TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
TELECOMMUNICATIONS DIVISION

Multiway Data Analysis: Nonnegative Tensor Factorization Algorithms and Parallel Implementations

by

Georgios Kostoulas

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

ELECTRICAL AND COMPUTER ENGINEERING

September 2016

THESIS COMMITTEE

Professor Athanasios P. Liavas, *Thesis Supervisor*
Professor Minos Garofalakis
Associate Professor George Karystinos

Abstract

We consider the problem of nonnegative tensor factorization. Our aim is to derive an efficient algorithm that is also suitable for parallel implementation. We adopt the alternating optimization (AO) framework and solve each matrix nonnegative least-squares problem via a Nesterov-type algorithm for strongly convex problems. We describe two parallel implementations of the algorithm, with and without data replication. We test the efficiency of the algorithm in extensive numerical experiments and measure the attained speedup in a parallel computing environment. It turns out that the derived algorithm is a competitive candidate for the solution of very large-scale dense nonnegative tensor factorization problems.

Acknowledgements

I would like to thank everyone that helped me with this thesis.

I would like to express my gratitude to Professor Athanasios Liavas, my thesis supervisor, for the continuous support of my M.Sc study and research, for the chance to work on this thesis and his research, for his patient guidance, and for his useful critiques on this work.

I would like to thank my friends for the adorable moments I had with them and for their support.

I thank my fellow labmates in Telecom Lab: Georgios Lourakis and Panos Alevizos.

I would like to offer my special thanks to my sister Katerina and my brother Nikos for the guidance and valuable support during my student life.

Last but not least, I would like to thank my parents for their support and encouragement throughout my study.

Table of Contents

List of Figures

List of Abbreviations

NTF	Nonnegative Tensor Factorization
CPD	Canonical Polyadic Decomposition
KKT	Karush-Kuhn-Tucker
SVD	Singular Value Decomposition
PCA	Principal Component Analysis
ALS	Alternating Least Squares
AO	Alternating Optimization
MNLS	Matrix Nonnegative Least Squares
NCG	Nonlinear Conjugate Gradient
CANDECOMP	Canonical decomposition
PARAFAC	Parallel factors

Chapter 1

Introduction

1.1 Motivation and Related Work

Tensors are mathematical objects that have recently gained great popularity due to their ability to model multiway data dependencies [1], [2], [3]. 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 [1], [2]. 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 nonnegative PARAFAC, which, for simplicity, we call 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 [2], [4]. Recent work for constrained tensor factorization/completion includes, among others, [5], [6], [7] and [8]. In [5], several NTF algorithms and a detailed convergence analysis have been developed. A general framework for joint matrix/tensor factorization/completion has been developed in [6]. In [7], 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 [8], 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 [9], 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 [10] and [11].

1.2 Contribution

In this work, we focus on (dense) NTF problems. Our aim is to derive an efficient NTF algorithm, suitable for parallel implementation. We adopt the AO framework and solve each matrix nonnegative least-squares (MNLS) problem via a Nesterov-type (accelerated gradient) algorithm for smooth *and strongly convex* problems [12]. We note that a Nesterov-type algorithm for *strictly convex* MNLS problems has been used in [13] for Nonnegative Matrix Factorization (NMF) and [14] for NTF. However, if the MNLS problem is strongly and not, simply, strictly convex, then exploitation of strong convexity is critical for the efficiency of the algorithm. We describe two parallel implementations of the algorithm, with and without data replication. We test the efficiency of the proposed algorithm in extensive numerical experiments and measure the speedup attained by its MPI implementations on a multi-core environment. We conclude that the proposed algorithm is a strong candidate for the solution of very large NTF problems.

1.3 Notation

Vectors, matrices, and tensors are denoted by small, capital, and calligraphic capital bold letters, respectively; for example, $\mathbf{x}$, $\mathbf{X}$, and $\boldsymbol{\mathcal{X}}$. $\mathbb{R}_+^{I\times J\times K}$ 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 $\mathbf{A} \odot \mathbf{B}$ 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.

1.4 Thesis Outline

The thesis is organized as follows :

- In Chapter 2, we briefly describe some tensor preliminaries that we are going to use. Then, we provide an introduction to the CP decomposition problem.
- In Chapter 3, we present the Nesterov algorithm for set-constrained smooth and strongly convex optimization problems and derive a Nesterov-type algorithm for the MNLS problem.
- In Chapter 4, we use the algorithmic framework presentend in Chapter 3 in order to propose an algorithm that can be used as building block for AO NTF and present the results of the Nesterov-based AO NTF algorithm.
- In Chapter 5, we propose two new parallel implementations of the algorithm we described in Chapter 4.
- Finally, in Chapter 6, we conclude our work and make suggestions for future work.

Chapter 2

Tensor Preliminaries

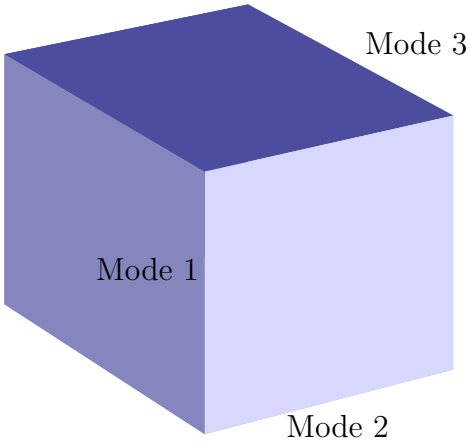

Figure 2.1: A 3 mode tensor

2.1 Tensors (Multi-way Arrays)

A tensor is a multi-way or multi-dimensional array. The order of a tensor is the number of dimensions, also known as ways or modes. A tensor, as shown in [2], can be formally defined as

Definition 2.1 ***(Tensor)*** *Let* $I_1, I_2, \ldots, I_N \in \mathbb{N}$ *denote index upper bounds. A tensor* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ *of order* N *is a* N*-way array, with elements* $\boldsymbol{\mathcal{X}}(i_1, i_2, \ldots, i_n)$ *for* $i_n \in \{1, 2, \ldots, I_n\}$ *and* $1 \leq n \leq N$.

Higher order tensors are generalizations of vectors and matrices. A vector is a first-order tensor while a matrix is a second-order tensor. For example, a third-order tensor (or three-way array) has three modes (or indices or dimensions) as shown in Figure 2.1.

2.1.1 Norm of a Tensor

The *Frobenius norm* of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the square root of the sum of the squares of all its elements, i.e.,

$$\|\boldsymbol{\mathcal{X}}\|_F = \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} [\boldsymbol{\mathcal{X}}(i_1, i_2, \ldots, i_n)]^2}.$$

This is analogous to the matrix Frobenius norm, which is denoted $\|\mathbf{A}\|_F$ for a matrix $\mathbf{A}$.

2.1.2 Rank-one Tensors

The outer product of two vectors creates a rank-one matrix. Similarly, in the tensor case, we have the following definition from [3].

Definition 2.2 *A tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is rank one if it can be written as the outer product of the N vectors $\boldsymbol{a}^{(1)}, \boldsymbol{a}^{(2)}, \ldots, \boldsymbol{a}^{(N)}$, that is,*

$$\boldsymbol{\mathcal{X}} = \boldsymbol{a}^{(1)} \circ \boldsymbol{a}^{(2)} \circ \cdots \circ \boldsymbol{a}^{(N)}$$

or each tensor element can be expressed as

$$\boldsymbol{\mathcal{X}}(i_1, i_2, \ldots, i_n) = \boldsymbol{a}^{(1)}(i_1) \circ \boldsymbol{a}^{(2)}(i_2) \circ \cdots \circ \boldsymbol{a}^{(N)}(i_N), \text{ for all } 1 \leq i_n \leq I_n$$

Figure 2.2 illustrates $\boldsymbol{\mathcal{X}} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$, a three-mode rank-one tensor.

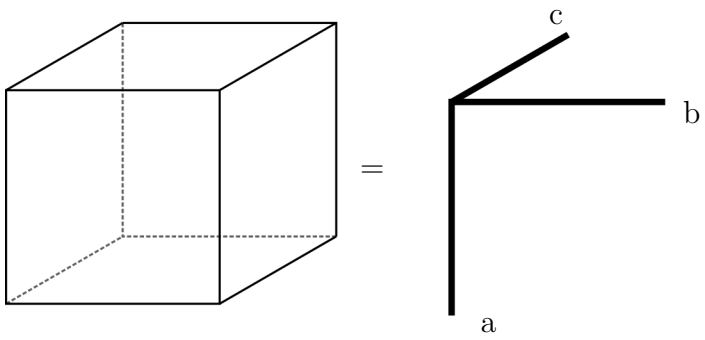

Figure 2.2: Schematic of a rank-one third-order tensor.

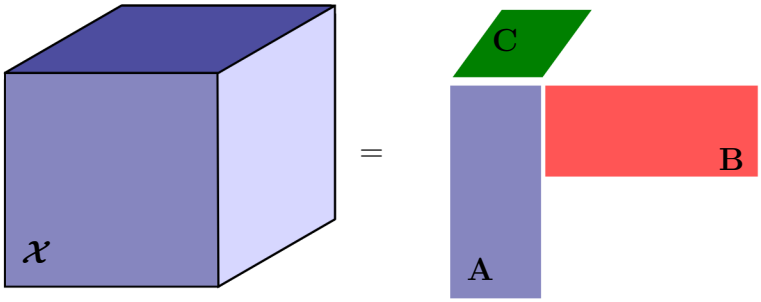

Figure 2.3: Schematic of a rank-F third-order tensor.

2.1.3 Rank of a Tensor

The rank of a matrix is the minimum number of rank one matrices needed to synthesize the given matrix. Alternatively, $\text{rank}(\mathbf{X}) = F$ if and only if F is the smallest integer such that $\mathbf{X} = \mathbf{A}\mathbf{B}^T$ for some matrices $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_F]$ and $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_F]$.

We now focus on a three way tensor of size $I \times J \times K$. Everything generalizes to higher-order tensors. The rank of a tensor $\boldsymbol{\mathcal{X}}$ is the minimum number of rank-one tensors needed to produce $\boldsymbol{\mathcal{X}}$ as their sum ([15]). Therefore a tensor of rank F can be written as

$$\boldsymbol{\mathcal{X}} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] = \sum_{f=1}^{F} \mathbf{a}_f \circ \mathbf{b}_f \circ \mathbf{c}_f,$$

and each tensor element can be written as

$$\boldsymbol{\mathcal{X}}(i,j,k) = \sum_{f=1}^{F} \mathbf{A}(i,f)\mathbf{B}(j,f)\mathbf{C}(k,f), \begin{cases} & i \in \{1, \ldots, I\}, \\ \forall & j \in \{1, \ldots, J\}, \\ & k \in \{1, \ldots, K\}, \end{cases}$$

were $A = [\mathbf{a}_1, \ldots, \mathbf{a}_F]$, $B = [\mathbf{b}_1, \ldots, \mathbf{b}_F]$ and $C = [\mathbf{c}_1, \ldots, \mathbf{c}_F]$. In Figure 2.3, we show a rank-F tensor.

2.1.4 Tensor Matricizations

Matricization, also known as unfolding or flattening, is the process of reordering the elements of an N-way array into a matrix ([3, pp. 5]). The mode-n matricization of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is denoted by $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times I_1 I_2 \ldots I_N / I_n}$ and arranges the mode-n slices to create the resulting matrix. If we have given the capital letters $\mathbf{A}, \mathbf{B}, \mathbf{C}$ to our modes then

we use the $\mathbf{X_A}, \mathbf{X_B}, \mathbf{X_C}$ to denote the matricization. We can see an unfolding example in Figure 2.4.

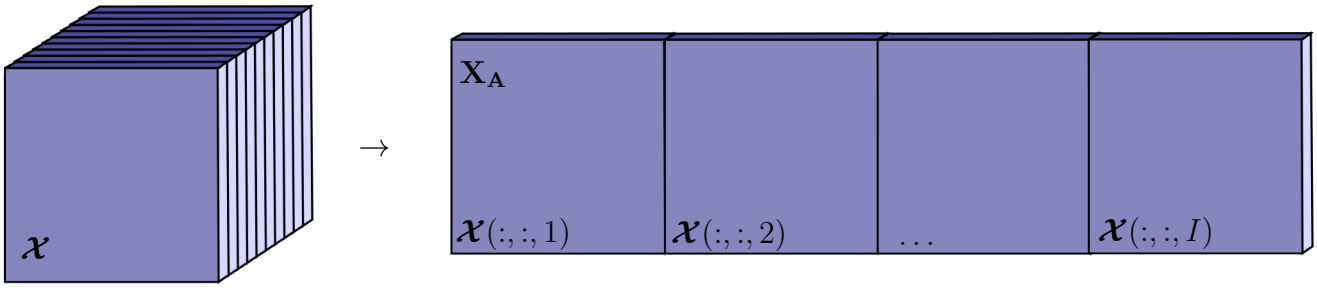

Figure 2.4: Mode one matricization of a third-order tensor.

2.2 Tensor rank and the CANDECOMP/PARAFAC decomposition

The CP decomposition factorizes a tensor into a sum of rank-one tensors. For example, given a third-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$, we wish to write it as

$$\boldsymbol{\mathcal{X}} = \sum_{f=1}^{F} \mathbf{a}_f \circ \mathbf{b}_f \circ \mathbf{c}_f, \tag{2.1}$$

where F is a positive integer, $\mathbf{a}_f \in \mathbb{R}^I$, $\mathbf{b}_f \in \mathbb{R}^J$, and $\mathbf{c}_f \in \mathbb{R}^K$, for $f = 1, \ldots, F$. The factor matrices refer to the combination of the vectors from the rank-one components, $\mathbf{A} = [\mathbf{a}_1 \ \cdots \ \mathbf{a}_F] \in \mathbb{R}_+^{I \times F}$, $\mathbf{B} = [\mathbf{b}_1 \ \cdots \ \mathbf{b}_F] \in \mathbb{R}_+^{J \times F}$, and $\mathbf{C} = [\mathbf{c}_1 \ \cdots \ \mathbf{c}_F] \in \mathbb{R}_+^{K \times F}$. Using these definitions, (2.1) can be written in matricized form, given by [3], as

$$\mathbf{X_A} = \mathbf{A}\,(\mathbf{C} \odot \mathbf{B})^T, \ \mathbf{X_B} = \mathbf{B}\,(\mathbf{C} \odot \mathbf{A})^T, \ \mathbf{X_C} = \mathbf{C}\,(\mathbf{B} \odot \mathbf{A})^T. \tag{2.2}$$

We can formulate the CPD problem as an optimization problem

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \quad f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}, \mathbf{B}, \mathbf{C}), \tag{2.3}$$

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

$$f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \frac{1}{2} \left\| \boldsymbol{\mathcal{X}} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \right\|_F^2, \tag{2.4}$$

More about the CP problem and the special case of Nonnegative Tensor Factorization will be presented in Chapter 4.

Chapter 3

Solving the Nonnegative Least-Squares Problem

In this chapter, we present an optimization approach of the nonnegative least-squares problem and its variations that we are going to use in this thesis. We solve the problem with an optimal method called "Nesterov's Accelerated Gradient Descent." Firstly, we start with some definitions that are going to prove useful and then we continue with the vector case of the LS problem and then we move to the matrix case.

3.1 Strongly Convex and Lipschitz functions

In this section, we present some useful definitions that will help us better understand the algorithms, their convergence properties, and the reason behind some decisions we are going to make (See [16] and [17]).

Definition 3.1 *[16, pp. 276] We say that $f : \mathcal{X} \to \mathbb{R}$ is μ-strongly convex if it satisfies the following inequality for some $\mu > 0$ and all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$:*

$$f(\mathbf{x}) - f(\mathbf{y}) \leq \nabla f(\mathbf{x})^T(\mathbf{x} - \mathbf{y}) - \frac{\mu}{2}\|\mathbf{x} - \mathbf{y}\|^2. \tag{3.1}$$

It is immediate to verify that a function f is μ-strongly convex if and only if $f(\mathbf{x}) - \frac{\mu}{2}\|\mathbf{x}\|^2$ is convex. In particular, if f is twice differentiable, then the eigenvalues of the Hessian of f have to be larger than μ:

$$\mu\mathbf{I} \preceq \nabla^2 f(\mathbf{x}).$$

For $\mu = 0$ we recover the basic inequality characterizing convexity. We have established a lower bound on the Hessian. We will now establish an upper bound.

Definition 3.2 *[16, pp.266] A continuously differentiable function is L-smooth if the gradient ∇f L-Lipschitz, that is*

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|^2, \ \forall \mathbf{x}, \mathbf{y} \in \mathcal{X} \subseteq \mathbb{R}^n. \tag{3.2}$$

In the same form with (3.1) we have

$$f(\mathbf{x}) - f(\mathbf{y}) \leq \nabla f(\mathbf{x})^T(\mathbf{x} - \mathbf{y}) + \frac{L}{2}\|\mathbf{x} - \mathbf{y}\|^2. \tag{3.3}$$

For a twice differentiable function f, L-smoothness is equivalent to the largest eigenvalue of the Hessian to be smaller or equal to L at any point, so we write

$$\nabla^2 f(\mathbf{x}) \preceq L\mathbf{I}, \ \forall \mathbf{x} \in \mathcal{X}.$$

The ratio $q = \dfrac{L}{\mu}$ is thus an upper bound on the condition number of the Hessian matrix, i.e. the ratio of its largest eigenvalue to its smallest eigenvalue.

3.2 Vector nonnegative least-squares

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^n$ and consider the nonnegative vector least-squares problem

$$\min_{\mathbf{x} \geq 0} f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2, \tag{3.4}$$

or, equivalently,

$$\min_{\mathbf{x} \geq 0} f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{A}\mathbf{x} + \frac{1}{2}\mathbf{b}^T\mathbf{b}. \tag{3.5}$$

The unconstrained problem arises in many fields and has many names, e.g., regression analysis or least-squares approximation. Its analytical solution is $\mathbf{x} = \mathbf{A}^\dagger\mathbf{b}$, where $\mathbf{A}^\dagger$ is the pseudo-inverse of $\mathbf{A}$. When linear inequality constraints are added, like nonnegativity in our case, the problem is called constrained regression or constrained least-squares, and there is no longer a simple analytical solution. So we have to recourse to iterative methods.

We know that the gradient of f is given by

$$\nabla f(\mathbf{x}) = \mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{A}^T\mathbf{b}.$$

Most iterative gradient methods require the computation of the gradient in each iteration.

We look for efficient computation of the quantities that are most expensive. We note that the quantities $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}^T\mathbf{b}$ may be computed once since they do not depend on $\mathbf{x}_k$. Furthermore,

1. The term $\mathbf{A}^T\mathbf{A} \in \mathbb{R}^{n\times n}$ needs $O(n^2m)$ computational complexity and its storage is $O(n^2)$ memory complexity. The term $\mathbf{A}^T\mathbf{A}\mathbf{x}_k$ has $O(n^2)$ computational complexity.

   If $n \gg m$, then is seems beneficial to avoid the computation of $\mathbf{A}^T\mathbf{A}$. Instead, we may compute $\mathbf{z}_k = \mathbf{A}^T(\mathbf{A}\mathbf{x}_k)$, with $O(mn)$ for $\mathbf{t}_k = \mathbf{A}\mathbf{x}_k$ and $O(mn)$ for $\mathbf{z}_k = \mathbf{A}^T\mathbf{t}_k$.

   If $n \ll m$, then it seems beneficial to compute once the term $\mathbf{Z} = \mathbf{A}^T\mathbf{A}$ and then, every time you need the gradient, compute $\mathbf{z}_k = \mathbf{Z}\mathbf{x}_k$. This is the case that we shall consider in this thesis.

2. It is beneficial to compute once the second part of the gradient, $\mathbf{A}^T\mathbf{b} \in \mathbb{R}^n$, and then use it freely.

3.3 Optimal first-order methods for set-constrained L-smooth μ-strongly convex problems

Let $0 < \mu \leq L < \infty$ and $f : \mathbb{R}^n \to \mathbb{R}$ be a smooth convex function, with Hessian $\nabla^2 f(\mathbf{x})$, such that

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

Then, we say that f is an L-smooth μ-strongly convex function, denoted as $f \in \mathcal{S}^{1,1}_{\mu,L}$ [12, p. 63]. Let $\mathbb{Q}$ be a closed convex set. Our aim is to solve problem

$$\min_{\mathbf{x}\in\mathbb{Q}} f(\mathbf{x}), \tag{3.7}$$

within accuracy $\epsilon > 0$, using only first-order, i.e., gradient, information. The accuracy of the solution is defined as follows. Let $f^* := \min_{\mathbf{x}\in\mathbb{Q}} f(\mathbf{x})$. A point $\bar{\mathbf{x}} \in \mathbb{Q}$ solves problem (3.7) within accuracy $\epsilon > 0$ if $f(\bar{\mathbf{x}}) - f^* \leq \epsilon$. It has been shown in [12, Chapter 2] that the black-box first-order oracle complexity of this class of problems is $O\left(\sqrt{\frac{L}{\mu}}\log\frac{1}{\epsilon}\right)$. An algorithm that achieves this complexity, and, thus, is first-order optimal, is given in Algorithm 1 (see [12, pp. 80, 81, 90]). If the projection onto set $\mathbb{Q}$ is easy to compute, then this algorithm is very efficient in practice. We note that, in general, constants μ and L are unknown and, thus, should be estimated.

Algorithm 1: Nesterov algorithm for set-constrained L-smooth and μ-strongly convex optimization

Input: $\mathbf{x}_0 \in \mathbb{R}^n$, μ, L. Set $\mathbf{y}_0 = \mathbf{x}_0$ and $\beta = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$.

1 k-th iteration

2 $\quad \mathbf{x}_{k+1} = \Pi_{\mathbb{Q}}\left(\mathbf{y}_k - \frac{1}{L}\nabla f(\mathbf{y}_k)\right)$

3 $\quad \mathbf{y}_{k+1} = \mathbf{x}_{k+1} + \beta(\mathbf{x}_{k+1} - \mathbf{x}_k)$.

We note that, if problem (3.7) is strictly but *not* strongly convex, the optimal algorithm differs from that presented in Algorithm 1 and the corresponding complexity becomes $O\left(\frac{1}{\sqrt{\epsilon}}\right)$. This is the framework that has been adopted in [13] and [14]. However, if relation (3.6) is true, then exploitation of strong convexity is crucial for the efficiency of the algorithm.

In order to apply Algorithm 1 to problem 3.4 we must initialize the values $\mathbf{Z} = \mathbf{A^T A}$ and $\mathbf{w} = \mathbf{A}^T\mathbf{b}$. We obtain μ and L from the highest and lowest eigenvalues of $\mathbf{Z}$. An example is shown in Figure 3.1. We can see that the convergence properties of the problem are strongly depended on the condition number of the problem.

3.4 Nesterov-type algorithm for strongly convex MNLS

In this subsection, we apply the approach of Nesterov to strongly convex Matrix nonnegative least-squares (MNLS) problems. We assume that $\mathbf{X} \in \mathbb{R}^{m\times k}$, $\mathbf{A} \in \mathbb{R}^{m\times n}$, and $\mathbf{B} \in \mathbb{R}^{k\times n}$ and consider the problem

$$\min_{\mathbf{A}\geq\mathbf{0}} f(\mathbf{A}) := \frac{1}{2}\left\|\mathbf{X} - \mathbf{A}\mathbf{B}^T\right\|_F^2. \tag{3.8}$$

The gradient of f is given by

$$\nabla f(\mathbf{A}) = -\left(\mathbf{X} - \mathbf{A}\mathbf{B}^T\right)\mathbf{B}. \tag{3.9}$$

If $\mathbf{B}^T\mathbf{B}$ is nonsingular, then problem (3.8) is L-smooth and μ-strongly convex, with L and μ being, respectively, equal to the largest and smallest eigenvalue of $\mathbf{B}^T\mathbf{B}$.

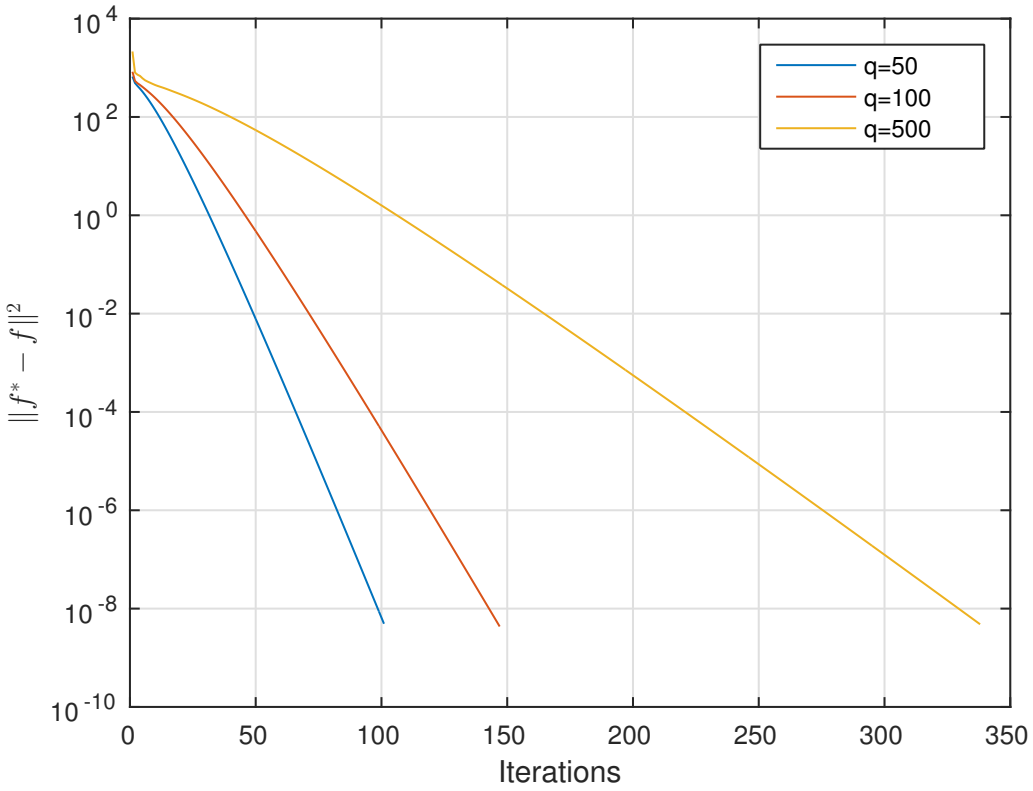

Figure 3.1: The convergence of Nesterov algorithm, for the vector NNLS problem for different conditions $q = 50, 100, 500$. The problem dimension is $n = 1000$.

Algorithm 2: Nesterov-type algorithm for MNLS

Input: $\mathbf{X} \in \mathbb{R}^{m\times k}$, $\mathbf{B} \in \mathbb{R}^{k\times n}$, $\mathbf{A}_0 \in \mathbb{R}^{m\times n}$, tol > 0.

Compute $\mathbf{W} = -\mathbf{X}\mathbf{B},\ \mathbf{Z} = \mathbf{B}^T\mathbf{B}$.

Compute $L = \max(\text{eig}(\mathbf{Z}))\ \ \mu = \min(\text{eig}(\mathbf{Z}))$.

Set $\mathbf{Y}_0 = \mathbf{A}_0$, $\beta = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$, $k = 0$.

while (1) **do**

  $\nabla f(\mathbf{Y}_k) = \mathbf{W} + \mathbf{A}_k\mathbf{Z}$;

  if $(\max(|\nabla f(\mathbf{Y}_k) \circledast \mathbf{Y}_k|) < \text{tol})$ **then**

    break;

  else

    $\mathbf{A}_{k+1} = \left[\mathbf{Y}_k - \frac{1}{L}\nabla f(\mathbf{Y}_k)\right]_+$;

    $\mathbf{Y}_{k+1} = \mathbf{A}_{k+1} + \beta\,(\mathbf{A}_{k+1} - \mathbf{A}_k)$;

    $k = k + 1$;

return $\mathbf{A}_k$.

Terminating condition If $\mathbf{\Lambda}$ denotes the Lagrange multiplier matrix associated with the matrix element-wise nonnegativity constraints in (3.8), then the Karush-Kuhn-Tucker (KKT) conditions for problem (3.8) are

$$\nabla f(\mathbf{A}) - \mathbf{\Lambda} = \mathbf{0},\ \ \mathbf{A} \geq \mathbf{0},\ \ \mathbf{\Lambda} \geq \mathbf{0},\ \ \mathbf{\Lambda} \circledast \mathbf{A} = \mathbf{0}. \tag{3.10}$$

From (3.10), we obtain that $\nabla f(\mathbf{A}) \circledast \mathbf{A} = \mathbf{\Lambda} \circledast \mathbf{A} = \mathbf{0}$. This equality can be used in a terminating condition. For example, we may terminate the algorithm if $\max\{|\nabla f(\mathbf{A}) \circledast \mathbf{A}|\} <$ tol, for a small real number tol > 0, where the operator max is applied element-wise.

A Nesterov-type algorithm for the solution of the MNLS problem (3.8) is given in Algorithm 2, where $[\cdot]_+$ denotes projection onto the set of matrices with nonnegative elements (see [18]).

Chapter 4

Alternating Optimization for Nonnegative Tensor Factorization

4.1 Nonnegative tensor factorization

Let tensor $\boldsymbol{\mathcal{X}}^o \in \mathbb{R}_+^{I \times J \times K}$ admit a factorization of the form

$$\boldsymbol{\mathcal{X}}^o = [\![\mathbf{A}^o, \mathbf{B}^o, \mathbf{C}^o]\!] = \sum_{f=1}^{F} \mathbf{a}_f^o \circ \mathbf{b}_f^o \circ \mathbf{c}_f^o, \tag{4.1}$$

where $\mathbf{A}^o = [\mathbf{a}_1^o \ \cdots \ \mathbf{a}_F^o] \in \mathbb{R}_+^{I \times F}$, $\mathbf{B}^o = [\mathbf{b}_1^o \ \cdots \ \mathbf{b}_F^o] \in \mathbb{R}_+^{J \times F}$, and $\mathbf{C}^o = [\mathbf{c}_1^o \ \cdots \ \mathbf{c}_F^o] \in \mathbb{R}_+^{K \times F}$. We observe the noisy tensor $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}}^o + \boldsymbol{\mathcal{E}}$, where $\boldsymbol{\mathcal{E}}$ is the additive noise. Estimates of $\mathbf{A}^o$, $\mathbf{B}^o$, and $\mathbf{C}^o$ can be obtained by computing matrices $\mathbf{A} \in \mathbb{R}_+^{I \times F}$, $\mathbf{B} \in \mathbb{R}_+^{J \times F}$, and $\mathbf{C} \in \mathbb{R}_+^{K \times F}$ that solve the optimization problem

$$\min_{\mathbf{A} \geq \mathbf{0}, \mathbf{B} \geq \mathbf{0}, \mathbf{C} \geq \mathbf{0}} \quad f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}, \mathbf{B}, \mathbf{C}), \tag{4.2}$$

where $f_{\boldsymbol{\mathcal{X}}}$ is a function measuring the quality of the factorization and the inequalities are element-wise. A common choice for $f_{\boldsymbol{\mathcal{X}}}$ is

$$f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \frac{1}{2} \left\| \boldsymbol{\mathcal{X}} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \right\|_F^2 . \tag{4.3}$$

If $\boldsymbol{\mathcal{Y}} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, then its matrix unfoldings, with respect to the first, second, and third dimension, are given by (2.2)

$$\mathbf{Y}_{\mathbf{A}} = \mathbf{A} \, (\mathbf{C} \odot \mathbf{B})^T, \ \mathbf{Y}_{\mathbf{B}} = \mathbf{B} \, (\mathbf{C} \odot \mathbf{A})^T, \ \mathbf{Y}_{\mathbf{C}} = \mathbf{C} \, (\mathbf{B} \odot \mathbf{A})^T.$$

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

$$\begin{aligned} f_{\boldsymbol{\mathcal{X}}}(\mathbf{A},\mathbf{B},\mathbf{C}) &= \frac{1}{2}\left\|\mathbf{X}_{\mathbf{A}} - \mathbf{A}\left(\mathbf{C}\odot\mathbf{B}\right)^{T}\right\|_F^2 \\ &= \frac{1}{2}\left\|\mathbf{X}_{\mathbf{B}} - \mathbf{B}\left(\mathbf{C}\odot\mathbf{A}\right)^{T}\right\|_F^2 \\ &= \frac{1}{2}\left\|\mathbf{X}_{\mathbf{C}} - \mathbf{C}\left(\mathbf{B}\odot\mathbf{A}\right)^{T}\right\|_F^2. \end{aligned} \tag{4.4}$$

These expressions form the basis for the AO NTF in the sense that, if we fix two matrix factors, then we can update the third by solving an MNLS problem. For reasons related with the conditioning of the MNLS problem, we propose to add a proximal term. More specifically, if $\mathbf{A}^k$, $\mathbf{B}^k$, and $\mathbf{C}^k$ are the estimates of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, respectively, after the k-th AO iteration, then $\mathbf{A}^{k+1}$ is given by

$$\mathbf{A}_{k+1} := \underset{\mathbf{A}\geq\mathbf{0}}{\operatorname{argmin}} \frac{1}{2}\left\|\mathbf{X}_{\mathbf{A}} - \mathbf{A}(\mathbf{C}_k\odot\mathbf{B}_k)^{T}\right\|_F^2 + \frac{\lambda}{2}\left\|\mathbf{A}-\mathbf{A}_k\right\|_F^2, \tag{4.5}$$

where $\lambda \geq 0$ determines the weight assigned to the proximal term. If $(\mathbf{C}_k \odot \mathbf{B}_k)$ is a well-conditioned matrix, then it is reasonable to put small weight on the proximal term and compute $\mathbf{A}_{k+1}$ that leads to a large decrease of the function $f_{\boldsymbol{\mathcal{X}}}(\mathbf{A},\mathbf{B}_k,\mathbf{C}_k)$. If, on the other hand, $(\mathbf{C}_k \odot \mathbf{B}_k)$ is an ill-conditioned matrix, then it is reasonable to put large weight on the proximal term, leading to a better conditioned problem and easy computation of $\mathbf{A}_{k+1}$ that improves the fit in $f_{\boldsymbol{\mathcal{X}}}(\mathbf{A},\mathbf{B}_k,\mathbf{C}_k)$ but is not very far from $\mathbf{A}_k$. This is the strategy we shall follow for the solution of problem (4.2) (see also [5], [19]).

The computational efficiency of the AO NTF heavily depends on the algorithm we use for the solution of problem (4.5). In this work, we adopt the approach of Nesterov for the solution of smooth and strongly convex problems. The derived algorithm is optimal under the (worst-case) black-box first-order oracle framework [12, Chapter 2] and is very efficient in practice. Furthermore, it leads to an AO NTF algorithm that is suitable for parallel implementation.

4.2 Nesterov-type algorithm for MNLS with proximal term

In this section, we present a variation of Algorithm 2 for the MNLS problem with proximal term. We have already pointed out that strong convexity is very important for the

Algorithm 3: Nesterov-type algorithm for MNLS with proximal term

Input: $\mathbf{X} \in \mathbb{R}^{m\times k}$, $\mathbf{B} \in \mathbb{R}^{k\times n}$, $\mathbf{A}_* \in \mathbb{R}^{m\times n}$, λ, tol > 0.

$\mathbf{W} = -\mathbf{X}\mathbf{B} - \lambda\mathbf{A}_*$, $\mathbf{Z} = \mathbf{B}^T\mathbf{B} + \lambda\mathbf{I}$.

$L' = \max(\text{eig}(\mathbf{B}^T\mathbf{B}))$, $\mu' = \min(\text{eig}(\mathbf{B}^T\mathbf{B}))$

$L = L' + \lambda$, $\mu = \mu' + \lambda$

$\mathbf{Y}_0 = \mathbf{A}_*$, $\beta = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$, $k = 0$

while (1) **do**

  $\nabla f(\mathbf{Y}_k) = \mathbf{W} + \mathbf{A}_k\mathbf{Z}$

  if $(\max(|\nabla f(\mathbf{Y}_k) \circledast \mathbf{Y}_k|) < \text{tol})$ **then**

    break

  else

    $\mathbf{A}_{k+1} = \left(\mathbf{Y}_k - \frac{1}{L}\nabla f(\mathbf{Y}_k)\right)_+$

    $\mathbf{Y}_{k+1} = \mathbf{A}_{k+1} + \beta\left(\mathbf{A}_{k+1} - \mathbf{A}_k\right)$

    $k = k + 1$

return $\mathbf{A}_k$.

algorithms efficiency. In order to avoid losing the strong convexity we use a proximal term. This algorithm will be the basic building block of the AO NTF algorithm of the next section.

4.2.1 Nesterov-type algorithm for MNLS with proximal term

In this subsection, we apply the approach of Nesterov to the MNLS problem with proximal term. Let $\mathbf{X} \in \mathbb{R}^{m\times k}$, $\mathbf{A}, \mathbf{A}_* \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{k\times n}$, and consider the problem

$$\min_{\mathbf{A}\geq\mathbf{0}} f(\mathbf{A}) := \frac{1}{2}\|\mathbf{X} - \mathbf{A}\mathbf{B}^T\|_F^2 + \frac{\lambda}{2}\|\mathbf{A} - \mathbf{A}_*\|_F^2. \tag{4.6}$$

As we mentioned in Section 4.1, the main reason we introduce the proximal term into the cost function is the improvement of the conditioning of the MNLS problem (and guarantee of strong convexity).

Problem (4.6) is L-smooth and μ-strongly convex, with $L = L' + \lambda$ and $\mu = \mu' + \lambda$, where L' and μ' are, respectively, the largest and smallest eigenvalue of $\mathbf{B}^T\mathbf{B}$. We note that the values of L and μ are *necessary* for the development of the Nesterov-type algorithm, thus, computation of L' and μ' is imperative.[1]

[1] An alternative to the direct computation of μ and L is to estimate L using line-search techniques and overcome the computation of μ using heuristic adaptive restart techniques [20]. However, in our case, this

Interestingly, if we consider problem (4.6) under the framework of the AO NTF algorithm, then we may use L' and μ' for adjusting the value of λ. More specifically, if $\mathbf{B}^T\mathbf{B}$ is a well-conditioned matrix and our main interest lies in the computation of $\mathbf{A}$ that satisfies approximate equality $\mathbf{X} \approx \mathbf{A}\mathbf{B}^T$ as accurately as possible, then we may set λ to a small positive value, say, $\lambda \lessapprox 10^{-2}$, putting small weight on the proximal term. If, on the other hand, $\mathbf{B}^T\mathbf{B}$ is an ill-conditioned matrix, then we may set $\lambda \gtrapprox 1$, putting more weight on the proximal term and improving the conditioning of the problem.

The gradient of f, at point $\mathbf{A}$, is

$$\nabla f(\mathbf{A}) = -\left(\mathbf{X} - \mathbf{A}\mathbf{B}^T\right)\mathbf{B} + \lambda(\mathbf{A} - \mathbf{A}_*). \tag{4.7}$$

If $\mathbf{\Lambda}$ denotes the Lagrange multiplier matrix associated with the matrix element-wise nonnegativity constraints in (4.6), then the Karush-Kuhn-Tucker (KKT) conditions for problem (4.6) are

$$\nabla f(\mathbf{A}) - \mathbf{\Lambda} = \mathbf{0},\ \mathbf{A} \geq \mathbf{0},\ \mathbf{\Lambda} \geq \mathbf{0},\ \mathbf{\Lambda} \circledast \mathbf{A} = \mathbf{0}. \tag{4.8}$$

From (4.8), we obtain that $\nabla f(\mathbf{A}) \circledast \mathbf{A} = \mathbf{\Lambda} \circledast \mathbf{A} = \mathbf{0}$. This equality can be used in a terminating condition. For example, we may terminate the algorithm if $\max\{|\nabla f(\mathbf{A}) \circledast \mathbf{A}|\} < \text{tol}$, for a small real number $\text{tol} > 0$, where the operator max is applied element-wise.

A Nesterov-type algorithm for the solution of the MNLS problem (4.6) is given in Algorithm 3, where $(\cdot)_+$ denotes projection onto the set of matrices with nonnegative elements. For notational convenience, we denote lines 2 to 13 of Algorithm 3 as

$$\mathbf{A}_{\text{opt}} = \text{Nesterov\_MNLS}(\mathbf{W}, \mathbf{Z}, \mathbf{A}_*, \lambda, \text{tol}).$$

We note that, in Algorithm 3, quantity λ is assumed to be an input. As we mentioned, an alternative is to adjust λ based on the values of L' and μ'.

Computational complexity of Algorithm 3

Quantities $\mathbf{W}$ and $\mathbf{Z}$ are computed once per algorithm call and cost, respectively, $O(mkn)$ and $O(kn^2)$ arithmetic operations. Quantities L and μ are also computed once and cost at most $O(n^3)$ operations. $\nabla f(\mathbf{Y}_k)$, $\mathbf{A}_k$, and $\mathbf{Y}_k$ are updated in every iteration with cost $O(mn^2)$, $O(mn)$, and $O(mn)$ arithmetic operations, respectively.

alternative is computationally demanding, especially for large-scale problems, and shall not be considered in this work.

Algorithm 4: Nesterov-based AO NTF

Input: $\boldsymbol{\mathcal{X}}$, $\mathbf{A}_0 \geq \mathbf{0}$, $\mathbf{B}_0 \geq \mathbf{0}$, $\mathbf{C}_0 \geq \mathbf{0}$, λ, tol.

Set $k = 0$
while (terminating condition is FALSE) **do**
  $\mathbf{W_A} = -\mathbf{X_A}(\mathbf{C}_k \odot \mathbf{B}_k) - \lambda \mathbf{A}_k$, $\mathbf{Z_A} = (\mathbf{C}_k \odot \mathbf{B}_k)^T(\mathbf{C}_k \odot \mathbf{B}_k) + \lambda \mathbf{I}$
  $\mathbf{A}_{k+1} = \text{Nesterov\_MNLS}(\mathbf{W_A}, \mathbf{Z_A}, \mathbf{A}_k, \lambda, \text{tol})$
  $\mathbf{W_B} = -\mathbf{X_B}(\mathbf{C}_k \odot \mathbf{A}_{k+1}) - \lambda \mathbf{B}_k$, $\mathbf{Z_B} = (\mathbf{C}_k \odot \mathbf{A}_{k+1})^T(\mathbf{C}_k \odot \mathbf{A}_{k+1}) + \lambda \mathbf{I}$
  $\mathbf{B}_{k+1} = \text{Nesterov\_MNLS}(\mathbf{W_B}, \mathbf{Z_B}, \mathbf{B}_k, \lambda, \text{tol})$
  $\mathbf{W_C} = -\mathbf{X_C}(\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1}) - \lambda \mathbf{C}_k$, $\mathbf{Z_C} = (\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1})^T(\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1}) + \lambda \mathbf{I}$
  $\mathbf{C}_{k+1} = \text{Nesterov\_MNLS}(\mathbf{W_C}, \mathbf{Z_C}, \mathbf{C}_k, \lambda, \text{tol})$
  $(\mathbf{A}_{k+1}, \mathbf{B}_{k+1}, \mathbf{C}_{k+1}) = \text{Normalize}(\mathbf{A}_{k+1}, \mathbf{B}_{k+1}, \mathbf{C}_{k+1})$
  $(\mathbf{A}_{k+1}, \mathbf{B}_{k+1}, \mathbf{C}_{k+1}) = \text{Accelerate}(\mathbf{A}_{k+1}, \mathbf{A}_k, \mathbf{B}_{k+1}, \mathbf{B}_k, \mathbf{C}_{k+1}, \mathbf{C}_k, k)$
  $k = k + 1$
return $\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{C}_k$.

4.3 Nesterov-based AO NTF

In Algorithm 4, we present the Nesterov-based AO NTF. We start from point $(\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0)$ and solve, in a circular manner, MNLS problems with proximal terms, based on the previous iteration.[2] In our algorithm, we incorporate two features borrowed from routine `parafac` of the N-way toolbox [21], implemented by functions termed "Normalize" and "Accelerate."

Function "Normalize" normalizes each column of $\mathbf{B}_{k+1}$ and $\mathbf{C}_{k+1}$ to unit Euclidean norm, putting all the power on the respective columns of $\mathbf{A}_{k+1}$.

Function "Accelerate" implements an acceleration mechanism which can be briefly described as follows. At iteration $k + 1 > k_0$, after the computation (and normalization) of $\mathbf{A}_{k+1}$, $\mathbf{B}_{k+1}$, and $\mathbf{C}_{k+1}$, we compute

$$\mathbf{A}_{\text{new}} = \mathbf{A}_{\text{old}} + s_{k+1}(\mathbf{A}_{k+1} - \mathbf{A}_{\text{old}}), \tag{4.9}$$

where $\mathbf{A}_{\text{old}}$ is the (normalized) output of the k-th AO iteration, and s_{k+1} is a small positive number; a simple choice for s_{k+1} is $s_{k+1} = (k+1)^{\frac{1}{n}}$, where n is initialized as $n = 3$ and its value may change as the algorithm progresses. In an analogous manner, we compute $\mathbf{B}_{\text{new}}$ and $\mathbf{C}_{\text{new}}$. If $f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}_{\text{new}}, \mathbf{B}_{\text{new}}, \mathbf{C}_{\text{new}}) \leq f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}_{k+1}, \mathbf{B}_{k+1}, \mathbf{C}_{k+1})$, then the acceleration step is successful, and we set $\mathbf{A}_{k+1} = \mathbf{A}_{\text{new}}$, $\mathbf{B}_{k+1} = \mathbf{B}_{\text{new}}$, and $\mathbf{C}_{k+1} = \mathbf{C}_{\text{new}}$. If the acceleration fails to decrease the cost function, then the acceleration step is ignored and we use $\mathbf{A}_{k+1}$,

[2] Of course, we do not need to use separate variables $\mathbf{W_A}$, $\mathbf{W_B}$, and $\mathbf{W_C}$; we did this for the reader's convenience (the same applies to the $\mathbf{Z}$'s).

$\mathbf{B}_{k+1}$, and $\mathbf{C}_{k+1}$ as input to the next AO update. If the acceleration step fails for n_0 iterations, then we set $n = n + 1$, thus, decreasing the exponent of the acceleration step. Typical values of k_0 and n_0 are $k_0 = 5$ and $n_0 = 3$. This acceleration scheme has been proven very useful, especially in ill-conditioned cases.

We can use various termination criteria for the AO NTF algorithm, based on either the (relative) change of the cost function values and/or the latent factors, or the KKT conditions. In our numerical experiments, we use a terminating criterion based on the relative change of the latent factors, that is,

$$\frac{\|\mathbf{M}_{k+1} - \mathbf{M}_k\|_F}{\|\mathbf{M}_k\|_F} < \text{tol}_{\text{AO}}, \text{ for } \mathbf{M} = \mathbf{A}, \mathbf{B}, \mathbf{C}, \tag{4.10}$$

where tol_{AO} is a small positive real number.

It has been shown in [19] that the AO NTF algorithm with proximal term (without acceleration) falls under the block successive upper bound minimization (BSUM) framework, which ensures convergence to a stationary point of problem (4.2). Our extensive numerical experiments have shown that acceleration can significantly improve the convergence speed in practice. We therefore include acceleration and aim to parallelize the accelerated algorithm.

4.4 Numerical experiments

In this section, we compare the performance of a Matlab implementation of the proposed algorithm with routines `parafac` of the N-way toolbox [21] and `sdf_nls` of tensorlab [22].[3] Our aim is to provide some general observations about the difficulty of the problems and the behavior of the algorithms and not a strict ranking of the algorithms.[4]

The `parafac` routine essentially implements an AO NTF algorithm, where each MNLS problem is solved via the function `fastnnls`, which is based on [23, §23.3]. It also incorporates the normalization and acceleration schemes briefly described in Section 4.3. The `sdf_nls` routine for NTF first applies a "squaring" transformation to the problem variables [24] and then solves an unconstrained problem via an AOO-based Gauss-Newton method.

[3] We note that we omit the presentation of numerical results concerning the Nesterov-type algorithm for strictly convex problems [14] because it needs a much larger number of iterations for the solution of each MNLS problem than the Nesterov-type algorithm for strongly convex problems.

[4] For our experiments, we run Matlab 2014a on a MacBook Pro with a 2.5 GHz Intel Core i7 Intel processor and 16 GB RAM.

Table 4.1: Average, over 10 realizations, `cputime` and maximum relative factor error for Nesterov-based AO NTF, `sdf_nls`, and `parafac`, for true latent factors with i.i.d. entries

Size	F	σ_N^2	AO-Nesterov		`sdf_nls`		`parafac`	
			`cputime`	`MRFE` $\times 10^4$	`cputime`	`MRFE` $\times 10^4$	`cputime`	`MRFE` $\times 10^4$
$1000 \times 100 \times 100$	15	10^{-2}	20	80	56	79	35	85
		10^{-4}	21	12	52	13	53	8
	50	10^{-2}	49	89	220	91	191	91
		10^{-4}	49	15	227	23	234	9
$500 \times 500 \times 100$	15	10^{-2}	56	37	135	37	68	41
		10^{-4}	55	9	137	10	100	4
	50	10^{-2}	103	41	341	44	259	44
		10^{-4}	105	10	341	18	344	4
$300 \times 300 \times 300$	15	10^{-2}	57	29	82	27	70	35
		10^{-4}	56	10	78	7	100	3
	50	10^{-2}	100	31	176	32	219	34
		10^{-4}	106	10	186	14	279	4

Table 4.2: Average, over 10 realizations, `cputime` and maximum relative factor error for Nesterov-based AO NTF, `sdf_nls`, and `parafac`, for true latent factors with correlated entries

Size	F	σ_N^2	Bottleneck	AO-Nesterov		`sdf_nls`		`parafac`	
				`cputime`	`MRFE` $\times 10^4$	`cputime`	`MRFE` $\times 10^4$	`cputime`	`MRFE` $\times 10^4$
$300 \times 300 \times 300$	50	10^{-4}	$\mathbf{A}$	106	73	189	79	340	69
			$\mathbf{A}, \mathbf{B}$	168	125	268	188	412	129
			$\mathbf{A}, \mathbf{B}, \mathbf{C}$	207	254	412	859	933	156

In our experiments with synthetic data, we focus on the `cputime` and the Maximum, over the three latent factors, Relative Factor Error (MRFE), which is computed via function `cpd_err` of tensorlab.

In all the experimental results we shall present in the sequel, the terminating conditions are determined by the parameter values Tol $= 10^{-5}$ for `parafac`, TolFun $= 10^{-9}$ for `sdf_nls`, and tol $= 10^{-2}$ and $\text{tol}_{\text{AO}} = 10^{-4}$, for the Nesterov-based AO NTF. These values have been chosen such that the resulting average MRFEs are approximately the same for all algorithms; of course, this is not always possible with one set of parameter values.

Parameter λ of our AO NTF algorithm is adjusted based on the condition number of

Table 4.3: `cputime` and relative factorization error for Nesterov-based AO NTF, `sdf_nls`, and `parafac`, for real-world data

Size	F	AO-Nesterov		`sdf_nls`		`parafac`	
		`cputime`	`RFE`	`cputime`	`RFE`	`cputime`	`RFE`
$1021 \times 1343 \times 33$	10	143	0.2349	2600	0.2422	130	0.2361
	20	244	0.1750	2330	0.2267	352	0.1744
	30	512	0.1445	2414	0.2180	699	0.1441

matrix $\mathbf{B}^T\mathbf{B}$ of each MNLS problem, $\mathcal{K}' := \frac{L'}{\mu'}$. More specifically, we set

$$\lambda = \begin{cases} 10^{-1.5}, & \text{if } \mathcal{K}' < 10^4, \\ 10^{-1}, & \text{if } 10^4 < \mathcal{K}' < 10^6, \\ 10^{0}, & \text{if } 10^6 < \mathcal{K}'. \end{cases} \tag{4.11}$$

All algorithms start from the same triple of random matrices, $\mathbf{A}_0$, $\mathbf{B}_0$, and $\mathbf{C}_0$, which have independent and identically distributed (i.i.d.) elements, uniformly distributed in $[0, 1]$.

4.4.1 True latent factors with i.i.d. elements

We start with synthetic data by assuming that the true latent factors consist of i.i.d. elements, uniformly distributed in $[0, 1]$. The additive noise is zero-mean white Gaussian with variance σ_N^2.

In Table 4.1, we present the average, over 10 realizations, `cputime` and MRFE for various tensor "shapes," ranks $F = 15, 50$, and noise variances $\sigma_N^2 = 10^{-2}, 10^{-4}$. We observe that the Nesterov-based AO NTF is very competitive in all cases, in the sense that it converges fast, achieving very good accuracy in most of the cases.

4.4.2 True latent factors with correlated elements

It is well-known that, if some columns of (at least) one latent factor are almost collinear, then convergence of the AO algorithm tends to be slow (these cases are known as "bottlenecks") [25]. In the sequel, we test the behavior of the three algorithms in cases with one, two, and three bottlenecks. More specifically, we generate the true latent factors with i.i.d. elements as before and we create a single "bottleneck" by modifying the last two columns of one latent factor so that each become highly correlated with another column

of the same latent factor (the correlation coefficient is larger that 0.98). In an analogous way, we generate double and triple "bottlenecks."

In Table 4.2, we focus on the case $I = J = K = 300$, $F = 50$, $\sigma_N^2 = 10^{-4}$, and present the average, over 10 realizations, `cputime` and MRFE. We observe that the problems become more difficult as the number of bottlenecks increases, in the sense that both the `cputime` and the MRFE increase as the number of bottlenecks increases. Again, the Nesterov-based AO NTF algorithm is very efficient in all cases. Analogous observations have been made in extensive numerical experiments with other tensor shapes and noise levels.

4.4.3 Real-world data

In order to test the behavior of the aforementioned algorithms with real-world data, we use the tensor with size $1021 \times 1343 \times 33$ derived from the hyperspectral image "Souto_Wood_Pile" [26]. Since, in this case, there are no true latent factors known, we focus on the `cputime` and the Relative Factorization Error (RFE), defined as

$$\texttt{RFE}(\mathbf{A}, \mathbf{B}, \mathbf{C}) := \frac{\|\boldsymbol{\mathcal{X}} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]\|_F}{\|\boldsymbol{\mathcal{X}}\|_F}. \tag{4.12}$$

In Table 4.3, we present the average `cputime` and `RFE` for ranks $F = 10, 20, 30$. The averages are with respect to the initial points $(\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0)$, which are random with i.i.d. elements uniformly distributed in $[0, 1]$, and are computed over 5 realizations. We observe that the Nesterov-based AO NTF is very efficient in these cases as well.

4.4.4 Overmodeling

By "Overmodeling" we name the case where the true rank of the tensor is smaller than the assumed rank F (which is a parameter of the algorithm). In this case, after some iterations, the factors tent to have linear dependent columns. This makes the condition number very big since the μ parameter is zero. Our algorithm, thanks to the proximal term, is tolerant in such problems. In Figures 4.1 and 4.2, we see the inner iteration count for a exact rank and overmodeling problem.

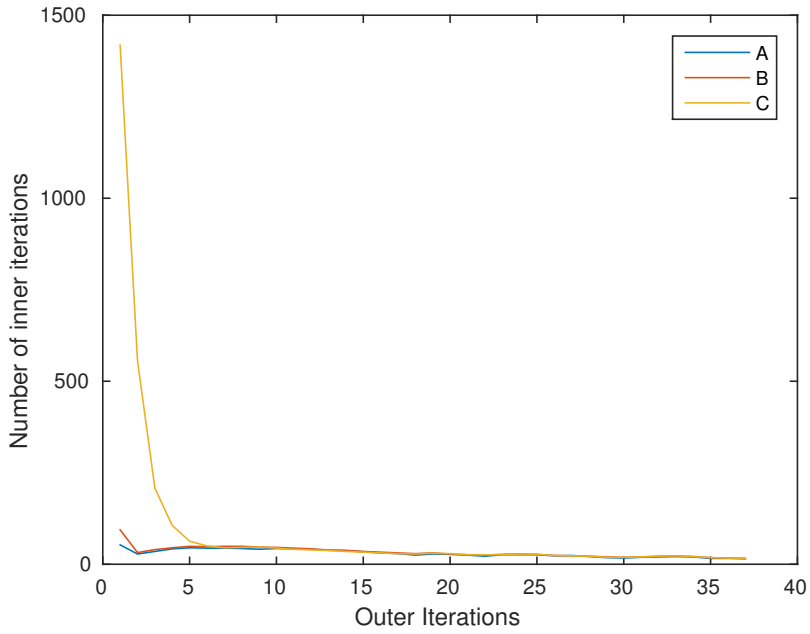

Figure 4.1: The inner versus the outer iterations of Nesterov-based AO-NTF for tensor size $300 \times 300 \times 300$ with $F = 10$ exact rank.

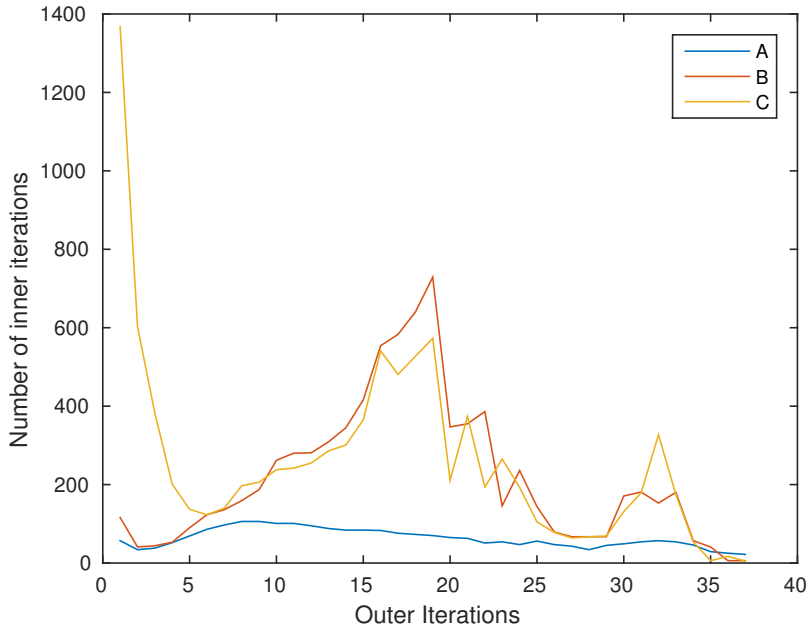

Figure 4.2: The inner versus the outer iterations of Nesterov-based AO-NTF for tensor size $300 \times 300 \times 300$ with $F = 10 + 2$ overmodeling.

Chapter 5

Parallel Nonnegative Tensor Factorization

5.1 Parallel implementation of AO NTF

In the sequel, we consider the implementation of the Nesterov-based AO NTF algorithm on a system with $N_p \geq 1$ processing elements, and describe two variations, with and without data replication. For our exposition, we use a pseudo-language, which we hope is able to highlight the basic algorithmic and communication requirements of the algorithms.

Data replication arises if we generate the tensor matricizations and, then, appropriately distribute their parts among the processing elements. On the other hand, data replication is avoided if we partition the tensor and appropriately distribute its parts among the processing elements. From a memory usage perspective, it seems preferable to adopt the latter approach. However, as we shall see in the sequel, in this case, the communication cost is slightly increased.

We start by partitioning matrices $\mathbf{A}_{k+1}$, $\mathbf{B}_{k+1}$, and $\mathbf{C}_{k+1}$ in block-rows as

$$\mathbf{A}_{k+1} = \left[\begin{array}{ccc} \left(\mathbf{A}_{k+1}^{1}\right)^T & \cdots & \left(\mathbf{A}_{k+1}^{N_p}\right)^T \end{array} \right]^T, \tag{5.1}$$

with $\mathbf{A}_{k+1}^{n} \in \mathbb{R}^{\frac{I}{N_p} \times F}$, for $n = 1, \ldots, N_p$, and analogous partitionings for $\mathbf{B}_{k+1}$ and $\mathbf{C}_{k+1}$. In both variations we shall describe, the n-th processing element computes the n-th block row of $\mathbf{A}_{k+1}$, $\mathbf{B}_{k+1}$, and $\mathbf{C}_{k+1}$, for $n = 1, \ldots, N_p$.

We note that, in order to keep the presentation simple and focus on the most important algorithmic aspects, in the Algorithms we shall present in the sequel, we do not include the details of the parallel implementation of the acceleration scheme. However, we shall briefly discuss this topic later.

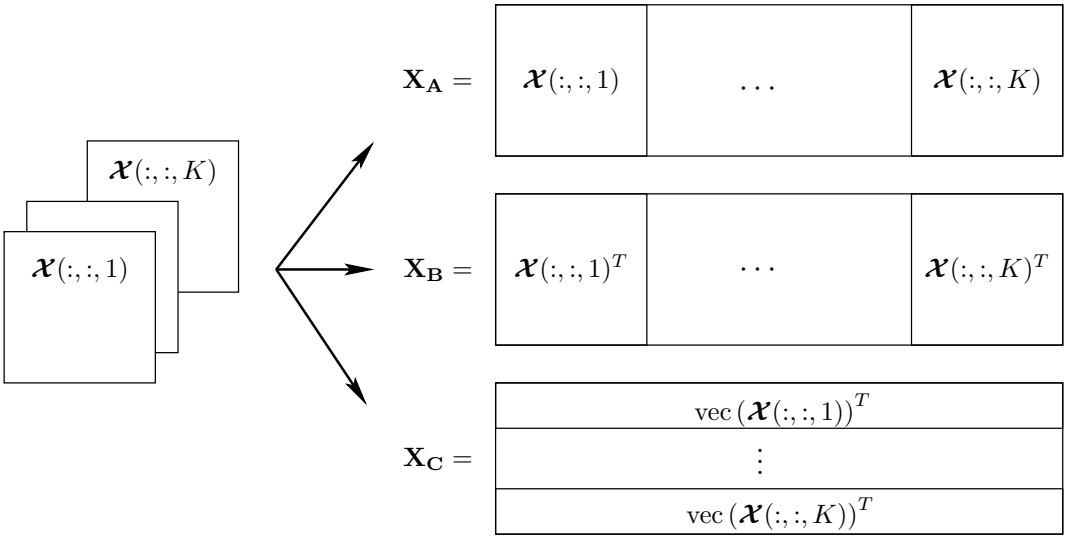

Figure 5.1: Tensor $\boldsymbol{\mathcal{X}}$ and its matricizations $\mathbf{X_A}$, $\mathbf{X_B}$, and $\mathbf{X_C}$, in terms of the frontal slices $\boldsymbol{\mathcal{X}}(:,:,k)$, for $k = 1, \ldots, K$.

5.1.1 Data replication

We create the matrix unfoldings of tensor $\boldsymbol{\mathcal{X}}$, $\mathbf{X_A}$, $\mathbf{X_B}$, and $\mathbf{X_C}$, and partition each of them into N_p block-rows of the same size, that is,

$$\mathbf{X_A} = \left[\begin{array}{ccc} \left(\mathbf{X_A^1}\right)^T & \cdots & \left(\mathbf{X_A^{N_p}}\right)^T \end{array} \right]^T, \tag{5.2}$$

with $\mathbf{X_A^n} \in \mathbb{R}^{\frac{I}{N_p} \times JK}$, for $n = 1, \ldots, N_p$, and analogous partitionings for $\mathbf{X_B}$ and $\mathbf{X_C}$. Then, we allocate blocks $\mathbf{X_A^n}$, $\mathbf{X_B^n}$, and $\mathbf{X_C^n}$ to the n-th processing element, for $n = 1, \ldots, N_p$.

Under these assumptions, a parallel implementation with data replication of Algorithm 4 is given in Algorithm 5. The algorithm proceeds as follows. All processing elements work in parallel. The n-th processing element uses its local data $\mathbf{X_A^n}$, as well as the whole matrices $\mathbf{B}_k$ and $\mathbf{C}_k$, and computes the n-th block-row of matrix $\mathbf{A}_{k+1}$, $\mathbf{A}_{k+1}^n$. Then, each processing element broadcasts its output to all other processing elements; this operation can be implemented via the MPI statement `MPI_Allgather`. At the end of this step, all processing elements possess the whole new $\mathbf{A}_{k+1}$. Then, we compute $\mathbf{B}_{k+1}$ and $\mathbf{C}_{k+1}$ with analogous computations, completing one iteration of the AO NTF algorithm. The algorithm continues until convergence.

The communication requirements of this implementation consist of one `Allgather` operation per MNLS problem, implying gathering of terms with IF, JF, and KF elements

per outer iteration.

Algorithm 5: Parallel Nesterov-based AO NTF with data replication

Input: Processing element n, for $n = 1, \ldots, N_p$, knows $\mathbf{X}^n_{\mathbf{A}}$, $\mathbf{X}^n_{\mathbf{B}}$, $\mathbf{X}^n_{\mathbf{C}}$, $\mathbf{A}_0$, $\mathbf{B}_0$, $\mathbf{C}_0$, λ, tol.

Set $k = 0$
while (terminating condition is FALSE) **do**
  In parallel, for $n = 1, \ldots, N_p$, **do**
    $\mathbf{W}^n_{\mathbf{A}} = -\mathbf{X}^n_{\mathbf{A}}(\mathbf{C}_k \odot \mathbf{B}_k) - \lambda \mathbf{A}^n_k$, $\mathbf{Z}_{\mathbf{A}} = (\mathbf{C}_k \odot \mathbf{B}_k)^T(\mathbf{C}_k \odot \mathbf{B}_k) + \lambda \mathbf{I}$
    $\mathbf{A}^n_{k+1} = \text{Nesterov\_MNLS}(\mathbf{W}^n_{\mathbf{A}}, \mathbf{Z}_{\mathbf{A}}, \mathbf{A}^n_k, \lambda, \text{tol})$
    All_gather($\mathbf{A}^n_{k+1}$)
  In parallel, for $n = 1, \ldots, N_p$, **do**
    $\mathbf{W}^n_{\mathbf{B}} = -\mathbf{X}^n_{\mathbf{B}}(\mathbf{C}_k \odot \mathbf{A}_{k+1}) - \lambda \mathbf{B}^n_k$, $\mathbf{Z}_{\mathbf{B}} = (\mathbf{C}_k \odot \mathbf{A}_{k+1})^T(\mathbf{C}_k \odot \mathbf{A}_{k+1}) + \lambda \mathbf{I}$
    $\mathbf{B}^n_{k+1} = \text{Nesterov\_MNLS}(\mathbf{W}^n_{\mathbf{B}}, \mathbf{Z}_{\mathbf{B}}, \mathbf{B}^n_k, \lambda, \text{tol})$
    All_gather($\mathbf{B}^n_{k+1}$)
  In parallel, for $n = 1, \ldots, N_p$, **do**
    $\mathbf{W}^n_{\mathbf{C}} = -\mathbf{X}^n_{\mathbf{C}}(\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1}) - \lambda \mathbf{C}^n_k$, $\mathbf{Z}_{\mathbf{C}} = (\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1})^T(\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1}) + \lambda \mathbf{I}$
    $\mathbf{C}^n_{k+1} = \text{Nesterov\_MNLS}(\mathbf{W}^n_{\mathbf{C}}, \mathbf{Z}_{\mathbf{C}}, \mathbf{C}^n_k, \lambda, \text{tol})$
    All_gather($\mathbf{C}^n_{k+1}$)
  $k = k + 1$
return $\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{C}_k$.

5.1.2 No data replication

We partition tensor $\boldsymbol{\mathcal{X}}$ and distribute its parts among the N_p processing elements, avoiding data replication (see, also, [9]). An interesting question is which partitioning of the tensor leads to the most (memory- and/or computation- and/or communication-) efficient implementation? In our experience, the answer to this question is not clear beforehand, as it depends on the tensor size and the computing environment, among other factors.

In the sequel, we describe an implementation where we assume that the third tensor dimension, K, is large enough[1] and allocate to the n-th processing element $\frac{K}{N_p}$ successive frontal slices of tensor $\boldsymbol{\mathcal{X}}$, that is, $\boldsymbol{\mathcal{X}}(:, :, (n-1) * K/N_p + 1 : n * K/N_p)$, for $n = 1, \ldots, N_p$. This implies that the n-th processing element contains $\frac{K}{N_p}$ successive blocks of (block) matrices $\mathbf{X}_{\mathbf{A}}$ and $\mathbf{X}_{\mathbf{B}}$, and $\frac{K}{N_p}$ successive rows of (block) matrix $\mathbf{X}_{\mathbf{C}}$ (see Figure 5.1).

Under these assumptions, a parallel implementation of Algorithm 4 with no data replication is given in Algorithm 6. We recall that the n-th processing element computes $\mathbf{A}^n_{k+1}$,

[1] Of course, we can reorder the tensor modes if necessary to bring the longest dimension in the third mode.

$\mathbf{B}_{k+1}^n$, and $\mathbf{C}_{k+1}^n$, for $n = 1, \ldots, N_p$. Towards this end, it needs quantities $\mathbf{X}_{\mathbf{A}}^n(\mathbf{C}_k \odot \mathbf{B}_k)$, $\mathbf{X}_{\mathbf{B}}^n(\mathbf{C}_k \odot \mathbf{A}_{k+1})$, and $\mathbf{X}_{\mathbf{C}}^n(\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1})$. Based on the assumed data allocation, the n-th processing element

1. computes partial sums related with $\mathbf{X}_{\mathbf{A}}(\mathbf{C}_k \odot \mathbf{B}_k)$ and $\mathbf{X}_{\mathbf{B}}(\mathbf{C}_k \odot \mathbf{A}_{k+1})$. These partial sums must be added and the results must be appropriately scattered to the processing elements. These operations can be implemented via the MPI statement `MPI_Reduce_scatter`;

2. computes the n-th block-row of $\mathbf{X}_{\mathbf{C}}(\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1})$, that is, $\mathbf{X}_{\mathbf{C}}^n(\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1})$. In this case, no further operation is necessary.

The communication cost per outer iteration consists of (i) three `All_gather` operations, one for each MNLS problem, which is exactly the same as in the first implementation, and (ii) two `Reduce_scatter` operations of matrices with IF and JF elements, respectively.

Algorithm 6: Parallel Nesterov-based AO NTF with no data replication

Input: Processing element n, for $n = 1, \ldots, N_p$, knows $\boldsymbol{\mathcal{X}}(:, :, (n-1) * K/N_p + 1 : n * K/N_p)$, $\mathbf{A}_0$, $\mathbf{B}_0$, $\mathbf{C}_0$, λ, tol.

Set $k = 0$
while (terminating condition is FALSE) **do**
  In parallel, for $n = 1, \ldots, N_p$, **do**
    Compute n-th partial sum of $\mathbf{W}_{\mathbf{A}} = -\mathbf{X}_{\mathbf{A}}(\mathbf{C}_k \odot \mathbf{B}_k) - \lambda \mathbf{A}_k$, $\mathbf{Z}_{\mathbf{A}} = (\mathbf{C}_k \odot \mathbf{B}_k)^T(\mathbf{C}_k \odot \mathbf{B}_k) + \lambda \mathbf{I}$
    Reduce_scatter($\mathbf{W}_{\mathbf{A}}$)
    $\mathbf{A}_{k+1}^n = \text{Nesterov\_MNLS}(\mathbf{W}_{\mathbf{A}}^n, \mathbf{Z}_{\mathbf{A}}, \mathbf{A}_k^n, \lambda, \text{tol})$
    All_gather($\mathbf{A}_{k+1}^n$)
  In parallel, for $n = 1, \ldots, N_p$, **do**
    Compute n-th partial sum of $\mathbf{W}_{\mathbf{B}} = -\mathbf{X}_{\mathbf{B}}(\mathbf{C}_k \odot \mathbf{A}_{k+1}) - \lambda \mathbf{B}_k$, $\mathbf{Z}_{\mathbf{B}} = (\mathbf{C}_k \odot \mathbf{A}_{k+1})^T(\mathbf{C}_k \odot \mathbf{A}_{k+1}) + \lambda \mathbf{I}$
    Reduce_scatter($\mathbf{W}_{\mathbf{B}}$)
    $\mathbf{B}_{k+1}^n = \text{Nesterov\_MNLS}(\mathbf{W}_{\mathbf{B}}^n, \mathbf{Z}_{\mathbf{B}}, \mathbf{B}_k^n, \lambda, \text{tol})$
    All_gather($\mathbf{B}_{k+1}^n$)
  In parallel, for $n = 1, \ldots, N_p$, **do**
    $\mathbf{W}_{\mathbf{C}}^n = -\mathbf{X}_{\mathbf{C}}^n(\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1}) - \lambda \mathbf{C}_k^n$, $\mathbf{Z}_{\mathbf{C}} = (\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1})^T(\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1}) + \lambda \mathbf{I}$
    $\mathbf{C}_{k+1}^n = \text{Nesterov\_MNLS}(\mathbf{W}_{\mathbf{C}}^n, \mathbf{Z}_{\mathbf{C}}, \mathbf{C}_k^n, \lambda, \text{tol})$
    All_gather($\mathbf{C}_{k+1}^n$)
  $k = k + 1$
return $\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{C}_k$.

5.1.3 Acceleration scheme

In both parallel implementation variations we described, after the $(k+1)$-st AO iteration, each processing element knows $\mathbf{A}_{k+1}$, $\mathbf{B}_{k+1}$, $\mathbf{C}_{k+1}$, as well as $\mathbf{A}_{\text{old}}$, $\mathbf{B}_{\text{old}}$, and $\mathbf{C}_{\text{old}}$, and can compute $\mathbf{A}_{\text{new}}$, $\mathbf{B}_{\text{new}}$, and $\mathbf{C}_{\text{new}}$ (see (4.9)). The computation of the cost function $f_{\boldsymbol{\mathcal{X}}}$ at points $(\mathbf{A}_{k+1}, \mathbf{B}_{k+1}, \mathbf{C}_{k+1})$ and $(\mathbf{A}_{\text{new}}, \mathbf{B}_{\text{new}}, \mathbf{C}_{\text{new}})$ is implemented collaboratively. More specifically, each processing element computes its "local contribution" to the cost function, and then, via an `All_reduce` statement, the value of the cost function becomes known to all processing elements. Based, on the values of the cost function at the two aforementioned points, all processing elements make the same decision about the success or failure of the acceleration step.

5.1.4 Experimental results - MPI

In this subsection, we present results obtained from the MPI programs implementing the variations described in section 5.1.

The programs are executed on a CentOS system with four AMD Opteron(tm) Processor 6376 (in total, 64 cores at 2.3 Gz) and 500 GB RAM. For many matrix operations, we use routines of the GSL library.

We assume a noiseless tensor $\boldsymbol{\mathcal{X}}$, whose true latent factors have i.i.d elements, uniformly distributed in $[0, 1]$. For the variation with data replication, we construct the matricizations, $\mathbf{X}_{\mathbf{A}}$, $\mathbf{X}_{\mathbf{B}}$, and $\mathbf{X}_{\mathbf{C}}$, and appropriately allocate their block-rows to the N_p processing elements. For the variation without data replication, we partition $\boldsymbol{\mathcal{X}}$, in terms of its frontal slices, and appropriately allocate the parts to the N_p processing elements. The timer that counts the program execution duration starts counting *after* the data allocation to the processing elements.

In Figure 5.2, we plot the achieved speedup for tensor with $I = J = K = 1000$, rank $F = 15, 30, 50$, and parameters $\text{tol} = 10^{-2}$, and $\text{tol}_{\text{AO}} = 10^{-4}$, for the variation with data replication (denoted as DR) and the variation with no data replication (denoted as NDR), respectively. We observe that

1. in all cases, we attain significant speedup;

2. for fixed rank, we attain slightly higher speedup with data replication; this happens because the communication cost is lower in this case;

3. for both variations, the speedup decreases for increasing rank, due to increased communication cost.

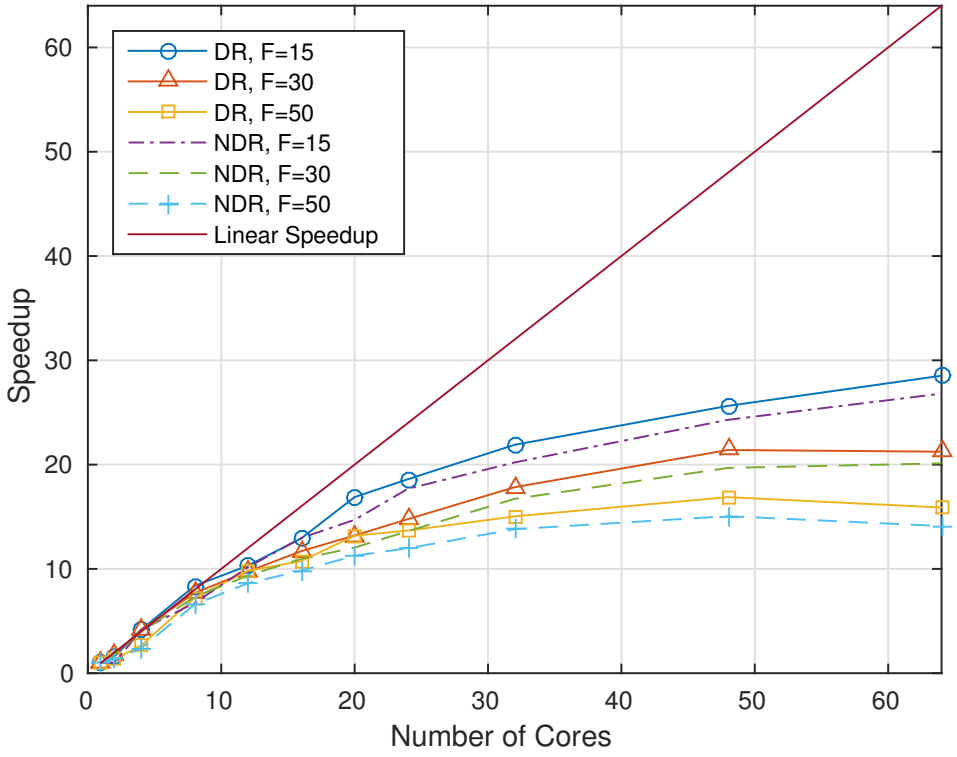

Figure 5.2: Speedup diagram of the Nesterov-based AO NTF algorithm, for rank $F = 15, 30, 50$, with data replication (DR) and with no data replication (NDR). The tensor size is $1000 \times 1000 \times 1000$ (8 Gb).

Chapter 6

Discussion and Future Work

6.1 Conclusion

We considered the NTF problem. We adopted the AO framework and solved each MNLS problem via a Nesterov-type algorithm for strongly convex problems. In extensive numerical experiments, the derived algorithm was proven very efficient. We presented two parallel implementations, with and without data replication, and measured the attained speedup for large-scale dense NTF problems. We concluded that the proposed algorithm is a strong alternative to state-of-the-art algorithms for the solution of very large-scale dense NTF problems.

6.2 Future work

In recent years, NTF is one of the tools for analysing nonnegative multi-way data. We suggest some future extensions in our work.

6.2.1 Higher-order tensors

We can extend and test our algorithms to higher order tensors ($N > 3$). The idea is the same but some of the details may change. We must find efficient ways to do the computations and avoid duplicated computations. The Parallel implementation may be prove useful.

6.2.2 Tensor completion

One other of the immediate extensions of NTF is working with tensors with missing data. We can test our algorithm in the case of missing elements. If the right factors are retrieved then we can find the missing elements easily. The completion problem for a 3-order tensor

can be formulated as follows

$$\begin{aligned} f_{\mathcal{W}}(\mathbf{A},\mathbf{B},\mathbf{C}) &:= \frac{1}{2}\|\mathcal{W} * (\mathcal{X} - [\mathbf{A},\mathbf{B},\mathbf{C}])\|_F^2 \\ &= \frac{1}{2}\|\mathbf{W}_A * (\mathbf{X}_A - \mathbf{A}(\mathbf{C} \odot \mathbf{B})^T)\|_F^2 \end{aligned} \tag{6.1}$$

where $\mathcal{W}$ is the the indicator tensor where

$$\mathcal{W}(i,j,k) = \begin{cases} 1, & \text{if } \mathcal{X}(i,j,k) \text{ exists,} \\ 0, & \text{if } \mathcal{X}(i,j,k) \text{ is missing.} \end{cases}$$

6.2.3 Effective rank estimation

Another topic is the effective rank estimation of a tensor. We have observed that when we apply the proposed algorithm on a tensor and the true rank is smaller than the one we use in the algorithm, then the condition of the problem grows because of the correlation of the columns of factors. If we can detect that then we can propose an algorithm that finds the real rank of a tensor based on the data only.

Another way to detect the rank is to apply the decomposition with an $l1$ regularization over the rank.

6.2.4 Online NTF

Todays datasets are often dynamically changing over time. Tracking the CP decomposition for such dynamic tensors is a crucial but challenging task. We can extend our algorithm to an online version like the recently published [27]. Let us consider the main idea of online CP decomposition problem. A third-order tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times \{t_{\text{old}}+t_{\text{new}}\}}$ is used as an example, where $\mathcal{X}$ is expanded from $\mathcal{X}_{\text{old}} \in \mathbb{R}^{I \times J \times \{t_{\text{old}}\}}$ by appending a new chunk of data $\mathcal{X}_{\text{new}} \in \mathbb{R}^{I \times J \times \{t_{\text{new}}\}}$ in its last mode. The problem is to efficiently find the CP decomposition of $\mathcal{X}$ by knowing the factors of $\mathcal{X}_{\text{old}}$.

Bibliography

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

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

[3] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, pp. 455–500, September 2009.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] K. Huang, N. D. Sidiropoulos, and A. P. Liavas, "Efficient algorithms for 'universally' constrained matrix and tensor factorization," *IEEE Transactions on Signal Processing*, submitted 2016.

[9] L. Karlsson, D. Kressner, and A. Uschmajew, "Parallel algorithms for tensor completion in the cp format," *Parallel Computing*, pp. –, 2015.

[10] S. Smith and G. Karypis, "A medium-grained algorithm for distributed sparse tensor factorization," 2016.

[11] 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*, SC '15, 2015.

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

[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] G. Zhou, Q. Zhao, Y. Zhang, and A. Cichocki, "Fast nonnegative tensor factorization by using accelerated proximal gradient," in *Advances in Neural Networks – ISNN 2014* (Z. Zeng, Y. Li, and I. King, eds.), pp. 459–468, Cham: Springer International Publishing, 2014.

[15] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *arXiv preprint arXiv:1607.01668*, 2016.

[16] S. Bubeck, "Convex optimization: Algorithms and complexity," *Foundations and Trends in Machine Learning*, vol. 8, no. 3-4, pp. 231–357, 2015.

[17] S. Boyd and L. Vandenberghe, "Convex optimization," 2004.

[18] A. P. Liavas, G. Kostoulas, G. Lourakis, K. Huang, and N. D. Sidiropoulos, "Nesterov-based alternating optimization for nonnegative tensor factorization: Algorithm and parallel implementations," *submitted to the IEEE Trans. on Signal Processing*.

[19] 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.

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

[21] C. A. Andersson and R. Bro, "The n-way toolbox for matlab."

[22] L. Sorber, M. Van Barel, and L. De Lathauwer, "Tensorlab v2.0."

[23] C. L. Lawson and R. J. Hanson, *Solving Lease Squares Problems*. SIAM, 1995.

[24] J.-P. Royer, N. Thirion-Moreau, and P. Comon, "Computing the polyadic decomposition of nonnegative third order tensors," *Signal Processing*, vol. 91, no. 9, pp. 2159–2171, 2011.

[25] M. Rajih, P. Comon, and R. A. Harshman, "Enhanced line search: A novel method to accelerate parafac," *SIAM journal on matrix analysis and applications*, vol. 30, no. 3, pp. 1128–1147, 2008.

[26] S. M. Nascimento, K. Amano, and D. H. Foster, "Spatial distributions of local illumination color in natural scenes," *Vision research*, vol. 120, pp. 39–44, 2016.

[27] S. Zhou, N. X. Vinh, J. Bailey, Y. Jia, and I. Davidson, "Accelerating online cp decompositions for higher order tensors," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 1375–1384, ACM, 2016.