r^{(1)} \circ \cdots \circ \mathbf{u}_r^{(N)}$, *with* $\mathbf{U}^{(i)} \in \mathbb{R}^{I_i \times R}$, $i \in \{1, \ldots, N\}$. *If* $\sum_{i=1}^{N} k_{\mathbf{U}^{(i)}} \geq 2R + N - 1$, *then the decomposition is unique up to scaling and permutation ambiguity.*

Methods to find the CP decomposition of a tensor are discussed in the next chapter.

4.3.3 Applications

A sample application of matrix and/or tensor decompositions includes recommender systems. Streaming services, for example, may deploy collaborative filtering and factorization techniques in order to suggest new products to their users [4]. Another application described in [20] is the modelling of chatroom data using tensors. In this study, the bilinear SVD model is tested versus the PARAFAC and Tucker3 models. The Tucker3 decomposition is another tensor decomposition model, it is not, however, a rank decomposition. For more applications of tensor factorizations, we refer the reader to [21], [22] and [4].

Chapter 5

Tensor Factorizations

5.1 PARAFAC model

Let tensor $\boldsymbol{\mathcal{X}}^o \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ admit a factorization of the form

$$\boldsymbol{\mathcal{X}}^o = [\![\mathbf{U}^{o(1)}, \ldots, \mathbf{U}^{o(N)}]\!] = \sum_{r=1}^{R} \mathbf{u}_r^{o(1)} \circ \cdots \circ \mathbf{u}_r^{o(N)}, \tag{5.1}$$

where $\mathbf{U}^{o(i)} = [\mathbf{u}_1^{o(i)} \ \cdots \ \mathbf{u}_R^{o(i)}] \in \mathbb{R}^{I_i \times R}$, with $i \in \{1, \ldots, N\}$. We observe the noisy tensor $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}}^o + \boldsymbol{\mathcal{E}}$, where $\boldsymbol{\mathcal{E}}$ is the additive noise. Then, estimates of $\mathbf{U}^{o(i)}$ can be obtained by computing matrices $\mathbf{U}^{(i)} \in \mathbb{R}^{I_i \times R}$, for $i \in \{1, \ldots, N\}$, that solve the optimization problem

$$\min_{\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}} f_{\boldsymbol{\mathcal{X}}} \left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)} \right), \tag{5.2}$$

where $f_{\boldsymbol{\mathcal{X}}}$ is a function measuring the quality of the factorization. A common choice for $f_{\boldsymbol{\mathcal{X}}}$ is

$$f_{\boldsymbol{\mathcal{X}}} \left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)} \right) = \frac{1}{2} \left\| \boldsymbol{\mathcal{X}} - [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!] \right\|_F^2. \tag{5.3}$$

If $\boldsymbol{\mathcal{Y}} = [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]$, then for an arbitrary mode i, the corresponding matrix unfolding is given by [1]

$$\begin{aligned} \mathbf{Y}_{(i)} &= \mathbf{U}^{(i)} \left(\mathbf{U}^{(N)} \odot \cdots \odot \mathbf{U}^{(i+1)} \odot \mathbf{U}^{(i-1)} \odot \cdots \odot \mathbf{U}^{(1)} \right)^T, \\ \mathbf{Y}_{(i)} &= \mathbf{U}^{(i)} \mathbf{K}^{i^T}, \end{aligned} \tag{5.4}$$

where we define $\mathbf{K}^i$ as

$$\mathbf{K}^i = \left(\bigodot_{j=i+1}^{N} \mathbf{U}^{(j)} \odot \bigodot_{j=1}^{i-1} \mathbf{U}^{(j)} \right). \tag{5.5}$$

Thus, $f_{\boldsymbol{\mathcal{X}}}$ can be expressed as

$$f_{\boldsymbol{\mathcal{X}}}(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}) = \frac{1}{2} \left\| \mathbf{X}_{(i)} - \mathbf{Y}_{(i)} \right\|_F^2 \tag{5.6}$$

These expressions form the basis of the alternating least squares algorithm (ALS) for tensor factorization, in the sense that, for fixed matrix factors $\mathbf{U}^{(j)}$, with $j \neq i$, we can update $\mathbf{U}^{(i)}$ by solving a matrix least–squares problem.

5.2 Dimension Trees

As we mentioned above, for fixed matrix factors $\mathbf{U}^{(j)}$, with $j \neq i$, updating $\mathbf{U}^{(i)}$ can be reduced to the solution of a matrix least–squares problem and by extension to the solution of the normal equations, which for the PARAFAC model are given by

$$\begin{aligned} &\mathbf{Y}_{(i)}\mathbf{K}^i = \mathbf{U}^{(i)} \left(\mathbf{K}^i\right)^T \mathbf{K}^i \Leftrightarrow \\ &\mathbf{Y}_{(i)}\mathbf{K}^i = \mathbf{U}^{(i)} \left(\circledast_{\substack{j=1 \\ j\neq i}}^{N} \mathbf{U}^{(j)T}\mathbf{U}^{(j)} \right) \Leftrightarrow \\ &\mathbf{Y}_{(i)}\mathbf{K}^i = \mathbf{U}^{(i)}\mathbf{H}^{(i)}, \end{aligned} \tag{5.7}$$

where $\left(\circledast_{\substack{j=1 \\ j\neq i}}^{N} \mathbf{U}^{(j)T}\mathbf{U}^{(j)} \right) = \mathbf{H}^{(i)}$ The left–hand term of (5.7) is referred to as matricized tensor times Khatri–Rao product (**MTTKRP**). For the PARAFAC decomposition, under the ALS framework, the calculation of MTTKRP constitutes the computational bottleneck and finding ways to reduce the computational complexity of MTTKRP has attracted the interest of many researchers.

Dimension trees are data structures that aim to avoid recalculating expressions that are common among computations of different MTTKRPs, during a full cycle of factor updates (ALS iteration). We introduce these expressions by initially examining how they are used in the case of a third–order tensor, and then of a fourth–order tensor. Based on the analysis for the fourth–order tensor, this approach can be easily generalized for the case of N–th order tensors, where $N > 4$.

Third–Order Tensor Case

Let us consider a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I\times J\times K}$, and let $\mathbf{U}^{(1)} \in \mathbb{R}^{I\times R}$, $\mathbf{U}^{(2)} \in \mathbb{R}^{J\times R}$, and $\mathbf{U}^{(3)} \in \mathbb{R}^{K\times R}$ be the approximating factors. Then, the MTTKRPs that correspond to factors $\mathbf{U}^{(1)}$

and $\mathbf{U}^{(2)}$, respectively, are given by

$$\mathbf{M}^{(1)} = \underline{\mathbf{X}_{(1)}(\mathbf{U}^{(3)}} \odot \mathbf{U}^{(2)}),$$
$$\mathbf{M}^{(2)} = \underline{\mathbf{X}_{(2)}(\mathbf{U}^{(3)}} \odot \mathbf{U}^{(1)}).$$

In the sequel, we show that the underlined temporary quantities correspond to a common term, which is actually a tensor. Each element of matrix $\mathbf{M}^{(1)}$ can be computed as

$$m^{(1)}_{i,r} = \sum_{j,k} x_{i,j,k} u^{(2)}_{j,r} u^{(3)}_{k,r} = \sum_{j} u^{(2)}_{j,r} \sum_{k} x_{i,j,k} u^{(3)}_{k,r} = \sum_{j} u^{(2)}_{j,r} t_{i,j,r},$$

where $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{I \times J \times R}$ is a tensor that can be reused for the computation of $\mathbf{M}^{(2)}$, since

$$m^{(2)}_{j,r} = \sum_{i,k} x_{i,j,k} u^{(1)}_{i,r} u^{(3)}_{k,r} = \sum_{i} u^{(1)}_{i,r} \sum_{k} x_{i,j,k} u^{(3)}_{k,r} = \sum_{i} u^{(1)}_{i,r} t_{i,j,r}.$$

However, calculating the MTTKRP that corresponds to the factor $\mathbf{U}^{(3)}$ has to be computed from ground.

Fourth–Order Tensor Case

Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K \times L}$ and $\mathbf{U}^{(1)} \in \mathbb{R}^{I \times R}$, $\mathbf{U}^{(2)} \in \mathbb{R}^{J \times R}$, $\mathbf{U}^{(3)} \in \mathbb{R}^{K \times R}$, and $\mathbf{U}^{(4)} \in \mathbb{R}^{L \times R}$ be the approximating factors. Then, the corresponding MTTKRPs will be

$$\mathbf{M}^{(1)} = \underline{\mathbf{X}^{(1)}(\mathbf{U}^{(4)} \odot \mathbf{U}^{(3)}} \odot \mathbf{U}^{(2)}), \tag{5.8}$$

$$\mathbf{M}^{(2)} = \underline{\mathbf{X}^{(2)}(\mathbf{U}^{(4)} \odot \mathbf{U}^{(3)}} \odot \mathbf{U}^{(1)}), \tag{5.9}$$

$$\mathbf{M}^{(3)} = \underline{\mathbf{X}^{(3)}}(\mathbf{U}^{(4)} \odot \underline{\mathbf{U}^{(2)} \odot \mathbf{U}^{(1)}}), \tag{5.10}$$

$$\mathbf{M}^{(4)} = \underline{\mathbf{X}^{(4)}}(\mathbf{U}^{(3)} \odot \underline{\mathbf{U}^{(2)} \odot \mathbf{U}^{(1)}}). \tag{5.11}$$

Working analogously as above, we can obtain

$$m^{(1)}_{i,r} = \sum_{j,k,l} x_{i,j,k,l} u^{(2)}_{j,r} u^{(3)}_{k,r} u^{(4)}_{l,r} = \sum_{j} u^{(2)}_{j,r} r^{1}_{i,j,r}, \tag{5.12}$$

$$m^{(2)}_{j,r} = \sum_{i,k,l} x_{i,j,k,l} u^{(1)}_{i,r} u^{(3)}_{k,r} u^{(4)}_{l,r} = \sum_{i} u^{(1)}_{i,r} r^{1}_{i,j,r}, \tag{5.13}$$

$$m^{(3)}_{k,r} = \sum_{i,j,l} x_{i,j,k,l} u^{(2)}_{j,r} u^{(1)}_{i,r} u^{(4)}_{l,r} = \sum_{l} u^{(4)}_{l,r} r^{2}_{k,l,r}, \tag{5.14}$$

$$m_{l,r}^{(4)} = \sum_{i,j,k} x_{i,j,k,l} u_{j,r}^{(2)} u_{i,r}^{(1)} u_{k,r}^{(3)} = \sum_{k} u_{k,r}^{(3)} r_{k,l,r}^{2}, \tag{5.15}$$

where $\boldsymbol{\mathcal{R}}^1 \in \mathbb{R}^{I \times J \times R}$ is the result of R TTV products between $\boldsymbol{\mathcal{T}}^1_{:,:,:,r}$ and $\mathbf{U}^{(3)}_{:,r}$, for $r \in \{1, \ldots, R\}$, while $\boldsymbol{\mathcal{R}}^2 \in \mathbb{R}^{K \times L \times R}$ is defined in a similar way.

Computing MTTKRP through tensor operations

We can compute the MTTKRP by using TTVs [23]. The i–th factor's MTTKRP $\mathbf{M}^{(i)}$ is computed column–by–column. The r–th column, for $r \in \{1, \ldots, R\}$, is a result of $(N-1)$ TTVs:

$$\mathbf{M}^{(n)}_{:,r} = \boldsymbol{\mathcal{X}} \times_{i \neq n} \mathbf{u}^{(i)}_r, \tag{5.16}$$

where $\mathbf{u}^{(i)}_r$ is the r–th column of $\mathbf{U}^{(i)}$. Equation (5.16) is written equivalently as

$$\mathbf{M}^{(n)}_{:,r} = \boldsymbol{\mathcal{X}} \times_1 \mathbf{u}^{(1)}_r \times_2 \cdots \times_{n-1} \mathbf{u}^{(n-1)}_r \times_{n+1} \mathbf{u}^{(n+1)}_r \times_{n+2} \cdots \times_N \mathbf{u}^{(N)}_r. \tag{5.17}$$

5.2.1 Dimension Tree Based ALS

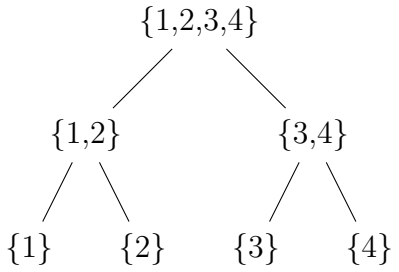

Figure 5.1: Dimension tree for a Tensor of order 4. In general, each node is associated with a tensor, and a set of indices $\mathcal{S} \subseteq \{1, 2, \ldots, order(\boldsymbol{\mathcal{X}})\}$, which give us information regarding the dimensions of the node's tensor.

As presented in the previous cases, in each ALS iteration different MTTKRPs share common terms, the partial MTTKRPs, that can be calculated once and then be saved to convenient data structures. One popular choice for the needs of the ALS–based PARAFAC decomposition consists in the deployment of a specific kind of binary trees known as dimension trees. As we show in chapter 6, this strategy can offer significant speedup in computing tensor decompositions.

Constructing a Dimension Tree

Given a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, a dimension tree follows the structure of a binary tree, where each node of the tree is associated with a tensor and is labeled by a set of indices. The cardinality of this set defines the order of the associated tensor, while the i–th index defines the i–th dimension of the associated tensor, in correspondence to the dimensions of the initial tensor $\boldsymbol{\mathcal{X}}$. Specifically, regarding the root node, the associated tensor will be $\boldsymbol{\mathcal{X}}$ and will be labeled by the set $\{1, \ldots, N\}$, while the nonroot nodes will be associated with tensors of order $\leq N$, with the last dimension being equal to the decomposition rank R, and labeled with a subset of $\{1, \ldots, N\}$. This specification of the nonroot nodes emerges from the fact that the associated tensors, in the case of nonroot nodes, are produced by partial MTTKRP operations. The reader should notice at this point that, for the case of the leaf nodes, the associated tensor will have a maximum order of 2, in the case where $R > 1$; then these nodes will be associated with a matrix, which is one of the MTTKRP terms $\mathbf{M}^{(n)}$, with $n \in \{1, \ldots, N\}$.

Next, we demonstrate an example of a dimension tree for the case of a fourth–order tensor. Let us assume that we are interested in factorizing a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K \times L}$. Then, a dimension tree that could be used along with the ALS algorithm is presented in Figure 5.1. In accordance to the relations in (3.18), nodes $\{1, 2\}$ and $\{3, 4\}$ will be associated to tensors $\boldsymbol{\mathcal{R}}^1$ and $\boldsymbol{\mathcal{R}}^2$, respectively, while nodes $\{1\}$, $\{2\}$, $\{3\}$, and $\{4\}$ will be associated to matrices $\mathbf{M}^1$, $\mathbf{M}^2$, $\mathbf{M}^3$, and $\mathbf{M}^4$, respectively.

Updating Procedure of a Dimension Tree

In this section, we rely on the notation and discuss the algorithms that are presented in [24]. Assume that factor $\mathbf{U}^{(i)}$ has been successfully updated after taking into consideration the associated matrix of leaf $\{i\}$. It should be clear that several contents of the dimension tree are no more up to date and proceeding with the course of the ALS algorithm could lead to inaccurate results. Hence, after updating a matrix factor, the need of updating efficiently the dimension tree arises. In order to describe the procedures one should perform to achieve this, we introduce some useful notation. A dimension tree generated by a tensor $\boldsymbol{\mathcal{X}}$ is denoted in the sequel as $\mathcal{T}(\boldsymbol{\mathcal{X}})$. The leaf node labeled by $\{i\}$ is denoted as l_i. For a node t, we denote the set of indices node t has as $\mu(t)$, while we define $\mu'(t)$ to be the complement of $\mu(t)$ with respect to $\mu(root)$, namely $\mu'(t) = \{1, \ldots, N\} \setminus \mu(t)$. The parent of node t is denoted as $\mathcal{P}(t)$, while the left and right children nodes are denoted as $\mathcal{L}(t)$ and $\mathcal{R}(t)$, respectively. The associated to node t tensor is denoted by $\boldsymbol{\mathcal{X}}^{(t)}$. At last, we let $\delta(t)$ to be the set of indices that the sibling node of t has, namely, $\delta(t) = \mu'(t) \setminus \mu'(\mathcal{P}(t)) =$

$\mu(\mathcal{P}(t)) \setminus \mu(t)$.

Updating a node t requires the parent node $\mathcal{P}(t)$ to be updated. Hence, in order to update node t, all the nodes that belong to the path connecting t and the *root* node have to be traversed, beginning from node t, until an updated node is met. Note that the existence of an updated node in every ending to the *root* path is guaranteed, since the content of the *root* node does not change during the execution of the ALS algorithm and therefore can be always considered as updated. As a result, updating node t can be achieved in a recursive fashion. Algorithms 3 and 4, presented on [24], are devised within this framework for the problem of efficient updating a dimension tree for the needs of PARAFAC decomposition via ALS optimization.

Dimension Tree based ALS

Algorithm 3 is a recursive algorithm that takes as input argument a node $t \in \mathcal{T}(\boldsymbol{\mathcal{X}})$, deals with the update of all nodes on the path connecting node t and the *root* node and returns an updated version of $\boldsymbol{\mathcal{X}}^{(t)}$. From the perspective of [24], a node t' is assumed to be updated, when tensor $\boldsymbol{\mathcal{X}}^{(t')}$ exists, since when node t' becomes outdated. Algorithm 4 proceeds to set $\boldsymbol{\mathcal{X}}^{(t')}$ as outdated. Algorithm 4 can be characterized as an customized version of vanilla ALS–PARAFAC, where the interaction of the dimension tree is supported by functions *Construct_Dimension_Tree*, *Destroy*, and *Dtree_TTV*.

Before introducing the algorithms, we present additional notation. We refer to $\boldsymbol{\mathcal{X}}_r^{(t)}$, as a subtensor of $\boldsymbol{\mathcal{X}}^{(t)}$, where the last index is fixed at value r, with $r \in \{1, \ldots, R\}$. For the root node *root*, we have that $\boldsymbol{\mathcal{X}}_r^{(root)} = \boldsymbol{\mathcal{X}}$, for all r.

Algorithm 3 DTree_TTV: Dimension tree–based TTV with R vectors in each mode

procedure DTREE_TTV(Node $t \in \mathcal{T}(\boldsymbol{\mathcal{X}})$)
  if exists($\boldsymbol{\mathcal{X}}^{(t)}$) **then**
    return $\boldsymbol{\mathcal{X}}^{(t)}$
  end if
  $\boldsymbol{\mathcal{X}}^{\mathcal{P}(t)} =$ DTree_TTV($\mathcal{P}(t)$)
  for $r = 1, \ldots, R$ **do**
    $\boldsymbol{\mathcal{X}}_r^{(t)} = \boldsymbol{\mathcal{X}}_r^{(\mathcal{P}(t))} \times_{d \in \delta(t)} \mathbf{u}_r^{(d)}$
  end for
  return $\boldsymbol{\mathcal{X}}^{(t)}$
end procedure

Algorithm 4 DTree–PARAFAC–ALS: A dimension tree–based PARAFAC–ALS algorithm, for unconstrained and NTF problems.

procedure DTree–ALS($\boldsymbol{\mathcal{X}}$,R,constraint_flag[ ])
  $k = 0$
  $\mathcal{T}(\boldsymbol{\mathcal{X}})$ = Construct_Dimension_Tree($\boldsymbol{\mathcal{X}}$)
  for $i = 1, \ldots, N$ **do**
    $\mathbf{W}^{(i)} = \mathbf{U}_k^{(i)T}\mathbf{U}_k^{(i)}$
  end for
  repeat
    for $i = 1, \ldots, N$ **do**
      for $t \in \mathcal{T}(\boldsymbol{\mathcal{X}})$ **do**
        if $i \in \mu'(t)$ **then**
          Set_outdated($\boldsymbol{\mathcal{X}}^{(t)}$)
        end if
      end for
      $\mathbf{M}^{(i)}$ = DTree_TTV(l_i)
      $\mathbf{H}^{(i)} = \circledast_{\substack{j=1 \\ j\neq i}}^{N} \mathbf{W}^{(i)}$
      $\mathbf{U}_{k+1}^{(i)}$ = Update_Factor($\mathbf{M}^{(i)}, \mathbf{H}^{(i)}, \mathbf{U}_k^{(i)}$, constraint_flag[i])
      $\mathbf{W}^{(i)} = \left(\mathbf{U}_{k+1}^{(i)}\right)^T \mathbf{U}_{k+1}^{(i)}$
    end for
    $k = k + 1$
  until convergence is achieved or maximum number of iterations has been reached
  return $[\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]$
end procedure

Algorithm 5 The Update_Factor function

procedure Update_Factor($\mathbf{M}^{(i)}$,$\mathbf{H}^{(i)}$,$\mathbf{U}_k^{(i)}$,constraint_flag)
  if constraint_flag = 'unconstrained' **then**
    $\mathbf{U}_{k+1}^{(i)} = \mathbf{M}^{(i)}\mathbf{H}^{(i)\dagger}$
  else if constraint_flag = 'nonnegativity' **then**
    $\mathbf{U}_{k+1}^{(i)}$ = Nesterov_MNLS($\mathbf{M}^{(i)}, \mathbf{H}^{(i)}, \mathbf{U}_k^{(i)}$)
  end if
  return $\mathbf{U}_{k+1}^{(i)}$
end procedure

5.3 Constrained Tensor Decomposition

In many applications, we are interested in tensor decompositions where the requested factors are desired to comply with constraints emerging from underlying models or for

interpretability reasons. Specifically, let tensor $\boldsymbol{\mathcal{X}}^o \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ admit a factorization of the form

$$\boldsymbol{\mathcal{X}}^o = [\![\mathbf{U}^{(1)o}, \ldots, \mathbf{U}^{(N)o}]\!] = \sum_{r=1}^{R} \mathbf{u}_r^{(1)o} \circ \cdots \circ \mathbf{u}_r^{(N)o}, \tag{5.18}$$

where $\mathbf{U}^{(n)o} = \left[\mathbf{u}_1^{(n)o} \ \cdots \ \mathbf{u}_R^{(n)o}\right] \in \mathbb{B}^n \subseteq \mathbb{R}^{I_n \times R}$, with $n \in \{1, \ldots, N\}$. We observe the noisy tensor $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}}^o + \boldsymbol{\mathcal{E}}$, where $\boldsymbol{\mathcal{E}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is the additive noise. Then, the problem of finding estimates of the factors $\mathbf{U}^{(n)o}$ can be formulated as

$$\begin{aligned} &\min_{\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(n)}} f_{\boldsymbol{\mathcal{X}}}\left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}\right) \\ &\quad s.t. \quad \mathbf{U}^{(n)} \in \mathbb{B}^n, \quad n \in \{1, \ldots, N\} \end{aligned} \tag{5.19}$$

where $f_{\boldsymbol{\mathcal{X}}}$ is a function measuring the quality of the factorization. As in the unconstrained case, we focus on the sum of squared errors cost function

$$f_{\boldsymbol{\mathcal{X}}}\left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}\right) = \frac{1}{2} \left\| \boldsymbol{\mathcal{X}} - [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!] \right\|_F^2. \tag{5.20}$$

Under the ALS framework, each factor can be updated via solving an unconstrained/constrained matrix least–squares problem. It can be formulated as:

$$\begin{aligned} \min_{\mathbf{U}^{(i)}} \quad & \frac{1}{2} \|\mathbf{X}_{(i)} - \mathbf{U}^{(i)} \mathbf{K}^i\|_F^2 \\ s.t. \quad & \mathbf{U}^{(i)} \in \mathbb{B}^i, \quad i \in \{1, \ldots, N\}. \end{aligned} \tag{5.21}$$

Set $\mathbb{B}^i$, for $i \in \{1, \ldots, N\}$, can be

- $\mathbb{R}^{I_i \times R}$: unconstrained case,
- $\mathbb{R}_+^{I_i \times R}$: case of nonnegativity constraints.

Tensor Decomposition with nonnegativity constraints

Recall expressions (5.2), (5.3) and (5.6):

$$\begin{aligned} &\min_{\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}} f_{\boldsymbol{\mathcal{X}}}\left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}\right), \\ f_{\boldsymbol{\mathcal{X}}}\left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}\right) &= \frac{1}{2} \left\| \boldsymbol{\mathcal{X}} - [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!] \right\|_F^2, \\ f_{\boldsymbol{\mathcal{X}}}(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}) &= \frac{1}{2} \left\| \mathbf{X}_{(i)} - \mathbf{Y}_{(i)} \right\|_F^2. \end{aligned}$$

If we introduce the constraint $\{\mathbf{U}^{(i)}\}_{i=1}^{N} \geq \mathbf{0}$, then these expressions form the basis for the AO NTF in the sense that, if we fix all but one of the matrix factors, we can update the remaining factor by solving an MNLS problem.

For later use, we note that the most demanding computations during the update of any factor matrix $\mathbf{U}_k^{(i)}$ via the Nesterov–type MNLS algorithm are (see lines 4 and 5 of Algorithm 2)

$$\widetilde{\mathbf{M}}_{(i)} := \mathbf{X}_{(i)} \left(\bigodot_{j=i+1}^{N} \mathbf{U}_k^{(j)} \odot \bigodot_{j=1}^{i-1} \mathbf{U}_{k+1}^{(j)} \right) = \mathbf{X}_{(i)} \mathbf{K}_k^i, \tag{5.22}$$

$$\widetilde{\mathbf{Z}}_{(i)} := \mathbf{K}_k^{i^T} \mathbf{K}_k^i = \left(\circledast_{j=i+1}^{N} \left(\mathbf{U}_k^{(j)^T} \mathbf{U}_k^{(j)} \right) \right) \circledast \left(\circledast_{j=1}^{i-1} \left(\mathbf{U}_{k+1}^{(j)^T} \mathbf{U}_{k+1}^{(j)} \right) \right), \tag{5.23}$$

where $\mathbf{K}_k^i$ is defined as:

$$\mathbf{K}_k^i = \left(\bigodot_{j=i+1}^{N} \mathbf{U}_k^{(j)} \odot \bigodot_{j=1}^{i-1} \mathbf{U}_{k+1}^{(j)} \right). \tag{5.24}$$

It has been shown in [25] that the AO NTF algorithm with proximal term falls under the block successive upper bound minimization (BSUM) framework, which ensures convergence to a stationary point of problem (5.2).

Chapter 6

Parallel Implementations

In this section, we assume that we have at our disposal $p = \prod_{i=1}^{N} p_i$ processing elements and describe a parallel implementation of the AO algorithms, which has been motivated by the medium–grained approach of [11].[1] The p processors form an n-dimensional Cartesian grid and are denoted as $p_{i_1,\ldots,i_N}$, with $i_j \in \{1,\ldots,p_j\}$ and $j \in \{1,\ldots,N\}$.

6.1 Variable partitioning and data allocation

In order to describe the parallel implementation, given a decomposition problem of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ into a set of factors $\mathbf{U}^{(i)} \in \mathbb{R}^{I_i \times R}$, for all $i \in \{1,\ldots,N\}$, we introduce certain partitionings of the factor matrices and tensor $\boldsymbol{\mathcal{X}}$. Specifically, we partition each factor matrix $\mathbf{U}^{(i)}$ into p_i block rows as

$$\mathbf{U}^{(i)} = \left[\begin{array}{ccc} \left(\mathbf{U}^{(i)^1}\right)^T & \cdots & \left(\mathbf{U}^{(i)^{p_i}}\right)^T \end{array} \right]^T, \tag{6.1}$$

with $\mathbf{U}^{(i)^j} \in \mathbb{R}^{\frac{I_i}{p_i} \times R}$, for all $j \in \{1,\ldots,p_i\}$. Additionally, we partition tensor $\boldsymbol{\mathcal{X}}$ into p subtensors as

$$\boldsymbol{\mathcal{X}}^{i_1,\ldots,i_N} = \boldsymbol{\mathcal{X}}\left((i_1-1)\frac{I_1}{p_1}+1 : i_1\frac{I_1}{p_1}, \ldots, (i_N-1)\frac{I_N}{p_N}+1 : i_N\frac{I_N}{p_N}\right) \in \mathbb{R}^{\frac{I_1}{p_1}\times\cdots\times\frac{I_N}{p_N}} \tag{6.2}$$

where $i_j \in \{1,\ldots,p_j\}$ and $j \in \{1,\ldots,N\}$.

From the perspective of the processors, processor $p_{i_1,\ldots,i_N}$ receives subtensor $\boldsymbol{\mathcal{X}}^{i_1,\ldots,i_N}$ and contributes into updating the i_j–th part of factor $\mathbf{U}^{(j)}$, $\mathbf{U}^{(j)^{i_j}}$, for all $j \in \{1,\ldots,N\}$. At last, we assume that, at the end of the k–th outer AO iteration,

(a) processor $p_{i_1,\ldots,i_N}$ knows $\mathbf{U}_k^{(1)^{i_1}}$, $\mathbf{U}_k^{(2)^{i_2}}$, …, $\mathbf{U}_k^{(N)^{i_N}}$

(b) *all* processors know $\left(\mathbf{U}_k^{(i)}\right)^T \mathbf{U}_k^{(i)}$, for all $i \in \{1,\ldots,N\}$.

[1] We note that both the single–core and the multi–core implementations solve the same problem, thus problems that are identifiable in single-core environments remain identifiable in multi–core environments and the solutions, in both cases, are practically the same.

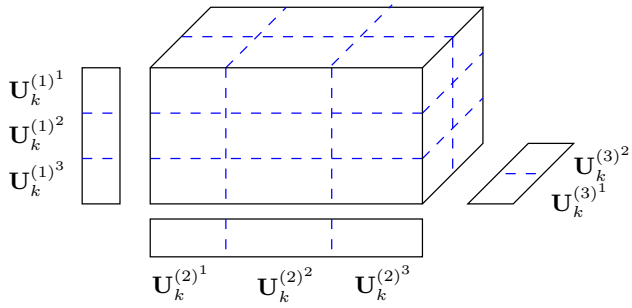

Figure 6.1: Tensor $\boldsymbol{\mathcal{X}}$, factors $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$, and $\mathbf{U}^{(3)}$, and their partitioning for $p_1 = p_2 = 3$ and $p_3 = 2$.

6.2 Communication groups

We define certain communication groups, also known as communicators [26], over subsets of the p processors, which are used for the efficient collaborative implementation of specific computational tasks, as explained in detail below.

First, we define as $\mathcal{C}_{i,j}$, for $i \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, p_i\}$, to be the $(N-1)$–dimensional group of processors, involving the $\prod_{k=1, k\neq i}^{N} p_k$ processors having the i–th index equal to j, which are used for the collaborative update of $\mathbf{U}_k^{(i)^j}$. Additionally, we define as $\mathcal{D}_{i,\mathcal{J}^i}$, for $i \in \{1, \ldots, N\}$ and $\mathcal{J}^i \in \bigtimes_{n=1, n\neq i}^{N} \mathcal{S}^n$, with $\mathcal{S}^n = \{1, \ldots, p_n\}$, to be the one–dimensional processor groups, each involving the p_i processors that differ only at the i–th index. Each of these groups is used for the collaborative computation of $\left(\mathbf{U}_{k+1}^{(j)}\right)^T \mathbf{U}_{k+1}^{(j)}$, for $j \in \{1, \ldots, N\}$.

6.3 Parallel implementation

6.3.1 Factor update implementation

In this section, we consider the case of updating the factor matrix $\mathbf{U}_k^{(1)}$. In order to facilitate our analysis, we introduce the following partitioning of the mode–1 matricization of the tensor $\boldsymbol{\mathcal{X}}$, as

$$\mathbf{X}_{(1)} = \left[\begin{array}{ccc} \left(\mathbf{X}_{(1)}^1\right)^T & \cdots & \left(\mathbf{X}_{(1)}^{p_1}\right)^T \end{array} \right]^T, \tag{6.3}$$

with $\mathbf{X}_{(1)}^j \in \mathbb{R}^{\frac{I_1}{p_1} \times \prod_{n=2}^{N} I_n}$, for all $j \in \{1, \ldots, p_1\}$. We describe in detail the update of $\mathbf{U}_k^{(1)}$, which is achieved via the parallel updates of $\mathbf{U}_k^{(1)^{i_1}}$, for all $i_1 \in \{1, \ldots, p_1\}$, and consists of

the following stages:

1. Processors in $\mathcal{C}_{1,i_1}$, for all $i_1 \in \{1,\ldots,p_1\}$, collaboratively compute the $\frac{I_1}{p_1} \times R$ matrix

$$\mathbf{M}^{(1)^{i_1}} = \mathbf{X}_{(1)}^{i_1}\mathbf{K}^{(1)}, \tag{6.4}$$

and the result is scattered among the processors in the group; thus, each processor in the group receives $\frac{I}{\prod_{i=1}^{N} p_i}$ successive rows of $\mathbf{M}^{(1)^{i_1}}$. Term $\mathbf{M}^{(1)^{i_1}}$ can be computed collaboratively because

$$\mathbf{X}_{(1)}^{i_1}\mathbf{K}^{(1)} = \sum_{i_2=1}^{p_2} \cdots \sum_{i_N=1}^{p_N} \mathbf{X}_{(1)}^{i_1,\ldots,i_N}(\mathbf{U}_k^{(N)^{i_N}} \odot \cdots \odot \mathbf{U}_k^{(2)^{i_2}}), \tag{6.5}$$

where $\mathbf{X}_{(1)}^{i_1,\ldots,i_N}$ is the matricization of $\boldsymbol{\mathcal{X}}^{i_1,\ldots,i_N}$ with respect to the first mode. Each processor $p_{i_1,\ldots,i_n}$ in $\mathcal{C}_{1,i_1}$ knows $\mathbf{X}_{(1)}^{i_1,\ldots,i_N}$, $\mathbf{U}_k^{(2)^{i_2}}$, $\ldots$, $\mathbf{U}_k^{(N)^{i_N}}$ and computes the corresponding term of (6.5). The sum is computed and scattered among all processors in $\mathcal{C}_{1,i_1}$ via a reduce–scatter operation.

2. Each processor in the group $\mathcal{C}_{1,i_1}$ uses the scattered part of $\mathbf{M}^{(1)^{i_1}}$, matrix $\mathbf{K}^{(1)}$ and partial factor $\mathbf{U}_k^{(1)^{i_1}}$, in order to compute the updated part of $\mathbf{U}_{k+1}^{(1)^{i_1}}$, via the update factor algorithm.

3. The updated parts of $\mathbf{U}_{k+1}^{(1)^{i_1}}$ are all–gathered at the processors of the group $\mathcal{C}_{1,i_1}$, so that *all* processors in the group learn the updated factor $\mathbf{U}_{k+1}^{(1)^{i_1}}$.

4. By applying an all–reduce operation to $\left(\mathbf{U}_{k+1}^{(1)^{i_1}}\right)^T \mathbf{U}_{k+1}^{(1)^{i_1}}$, for all $i_1 \in \{1,\ldots,p_1\}$, on each of the single–dimensional processor groups $\mathcal{D}_{1,\mathcal{J}^1}$, *all* p processors learn $\mathbf{U}_{k+1}^{(1)^T}\mathbf{U}_{k+1}^{(1)}$.[2]

As for the rest of the factors, the updates can be implemented by following analogous steps.

6.3.2 Communication cost

We focus on the parallel updates of $\mathbf{U}_k^{(1)^{i_1}}$, for all $i_1 \in \{1,\ldots,p_1\}$, and present results concerning the associated communication cost. Analogous results hold for the updates of

[2]In the cases where $R \gtrapprox \frac{I_1}{p_1}$ it seems preferable to compute $\left(\mathbf{U}_{k+1}^{(1)}\right)^T \left(\mathbf{U}_{k+1}^{(1)}\right)$ via an all–gather operation on terms $\mathbf{U}_{k+1}^{(1)^{i_1}}$, for all $i_1 \in \{1,\ldots,p_1\}$, on each of the single–dimensional processor groups $\mathcal{D}_{1,\mathcal{J}^1}$. However, in this work, we mainly focus on small–rank factorizations, thus, in our communication cost analysis and experiments we do not present results for this alternative.

the remaining factor matrices $\mathbf{U}_k^{(j)^{i_j}}$, for all $j \in \{2, \ldots, N\}$ and $i_j \in \{1, \ldots, p_j\}$.

We assume that an m–word message is transferred from one process to another with communication cost $t_s + t_w m$, where t_s is the latency, or startup time for the data transfer, and t_w is the word transfer time [26].

Communication occurs at three algorithm execution points.

1. The $\frac{I_1}{p_1} \times R$ matrix $\mathbf{M}^{1^{i_1}}$ is computed and scattered among the $\prod_{j=2}^{N} p_j$ processors of group $\mathcal{C}_{1,i_1}$, using a reduce–scatter operation, with communication cost [26, §4.2]

$$\mathcal{C}ost_1^{\mathbf{1}} = t_s \left(\sum_{j=2}^{N} p_j - (N-1) \right) + t_w \frac{I_1 R}{p} \left(\prod_{j=2}^{N} p_j - 1 \right).$$

2. Processors in $\mathcal{C}_{1,i_1}$ learn the updated $\mathbf{U}_{k+1}^{(1)^{i_1}}$ through an all–gather operation on its updated parts, each of dimension $\frac{I}{p} \times R$, with communication cost [26, §4.2]

$$\mathcal{C}ost_2^{\mathbf{1}} = t_s \left(\sum_{j=2}^{N} p_j - (N-1) \right) + t_w \frac{I_1 R}{p} \left(\prod_{j=2}^{N} p_j - 1 \right).$$

3. Finally, $\left(\mathbf{U}_{k+1}^{(1)}\right)^T \mathbf{U}_{k+1}^{(1)}$ is computed by using an all–reduce operation on quantities $\left(\mathbf{U}_{k+1}^{(1)^{i_1}}\right)^T \mathbf{U}_{k+1}^{(1)^{i_1}}$, for all $i_1 \in \{1, \ldots, p_1\}$, on each single–dimensional processor group $\mathcal{D}_{1,\mathcal{J}^1}$, with communication cost [26, §4.3]

$$\mathcal{C}ost_3^{\mathbf{1}} = \left(t_s + t_w R^2\right) \log_2 p_1. \tag{6.6}$$

The communication that takes place during the acceleration step involves scalar quantities and, thus, is ignored.

When we are dealing with large messages, the t_w terms dominate the communication cost. Thus, if we ignore the startup time, the total communication time is

$$\begin{aligned} \mathcal{C}^{\mathbf{1}} &= t_w \left(\frac{2 I_1 R}{p} \left(\prod_{j=2}^{N} p_j - 1 \right) + R^2 \log_2 p_1 \right) \\ &\approx t_w \left(\frac{2 I_1 R}{p_1} + R^2 \log_2 p_1 \right) \\ &\approx \frac{2 I_1 R\, t_w}{p_1}, \end{aligned} \tag{6.7}$$

with the second approximation being accurate for $R \ll \frac{I_1}{p_1}$. The presence of p_1 in the denominator of the last expression of (6.7) implies that our implementation is scalable in the sense that, if we double I_1, then we can have (approximately) the same communication cost per processor by doubling p_1. Again, analogous results hold for the updates of the remaining factor matrices $\mathbf{U}_k^{(j)^{i_j}}$, for all $j = \{2, \ldots, N\}$ and $i_j \in \{1, \ldots, p_j\}$.

Chapter 7

Simulations

7.1 Parallel environment–MPI

We now present results obtained from the MPI implementation described in detail in Section 6.3. The program is executed on a DELL PowerEdge R820 system with SandyBridge–Intel(R) Xeon(R) CPU $E5-4650v2$ (in total, 16 nodes with 40 cores each at 2.4 GHz) and 490 GB RAM per node. The matrix and tensor operations are implemented using a combination of routines of the C++ library Eigen's Matrix module, as well as Eigen's unsupported tensor module. [27]. For the NTF problem, we assume a noiseless tensor $\boldsymbol{\mathcal{X}}$, whose true latent factors have i.i.d elements, uniformly distributed in $[0, 1]$, while for the unconstrained problem, its elements are i.i.d, uniformly distributed in $[-1, 1]$. The terminating conditions for MNLS are determined by values $\delta_1 = \delta_2 = 10^{-2}$. In addition, we set the limit for the number of AO iterations to 10, and for the Nesterov algorithm iterations to 50.

The AO algorithm terminates at iteration k if (recall that tensor $\boldsymbol{\mathcal{X}}$ is noiseless)

$$\texttt{RFE}(\mathbf{U}_k^{(1)}, \mathbf{U}_k^{(2)}, \ldots, \mathbf{U}_k^{(N)}) < 10^{-3}, \tag{7.1}$$

where

$$\texttt{RFE}(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}) := \frac{\left\|\boldsymbol{\mathcal{X}} - [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!]\right\|_F}{\|\boldsymbol{\mathcal{X}}\|_F} \tag{7.2}$$

We test the behavior of our implementation for various tensor orders and rank $R = 10, 50$. The performance metric we compute is the speedup attained using $p = \bigtimes_{i=1}^{N} p_i$ processors. In the sequel, we plot the speedup and show exact performance times. The times that will be presented are the averages of five Monte Carlo trials. The tensors that we conducted our tests on are the following (in all cases, the tensor $\boldsymbol{\mathcal{X}}$ has one billion entries or a little above a billion entries):

1. Third order tensor: we set $I = J = K = 1000$.

2. Fourth order tensor: we set $I = J = K = L = 178$.

3. Fifth order tensor: we set $I = J = K = L = M = 64$.

The number of processors used in the trials were $p = 1, 2, 4, 8, 16, 32, 64, 128, 256$. For specific tensor orders, we also used $p = 27, 81, 125, 243$, in order to test the performance of cubic grids. We present the formation of the grid for the various experiments carried out:

Table 7.1: Grid formations used.

Cores	Third order tensor	Fourth order tensor	Fifth order tensor
1	$1 \times 1 \times 1$	$1 \times 1 \times 1 \times 1$	$1 \times 1 \times 1 \times 1 \times 1$
2	$2 \times 1 \times 1$	$2 \times 1 \times 1 \times 1$	$2 \times 1 \times 1 \times 1 \times 1$
4	$2 \times 2 \times 1$	$2 \times 2 \times 1 \times 1$	$2 \times 2 \times 1 \times 1 \times 1$
8	$2 \times 2 \times 2$	$2 \times 2 \times 2 \times 1$	$2 \times 2 \times 2 \times 1 \times 1$
16	$4 \times 2 \times 2$	$2 \times 2 \times 2 \times 2$	$2 \times 2 \times 2 \times 2 \times 1$
27	$3 \times 3 \times 3$	–	–
32	$4 \times 4 \times 2$	$4 \times 2 \times 2 \times 2$	$2 \times 2 \times 2 \times 2 \times 2$
64	$4 \times 4 \times 4$	$4 \times 4 \times 2 \times 2$	$4 \times 2 \times 2 \times 2 \times 2$
81	–	$3 \times 3 \times 3 \times 3$	–
125	$5 \times 5 \times 5$	–	–
128	$8 \times 4 \times 4$	$4 \times 4 \times 4 \times 2$	$4 \times 4 \times 2 \times 2 \times 2$
243	–	–	$3 \times 3 \times 3 \times 3 \times 3$
256	$8 \times 8 \times 4$	$4 \times 4 \times 4 \times 4$	$4 \times 4 \times 4 \times 2 \times 2$

We first present results from the unconstrained problem.

7.1.1 Unconstrained Problem

Speedup Plots

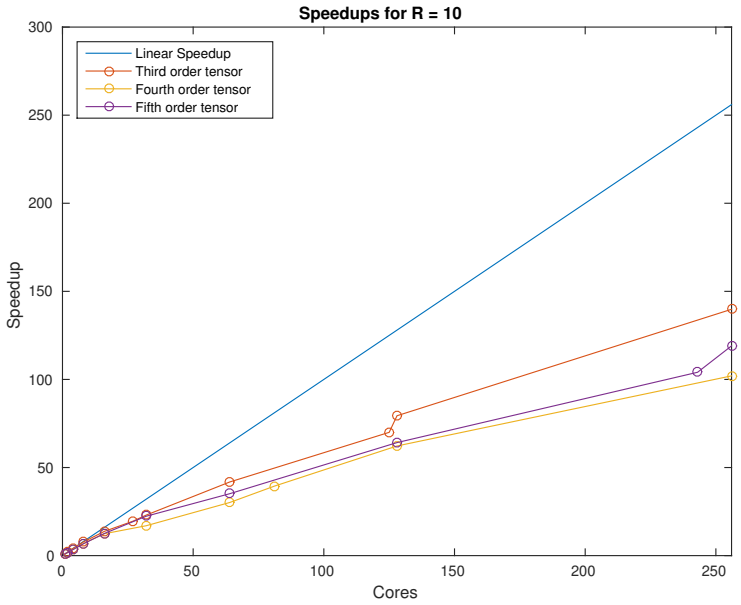

Figure 7.1: Speedup plot for the unconstrained problem, for $R = 10$ and for $p = 1, 2, 4, 8, 16, 27, 32, 64, 81, 125, 128, 243, 256$.

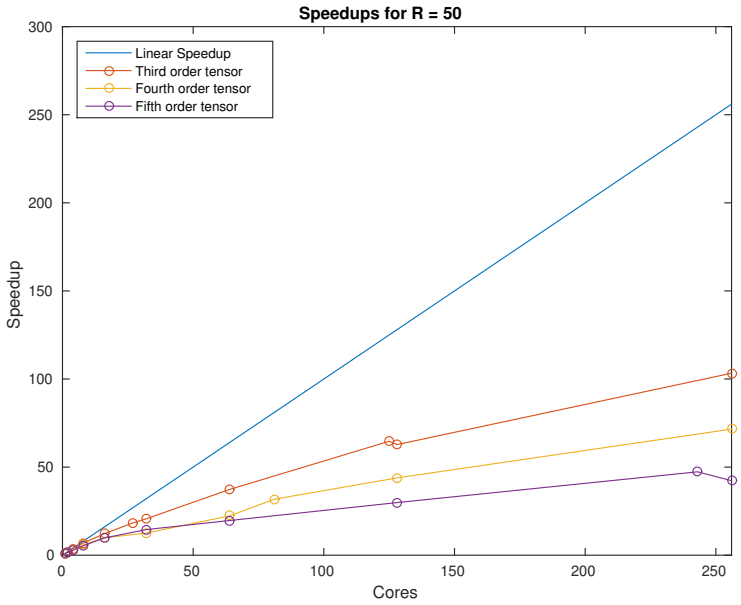

Figure 7.2: Speedup plot for the unconstrained problem, for $R = 50$ and for $p = 1, 2, 4, 8, 16, 27, 32, 64, 81, 125, 128, 243, 256$.

Exact times

Table 7.2: Exact times for various tensor orders and rank $R = 10$ for the unconstrained problem.

Cores	Third order tensor	Fourth order tensor	Fifth order tensor
1	159.257 s	222.011 s	774.17 s
2	78.77 s	125.646 s	434.156 s
4	39.147 s	63.879 s	227.832 s
8	20.534 s	32.34 s	115.235 s
16	11.589 s	17.886 s	61.672 s
27	8.142 s	–	–
32	6.928 s	13.096 s	34.648 s
64	3.81 s	7.362 s	22.027 s
81	–	5.635 s	–
125	2.276 s	–	–
128	2.006 s	3.567 s	12.067 s
243	–	–	7.439 s
256	1.139 s	2.176 s	6.503 s

Table 7.3: Exact times for various tensor orders and rank $R = 50$ for the unconstrained problem.

Cores	Third order tensor	Fourth order tensor	Fifth order tensor
1	423.309 s	712.77 s	2146.863 s
2	222.632 s	443.927 s	1379.992 s
4	112.565 s	245.307 s	754.291 s
8	61.004 s	123.023 s	399.606 s
16	34.716 s	72.039 s	219.717 s
27	23.059 s	–	–
32	20.524 s	56.895 s	147.784 s
64	11.334 s	31.891 s	109.083 s
81	–	22.449 s	–
125	6.547 s	–	–
128	6.744 s	16.247 s	71.906 s
243	–	–	50.77 s
256	4.098 s	9.96 s	45.361 s

We will now present results for the NTF problem.

7.1.2 NTF

Speedup Plots

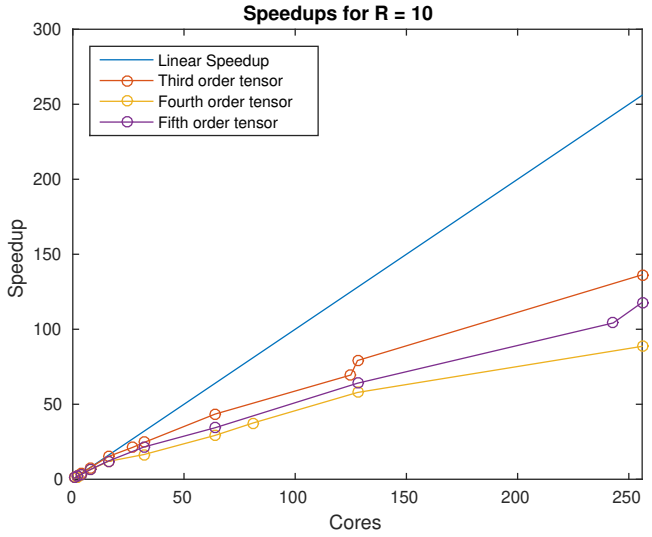

Figure 7.3: Speedup plot for the Nonnegative Tensor Factorization problem, for $R = 10$ and for $p = 1, 2, 4, 8, 16, 27, 32, 64, 81, 125, 128, 243, 256$.

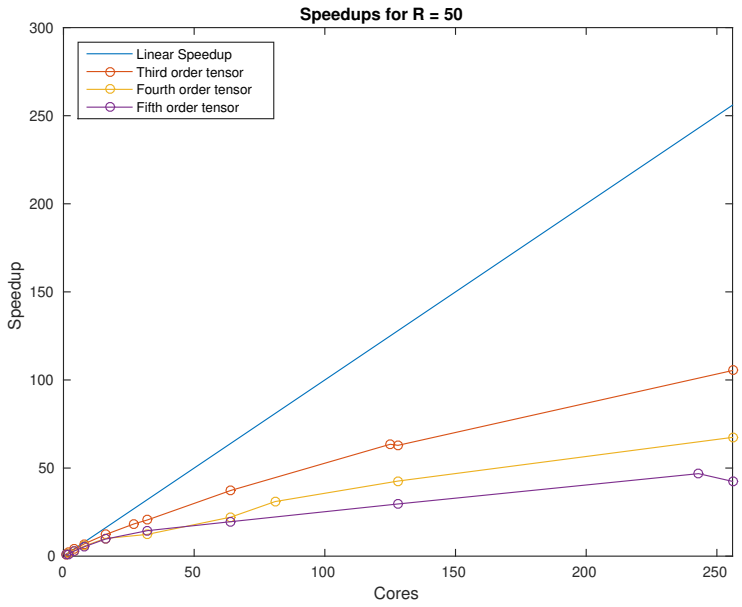

Figure 7.4: Speedup plot for the Nonnegative Tensor Factorization problem, for $R = 50$ and for $p = 1, 2, 4, 8, 16, 27, 32, 64, 81, 125, 128, 243, 256$.

Exact times

Table 7.4: Exact times for various tensor orders and rank $R = 10$ for the NTF problem.

Cores	Third order tensor	Fourth order tensor	Fifth order tensor
1	165.193 s	215.081 s	762.868 s
2	79.047 s	125.035 s	435.582 s
4	40.019 s	63.829 s	228.697 s
8	21.017 s	33.663 s	115.854 s
16	10.881 s	18.047 s	62.748 s
27	7.751 s	–	–
32	6.738 s	13.041 s	35.435 s
64	3.891 s	7.355 s	22.205 s
81	–	5.736 s	–
125	2.377 s	–	–
128	2.087 s	3.716 s	11.915 s
243	–	–	7.32 s
256	1.212 s	2.426 s	6.471 s

Table 7.5: Exact times for various tensor orders and rank $R = 50$ for the NTF problem.

Cores	Third order tensor	Fourth order tensor	Fifth order tensor
1	431.771 s	709.627 s	2137.673 s
2	225.379 s	443.120 s	1396.606 s
4	112.651 s	240.3 s	754.005 s
8	61.646 s	123.059 s	400.661 s
16	35.355 s	70.906 s	221.396 s
27	23.569 s	–	–
32	21.076 s	57.533 s	147.984 s
64	11.582 s	32.044 s	109.259 s
81	–	22.901 s	–
125	6.8 s	–	–
128	6.865 s	16.651 s	72.164 s
243	–	–	45.651 s
256	4.098 s	10.526 s	50.381 s

7.1.3 Conclusions

Based on the results, we observe that performance is similar for both problems; the procedure that solves the least squares problems has little effect on the algorithm's per-

formance. This is in accordance to what was mentioned in chapter 5, regarding the ALS algorithm's computational bottleneck.

Chapter 8

Discussion and Future Work

8.1 Conclusions

We considered tensor rank factorization problems, unconstrained and problems with nonnegativity constraints. In order to tackle the computational bottleneck that characterizes the ALS algorithm, we presented the Dimension Trees. These structures store intermediate results that are repeated among several factors. We explained how they function, which quantities are stored, which operations are used to calculate them, and we presented examples for third–order tensors and fourth–order tensors. Dimension Trees can be used regardless of factor constraints. We also presented a parallel implementation, based on [28] and [29], for N–th order tensors, with $N \geq 3$.

8.2 Future Work

We conclude this thesis by presenting possible future extensions of this work.

8.2.1 Inclusion of additional constraints

While we focused on unconstrained and nonnegativity constraints, we may wish to impose other constraints on factors, based on how we model a problem. Sparsity constraints and orthogonality constraints, for example, in which cases, for each factor we will solve the corresponding matrix least squares problems.

8.2.2 Using Dimension Tree–like schemes for other decompositions

We can examine if we can derive dimension tree–like schemes for other decomposition models. Some examples follow.

Tucker3 Decomposition

The Tucker3 decomposition of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ is given by

$$x_{i,j,k} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} q_{p,q,r} a_{i,p} b_{j,q} c_{k,r} + e_{i,j,k}, \tag{8.1}$$

where $\mathbf{A} \in \mathbb{R}^{I \times P}$, $\mathbf{B} \in \mathbb{R}^{J \times Q}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$ are the factor matrices of the first, second and third modes respectively, $\boldsymbol{\mathcal{E}} \in \mathbb{R}^{I \times J \times K}$ contains the residuals, and $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{P \times Q \times R}$ is called the core tensor. A key difference between PARAFAC and Tucker3 is the existence of three rank parameters in the latter, instead of one rank parameter in PARAFAC. PARAFAC can be viewed as a special case of Tucker3, where $P = Q = R$, and $g_{i,j,k} = 1$, if $i = j = k$, otherwise it is equal to zero.

Tensor Completion

In cases where we have missing data, we must use algorithms that take this fact into account. Algorithms for factorizing tensors with missing elements are called *tensor completion* algorithms. The cost function for this problem is the following

$$f_{\boldsymbol{\mathcal{W}}}(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}) = \frac{1}{2} \| \boldsymbol{\mathcal{W}} \circledast \left(\boldsymbol{\mathcal{X}} - [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!] \right) \|_F^2, \tag{8.2}$$

where the tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is called the *mask* and is defined as

$$w_{i_1,\ldots,i_N} = \begin{cases} 1 & \text{if } x_{i_1,\ldots,i_N} \text{ is observed} \\ 0 & \text{if } x_{i_1,\ldots,i_N} \text{ is missing.} \end{cases} \tag{8.3}$$

Bibliography

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

[2] P. M. Kroonenberg, *Applied Multiway Data Analysis*. Wiley-Interscience, 2008.

[3] A. Cichocki, R. Zdunek, A. H. Phan, and S. Amari, *Nonnegative Matrix and Tensor Factorizations*. Wiley, 2009.

[4] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017.

[5] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *Signal Processing Magazine, IEEE*, vol. 32, no. 2, pp. 145–163, 2015.

[6] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM Journal on imaging sciences*, vol. 6, no. 3, pp. 1758–1789, 2013.

[7] L. Sorber, M. Van Barel, and L. De Lathauwer, "Structured data fusion." *IEEE Journal on Selected Topics in Signal Processing*, vol. 9, no. 4, pp. 586–600, 2015.

[8] A. P. Liavas and N. D. Sidiropoulos, "Parallel algorithms for constrained tensor factorization via alternating direction method of multipliers," *IEEE Transactions on Signal Processing*, vol. 63, no. 20, pp. 5450–5463, 2015.

[9] K. Huang, N. D. Sidiropoulos, and A. P. Liavas, "A flexible and efficient framework for constrained matrix and tensor factorization," *IEEE Transactions on Signal Processing*, accepted for publication, May 2016.

[10] L. Karlsson, D. Kressner, and A. Uschmajew, "Parallel algorithms for tensor completion in the CP format," *Parallel Computing*, 2015.

[11] S. Smith and G. Karypis, "A medium-grained algorithm for distributed sparse tensor factorization," *30th IEEE International Parallel & Distributed Processing Symposium*, 2016.

[12] O. Kaya and B. Uçar, "Scalable sparse tensor decompositions in distributed memory systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 77.

[13] N. Guan, D. Tao, Z. Luo, and B. Yuan, "Nenmf: An optimal gradient method for nonnegative matrix factorization," *IEEE Transactions on Signal Processing*, vol. 60, no. 6, pp. 2882–2898, 2012.

[14] Y. Zhang, G. Zhou, Q. Zhao, A. Cichocki, and X. Wang, "Fast nonnegative tensor factorization based on accelerated proximal gradient and low-rank approximation," *Neurocomputing*, vol. 198, no. Supplement C, pp. 148 – 154, 2016.

[15] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.

[16] Y. Nesterov, *Introductory lectures on convex optimization*. Kluwer Academic Publishers, 2004.

[17] B. O' Donoghue and E. Candes, "Adaptive restart for accelerated gradient schemes," *Foundations of computational mathematics*, vol. 15, no. 3, pp. 715–732, 2015.

[18] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. SIAM, 1997.

[19] J. Håstad, "Tensor rank is np-complete," *J. Algorithms*, vol. 11, no. 4, pp. 644–654, Dec. 1990. [Online]. Available: http://dx.doi.org/10.1016/0196-6774(90)90014-6

[20] E. Acar, S. Camtepe, M. Krishnamoorthy, and B. Yener, "Modeling and multiway analysis of chatroom tensors," vol. 3495, 05 2005, pp. 256–268.

[21] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "Tensors for data mining and data fusion: Models, applications, and scalable algorithms," *ACM Trans. Intell. Syst. Technol.*, vol. 8, no. 2, pp. 16:1–16:44, Oct. 2016. [Online]. Available: http://doi.acm.org/10.1145/2915921

[22] E. Acar and B. Yener, "Unsupervised multiway data analysis: A literature survey," *IEEE Trans. on Knowl. and Data Eng.*, vol. 21, no. 1, pp. 6–20, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2008.112

[23] B. W. Bader and T. G. Kolda, "Efficient matlab computations with sparse and factored tensors," *SIAM J. Sci. Comput.*, vol. 30, no. 1, pp. 205–231, Dec. 2007. [Online]. Available: http://dx.doi.org/10.1137/060676489

[24] O. Kaya and B. Uçar, "Parallel Candecomp/Parafac Decomposition of Sparse Tensors Using Dimension Trees," *SIAM Journal on Scientific Computing*, vol. 40, no. 1, pp. C99 – C130, 2018. [Online]. Available: https://hal.inria.fr/hal-01397464

[25] M. Razaviyayn, M. Hong, and Z.-Q. Luo, "A unified convergence analysis of block successive minimization methods for nonsmooth optimization," *SIAM Journal on Optimization*, vol. 23, no. 2, pp. 1126–1153, 2013.

[26] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to Parallel Computing (2nd Edition)*. Pearson, 2003.

[27] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[28] A. P. Liavas, G. Kostoulas, G. Lourakis, K. Huang, and N. D. Sidiropoulos, "Nesterov-based parallel algorithm for large-scale nonnegative tensor factorization," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, USA, March 5-9, 2017*. IEEE, 2017.

[29] ——, "Nesterov-based alternating optimization for nonnegative tensor factorization: Algorithm and parallel implementations," *IEEE Transactions on Signal Processing*, vol. 66, no. 4, pp. 944–953, Feb. 2018.