TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Stochastic Optimization on Tensor Factorization and Completion

by

Ioanna Siaminou

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

ELECTRICAL AND COMPUTER ENGINEERING

1/3/2021

THESIS COMMITTEE

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

Abstract

We consider the problem of structured canonical polyadic decomposition (CPD). If the size of the problem is very big, then stochastic optimization approaches are viable alternatives to classical methods, such as Alternating Optimization (AO) and All-At-Once (AAO) optimization. We extend a recent stochastic gradient approach by employing an acceleration step (Nesterov momentum) in each iteration. We compare our approach with state-of-the-art alternatives, using both synthetic and real-world data, and find it to be very competitive. Furthermore, we examine the drawbacks of a parallel implementation of our accelerated stochastic algorithm and describe an alternative method that deals with these limitations. Finally, we propose an accelerated stochastic algorithm for the Nonnegative Tensor Completion problem and its parallel implementation via the shared–memory API OpenMP. Through numerical experiments, we test its efficiency in very large problems.

Acknowledgements

First, I would like to thank my thesis supervisor Prof. Athanasios Liavas for his continuous guidance throughout this thesis. Furthermore, I would like to thank the group members Giannis, Christos and Paris for their advice and help. Working with them was one of the most fruitful and productive experiences in my academic years. Last but not least, I would like to thank Giorgos for his continuous encouragement and support all this time. Finally, I must express my gratitude to my family, who were always there for me throughout my years of study. This thesis would not have been possible without them. Thank you.

Table of Contents

List of Figures

List of Abbreviations

ADMM	Alternating Direction Method of Multipliers
AGD	Accelerated Gradient Descent
ALS	Alternating Least Squares
AO	Alternating Optimization
AOO	All-at-Once Optimization
CANDECOMP	Canonical Decomposition
CPD	Canonical Polyadic Decomposition
i.i.d.	independent and identically distributed
NAG	Nesterov's Accelerated Gradient
MNLS	Matrix Nonnegative Least-Squares
MU	Multiplicative Updates
NMC	Nonnegative Matrix Completion
NTC	Nonnegative Tensor Completion
NTF	Nonnegative Tensor Factorization
PARAFAC	Parallel Factor Analysis
PGD	Projected Gradient Descent
RMSE	Root Mean Square Error
SGD	Stochastic Gradient Descent

Chapter 1

Introduction

Recently, the need to understand and analyze multi-dimensional data and their dependencies has led to the extended use of tensors, a multi-dimensional mathematical object, with high interpretability. Tensors, and tensor algebra are used in a variety of applications and scientific fields from medicine to geodesy. For instance, in cases where a matrix suffices to represent the relationship of the data at given point in time, tensors are used for temporal analysis of the data. For example, an fMRI scan can be represented by a 3–rd order tensor $\boldsymbol{\mathcal{X}}$, where its element corresponds to a specific part of the brain and its state in time [1]. An overview of tensor applications can be found in [2]. Canonical Tensor Decomposition (CPD) or Parallel Factor Analysis (PARAFAC) is a well known model to data scientists since it can extract meaningful structure from a given dataset. Alternating Optimization (AO), All-at-Once Optimization(AOO), and Multiplicative Updates (MUs) are the workhorse methods for CPD. In case of constrained tensor decomposition, especially the case of Nonnegative Tensor Factorization problem (NTF), a new scheme is proposed that makes use of momentum in [3]. In [4], a Gauss-Newton type algorithm is employed to solve NTF.

However, due to the information explosion and the continuous data collection, CPD of very large tensors may be prohibitive. In the literature, different approaches are introduced in order to deal with large-scale problems in the context of CPD. Distributed programming and shared memory APIs are extensively used to retain the scalabitity of the CPD model. Recently, randomized techniques have gained much attention, since they are relatively easy to implement, have lower computational cost and can guarantee accurate solutions.

Tensor Factorization with missing elements or Tensor Completion is another interesting application of the CPD model. In this setting, a set of observations is organized in a N–mode tensor $\boldsymbol{\mathcal{X}}$. Using only the available elements, accurate estimations for the missing ones can be extracted. Recommendation systems which are heavily used in ML applications, are related to Matrix/Tensor Completion problems. We highlight the Netflix Prize competition where the efficiency of Matrix Completion was shown [5].

1.1 Notation

Throughout this thesis, vectors and matrices are denoted by lowercase and uppercase bold letters, for example $\mathbf{x}$ and $\mathbf{X}$. Tensors are denoted by calligraphic capital letters, namely $\boldsymbol{\mathcal{X}}$. $\|\cdot\|_F$ denotes the Frobenius norm of the matrix or tensor argument. The identity matrix is denoted by the letter $\mathbf{I}$. The sets $\mathbb{R}^{(\prod_{n=1}^{N} I_n)}$ and $\mathbb{R}_{+}^{(\prod_{n=1}^{N} I_n)}$ denote the sets of N-th order real and real nonnegative tensors, respectively. The inequality $\mathbf{A} - \mathbf{B} \succeq \mathbf{0}$

means that the matrix $\mathbf{A} - \mathbf{B}$ is positive semidefinite. Finally, MATLAB notation is used when it seems appropriate.

Chapter 2

Mathematical Background

We proceed with some definitions in the context of tensors and the CPD model. For further details we refer the reader to [6] and [7]. Finally, we make an introduction to Stochastic Gradient Descent (SGD), an optimization algorithm heavily used in Machine Learning field [8], [9].

2.1 Tensor Preliminaries

Definition A.1 *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*

$$[\mathbf{a} \circ \mathbf{b}]_{n,p} = a_n b_p, \tag{2.1}$$

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

$$[\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}]_{n,p,j} = a_n b_p c_j, \tag{2.2}$$

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

Definition A.2 *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*

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} A_{1,1}\mathbf{B} & \cdots & A_{1,M}\mathbf{B} \\ \vdots & \ddots & \vdots \\ A_{N,1}\mathbf{B} & \cdots & A_{N,M}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{NP \times MK}. \tag{2.3}$$

Definition A.3 *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*

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{A}_{:,1} \otimes \mathbf{B}_{:,1} & \cdots & \mathbf{A}_{:,M} \otimes \mathbf{B}_{:,M} \end{bmatrix} \in \mathbb{R}^{NP \times M}. \tag{2.4}$$

Definition A.4 *Let* $\mathbf{A} \in \mathbb{R}^{N \times M}$ *and* $\mathbf{B} \in \mathbb{R}^{N \times M}$. *The* ***Hadamard Product*** *or elementwise matrix product of* $\mathbf{A}$ *and* $\mathbf{B}$, *is a matrix of size* $N \times M$, *and is defined as*

$$[\mathbf{A} \circledast \mathbf{B}]_{n,m} = a_{n,m} b_{n,m}, \tag{2.5}$$

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

Definition A.5 *The n-mode* ***unfolding or matricization*** *of a tensor* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{(\prod_{n=1}^{N} I_n)}$,

is a $\prod_{m=1,m\neq n}^{N} I_m \times I_n$ *matrix where*

$$\mathbf{X}_{(n)}(j, i_n) = \boldsymbol{\mathcal{X}}(i_1, i_2, \ldots, i_N), \tag{2.6}$$

where, $j = 1 + \sum_{k=1,k\neq n}^{N}(i_k - 1)J_k$ *and* $J_k = \prod_{m=1,\ m\neq n}^{k-1} I_m$.

2.1.1 Canonical Polyadic Decomposition (PARAFAC model)

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

$$\boldsymbol{\mathcal{X}}^o = [\![\mathbf{A}^{o(1)}, \ldots, \mathbf{A}^{o(N)}]\!] = \sum_{r=1}^{R} \mathbf{a}_r^{o(1)} \circ \cdots \circ \mathbf{a}_r^{o(N)}, \tag{2.7}$$

where $\mathbf{A}^{o(i)} = [\mathbf{a}_1^{o(i)} \ \cdots \ \mathbf{a}_R^{o(i)}] \in \mathbb{R}^{I_i \times R}$, 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 $\mathbf{A}^{o(i)}$ can be obtained by computing matrices $\mathbf{A}^{(i)} \in \mathbb{R}^{I_i \times R}$, for $i \in \{1, \ldots, N\}$, that solve the optimization problem

$$\min_{\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}} f_{\boldsymbol{\mathcal{X}}}\left(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}\right), \tag{2.8}$$

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

$$f_{\boldsymbol{\mathcal{X}}}\left(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}\right) = \left\| \boldsymbol{\mathcal{X}} - [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!] \right\|_F^2. \tag{2.9}$$

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

$$\widehat{\mathbf{X}}_{(i)} = \mathbf{K}^{(i)} \mathbf{A}^{(i)T}. \tag{2.10}$$

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

$$\mathbf{K}^{(i)} = \mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(i+1)} \odot \mathbf{A}^{(i-1)} \odot \cdots \odot \mathbf{A}^{(1)}. \tag{2.11}$$

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

$$f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}) = \left\| \mathbf{X}_{(i)} - \widehat{\mathbf{X}}_{(i)} \right\|_F^2. \tag{2.12}$$

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

2.1.2 ALS algorithm

The ALS algorithm is the workhorse method for problem (2.8). At each iteration, ALS solves

$$\mathbf{A}^{(i)} = \underset{\mathbf{A}}{\operatorname{argmin}} \ \|\mathbf{X}_{(i)} - \mathbf{K}^{(i)} \mathbf{A}^T\|_F^2, \tag{2.13}$$

in a cyclical manner for $i = 1, 2, \dots N$.

In the unconstrained case, there is a closed form solution to the problem (2.13). Specifically, if $\mathbf{K}^{(i)}$ is full rank, we have

$$\mathbf{A}^{(i)} = \left(\left(\mathbf{K}^{(i)T} \mathbf{K}^{(i)} \right)^{-1} \mathbf{K}^{(i)T} \mathbf{X}_{(i)} \right)^T. \tag{2.14}$$

Note that $\mathbf{K}^{(i)T} \mathbf{K}^{(i)} = *_{n=1, n \neq i}^{N} \mathbf{A}^{(n)T} \mathbf{A}^{(n)}$.

The product of the last two terms in (2.14) is known as matricized tensor times Khatri-Rao product (MTTKRP) and is the main computational bottleneck of the ALS algorithm. The MTTKRP costs $\mathcal{O}(\prod_{n=1}^{N} I_n R)$ operations which can be very large even if the dimensions are quite small.

2.2 Stochastic Optimization

In this section, we briefly discuss the Stochastic Gradient Descent algorithm (SGD). SGD gained attention in the recent years due to its efficiency in large scale problems. More specifically, SGD alleviates the computational cost at each iteration, by employing an estimate of the gradient of the cost function $f(\cdot)$ instead of computing the true one. In case of a decomposable cost function $f : \mathbb{R}^N \to \mathbb{R}$, namely

$$f(\mathbf{x}) = \frac{1}{m} \sum_{i=1}^{m} f_i(\mathbf{x}), \tag{2.15}$$

SGD solves the optimization problem

$$\min_{\mathbf{x} \in \mathbb{R}^N} \frac{1}{m} \sum_{i=1}^{m} f_i(\mathbf{x}), \tag{2.16}$$

by performing one call to the stochastic oracle at each iteration of the algorithm. Given an initial point $\mathbf{x}_0 \in \mathbb{R}^N$ the stochastic gradient step is defined by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f_{i_k}(\mathbf{x}_k), \tag{2.17}$$

where i_k is chosen randomly from the set $\{1, 2, \dots, m\}$, according to a probability distribution. Thus, the sequence $\{\mathbf{x}\}$ is a stochastic process.

We remind the reader that the estimate of the gradient $-\nabla_{i_k} f(\mathbf{x}_k)$[1] may not be a descent direction of $f(\mathbf{x}_k)$. In some cases, the optimum can be attained if $-\nabla_{i_k} f(\mathbf{x}_k)$ is a direction of descent in expectation. Note that problem (2.16), is referred as Empirical Risk Minimization in Machine Learning literature.

[1]The estimates $-\nabla_{i_k} f(\mathbf{x}_k)$ are referred as noisy estimates of the full gradient, rather than a single component of it.

2.2.1 Convergence Results

We continue with some theoretical results for SGD algorithm applied on cost functions with different properties. We also refer to the analogous results of GD (full batch GD in this context) for the sake of comparison. For the case of L–smooth and μ–strongly convex function $f(\cdot)$, via full-batch GD, the error between the value of $f(\cdot)$ at iteration k, with the optimal value $\mathbf{x}^*$ satisfies [10], [9]

$$f(\mathbf{x}_k) - f(\mathbf{x}_*) \leq \mathcal{O}(\rho^k), \tag{2.18}$$

where $\rho \in (0, 1)$. The result (2.18) is commonly known as *linear convergence*. To attain accuracy of ϵ, with $\epsilon > 0$, one has to perform $\mathcal{O}(\log(1/\epsilon))$ iterations with total computational cost of $\mathcal{O}(m \log(1/\epsilon))$. For the SGD algorithm, for the same class of functions $f(\cdot)$, we have

$$\mathbb{E}\left[f(\mathbf{x}_k) - f(\mathbf{x}_*)\right] \leq \mathcal{O}(1/k), \tag{2.19}$$

which is usually called *sublinear convergence*. In a similar manner, the number of iterations required for ϵ–optimality is $\mathcal{O}(1/\epsilon)$. Although, the number of iterations required via SGD may be greater than its full-batch counterpart, in case of big data (m very large), the total cost $\mathcal{O}(m \log(1/\epsilon))$ may be prohibitive.

2.2.2 Determination of Stepsize

The sequence of stepsizes $\{\alpha_k\}$ in (2.17) is often challenging to determine. Various schemes are proposed, from constant to diminishing stepsizes at every iteration. Although there are convergence results concerning both fixed and diminishing stepsizes, a combination of the two is commonly used in practice. In most cases, the mechanism for computing the stepsize α_k employs all the knowledge for the current problem. In Figure 2.1, the performance of SGD for two different stepsizes $\{\alpha_k\}$ is illustrated.

2.2.3 Variants of SGD

We briefly discuss the most notable variants of SGD in the optimization and machine learning literature. Most of the variants can be classified either as noise reduction SGD or stochastic second order methods. Initially, we mention a noise reduction method, the Mini–batch algorithm. Through the name, one can quickly deduce that mini-batch lies in between SGD and full–batch method. Specifically, a mini-batch approach employs small, randomly selected sets $\mathcal{S}_k \subseteq \{1, 2, \ldots m\}$ to make an estimate of the gradient, at each step of the algorithm.

Also, an adaptive scheme on the batch size can be used in order to obtain a less noisy estimate of the gradient. For example, in [9] a method with a geometrically increasing batch is described. The discussion about noise reduction methods, would be incomplete without mentioning Stochastic Variance Reduced Gradient (SVRG) [11] and SAGA [12]. Both methods, for certain classes of cost functions, can achieve linear convergence. However, they have a bigger per iteration cost than SGD. Adam [13], a first order stochastic

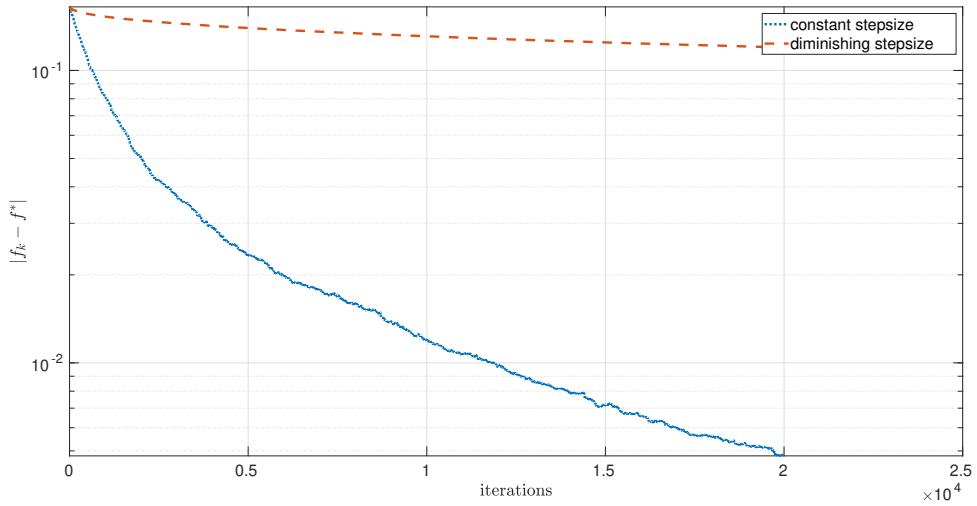


Figure 2.1: Stochastic Gradient descent for Least Squares problem $\frac{1}{2m}\sum_{i=1}^{m}(\mathbf{a}_i^T\mathbf{x}-b_i)^2$ for $m = 500$. The constant and variant stepsizes are $\alpha = \frac{B}{\rho}$, and $\alpha_k = \frac{B}{\rho\sqrt{k}}$. The parameters B, and ρ are set to $\|\mathbf{x}^*\|_2$, and $LB + \|\mathbf{A}^T\mathbf{b}\|_2$, respectively where L is $\lambda_{\max}(\mathbf{A}^T\mathbf{A})$.

algorithm is used extensively in neural network training process. For example, Keras [14], one of the most successful machine learning libraries these days, contains Adam (among other adaptive stochastic methods) in its set of optimizers.

In the sequel, we mention the second–order stochastic methods. Various schemes have been proposed in the literature, based on different choices for the second order information. We refer to Adagrad [15], an adaptive method that incorporates knowledge from previous iterations in order to tune in a more efficient manner the current step size. For a more detailed presentation of the SGD variants see [9], [16].

Finally, the role of momentum in SGD is explored in [17]. In [18], a method that exploits the Nesterov acceleration scheme is presented.

Chapter 3

Stochastic Canonical Polyadic Decomposition

Randomized techniques for solving problem (2.8) became very popular since they combine lower computational cost with sufficiently good accuracy. There are numerous works that employ different randomization techniques. Sub-sampling of the target tensor $\boldsymbol{\mathcal{X}}$ using regular sampling method has been introduced in [19]. Also, in [20], entries of the target tensor are sampled in a random manner and the respective blocks of the latent factors are updated at each iteration. In [21] and [22], a distributed framework is employed, where smaller replicas of the target tensor are independently factored. The resulting factors of each independent decomposition are effectively merged at the end to obtain the final latent factors.

In vanilla ALS, the MTTKRP is considered the "heavy burden" of the method. Consequently, a group of algorithms is developed towards the alleviation of the MTTKRP on the overall computational cost. A fiber sampling technique is used in [23] and [24]. With this approach, at each iteration the whole factor is updated, in contrast to [20]. This scheme facilitates the adaption of constraints on the latent factors.

3.1 Model Formulation - BrasCPD and AdaCPD

The BrasCPD algorithm of [24], uses a stochastic approach for the update of each factor by randomly sampling fibers from the tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{(\prod_{i=1}^{N} I_i)}$. More specifically, if the mode–i fibers are indexed by $\mathcal{F}^i \subset \{1, 2, \dots \prod_{m=1, m\neq i}^{N} I_m\}$, where the mode–$i$ fibers are the rows of $\mathbf{X}_{(i)}$, we obtain a smaller problem than (2.13). Namely, for each iteration k, we have

$$\min f_k^{(i)}(\mathbf{A}^{(i)}), \tag{3.1}$$

where

$$f_k^{(i)}(\mathbf{A}^{(i)}) = \|\mathbf{X}_{(i)}(\mathcal{F}_k^i, :) - \mathbf{K}_k^{(i)}(\mathcal{F}_k^i, :)\mathbf{A}^{(i)T}\|_F^2. \tag{3.2}$$

BrasCPD combines a fiber sampling technique, with ALS algorithm. In more detail, at each iteration k, a mode i is picked randomly. Then, $|\mathcal{F}^i|$ mode–i fibers are sampled. Finally, the update of factor $\mathbf{A}^{(i)}$ is performed via

$$\mathbf{A}_{k+1}^{(i)} = \mathbf{A}_k^{(i)} - \alpha_k \nabla f_k^{(i)}(\mathbf{A}^{(i)}), \quad i = 1, 2, \dots N, \tag{3.3}$$

where the matrix $\nabla f_k^{(i)}(\mathbf{A}^{(i)})$ is a stochastic approximation of the gradient

$$\nabla_{\mathbf{A}^{(i)}} f_k(\mathbf{A}^{(1)}, \mathbf{A}^{(2)} \dots \mathbf{A}^{(N)})$$

of the full problem (2.13) w.r.t factor $\mathbf{A}^{(i)}$[1]. The matrix $\nabla f_k^{(i)}(\mathbf{A}^{(i)})$ is computed via

$$\nabla f_k^{(i)}(\mathbf{A}^{(i)}) = \mathbf{A}_k^{(i)} \mathbf{K}_k^{(i)T}(\mathcal{F}_k^i, :)\mathbf{K}_k^{(i)}(\mathcal{F}_k^i, :) - \mathbf{X}_{(i)}^T(\mathcal{F}_k^i, :)\mathbf{K}_k^{(i)}(\mathcal{F}_k^i, :). \tag{3.4}$$

The other factors do not change during iteration k, that is, for $j \neq i$, $\mathbf{A}_{k+1}^{(j)} = \mathbf{A}_k^{(j)}$. Note that the computational cost of the MTTKRP drops to $\mathcal{O}(|\mathcal{F}_i|RI_i)$ flops. In the method described above, one stochastic gradient step is performed at each iteration, concerning a random set of fibers of the initial tensor $\boldsymbol{\mathcal{X}}$. The stepsize parameter sequence α_k follows the Robbins - Monro update rule [8]. The performance of the algorithm is mainly determined by the step-sizes α_k [9]. In BrasCPD, diminishing step-sizes are employed, namely, $\alpha_k = \frac{\alpha}{k^\beta}$, for appropriate values of parameters α and β. A method based on Adagrad [15], called AdaCPD, has also been proposed in [24]. The AdaCPD computes the necessary step-sizes using an accumulated-gradient mechanism with parameters $\eta > 0$, $\beta > 0$, and $\epsilon > 0$, namely

$$\alpha_k^{(i)} = \frac{\eta}{\left(\beta + \sum_{l=1}^k \nabla f_l^{(i)}(\mathbf{A})\right)^{1/2+\epsilon}}. \tag{3.5}$$

3.2 Accelerated Stochastic CPD - the algorithm

In this section, we describe the proposed method Accelerated Stochastic CPD (ASCPD). Following the sampling scheme introduced in [24], [23] we form the sketched problem w.r.t the factor $\mathbf{A}^{(i)}$

$$\min f_k^{(i)}(\mathbf{A}^{(i)}), \tag{3.6}$$

where $f_k^{(i)}(\mathbf{A}^{(i)}) = \|\mathbf{X}^{(i)}(\mathcal{F}_k^i, :) - \mathbf{K}_k^{(i)}(\mathcal{F}_k^i, :)\mathbf{A}^{(i)T}\|_F^2$. However, instead of a simple gradient step, we perform an iteration of Accelerated Gradient Descent or Nesterov Accelerated Gradient (NAG) [10, p. 91]. First at each iteration k, we compute the Hessian matrix

$$\mathbf{H}_k^{(i)} := \nabla^2 f_k^{(i)}(\mathbf{A}^{(i)}) = \mathbf{K}_k^{(i)T}(\mathcal{F}_k^i, :)\mathbf{K}_k^{(i)}(\mathcal{F}_k^i, :) \otimes \mathbf{I}.$$

In the sequel, we compute the quantities $L_k^{(i)}$ (smoothness parameter), and $\mu_k^{(i)}$ (strong-convexity parameter). We update the factor via

$$\mathbf{A}_{k+1}^{(i)} = \mathbf{Y}_k^{(i)} - \frac{1}{L_k^{(i)}} \nabla f_k^{(i)}(\mathbf{Y}_k^{(i)}), \tag{3.7}$$

followed by a momentum step (Nesterov momentum)

$$\mathbf{Y}_{k+1}^{(i)} = \mathbf{A}_{k+1}^{(i)} + \gamma_k(\mathbf{A}_{k+1}^{(i)} - \mathbf{A}_k^{(i)}). \tag{3.8}$$

[1]Note that, the gradient $\nabla f(\mathbf{A}^{(1)}, \mathbf{A}^{(2)} \dots \mathbf{A}^{(N)}) \in \mathbb{R}^{I_1 \times R} \times \mathbb{R}^{I_2 \times R} \times \cdots \times \mathbb{R}^{I_N \times R}$ of problem (2.8), is a block matrix, where block i is the gradient w.r.t the factor $\mathbf{A}^{(i)}$.

The quantity $\nabla f_k^{(i)}(\mathbf{Y}_k^{(i)})$ is the gradient calculated at $\mathbf{Y}_k{}^{(i)}$, namely

$$\nabla f_k^{(i)}(\mathbf{Y}_k^{(i)}) = \mathbf{Y}_k^{(i)}\mathbf{K}_k^{(i)T}(\mathcal{F}_k^i,:)\mathbf{K}_k^{(i)}(\mathcal{F}_k^i,:) - \mathbf{X}_{(i)}^T(\mathcal{F}_k^i,:)\mathbf{K}_k^{(i)}(\mathcal{F}_k^i,:). \tag{3.9}$$

We highlight that each factor $\mathbf{A}^{(i)}$ has its own sequence $\mathbf{Y}^{(i)}$. The parameter γ_k in the interpolation step (3.8), denotes the momentum parameter and is computed via $\gamma_k = \dfrac{1-\sqrt{\frac{\mu_k^{(i)}}{L_k^{(i)}}}}{1+\sqrt{\frac{\mu_k^{(i)}}{L_k^{(i)}}}}$. NAG algorithm for L-smooth, μ-strongly convex problems, achieves an accuracy of ϵ in

$$\mathcal{O}\left(\sqrt{\frac{L}{\mu}}\log\frac{1}{\epsilon}\right), \tag{3.10}$$

iterations. The performance of NAG algorithm, depends heavily on the condition number of the problem, namely the condition number of the matrix $\mathbf{H}_k^{(i)}$. Ill-conditioned Hessian matrices $\mathbf{H}_k^{(i)}$ may occur, when $|\mathcal{F}^i| < R$. To counter the ill–conditioned cases that may arise, we add a proximal term to the sketched problem (3.6). Thus, we obtain

$$\min_{\mathbf{A}^{(i)}} f_k^{(i)}(\mathbf{A}^{(i)}) + \frac{\lambda_k^{(i)}}{2}\|\mathbf{A}^{(i)} - \mathbf{A}_k^{(i)}\|_F^2, \tag{3.11}$$

where the proximal parameter $\lambda_k^{(i)}$ is tuned according to the problem. In more detail, the value of $\lambda_k^{(i)}$ increases, to attain small change on the current factor, between two consecutive iterations. In the presence of noise, we should opt for higher values of $\lambda_k^{(i)}$. On the contrary, in well-conditioned cases the proximal parameter $\lambda_k^{(i)}$ may set to 0 (or values < 1). In our implementation we use the $L_k^{(i)}$ parameter to tune $\lambda_k^{(i)}$, according to the rule

$$\lambda_k^{(i)} = \begin{cases} \mu_k^{(i)}, & \text{if } \frac{L_k^{(i)}}{\mu_k^{(i)}} < \mathcal{C}, \\ \frac{L_k^{(i)}}{\mathcal{C}}, & \text{otherwise.} \end{cases} \tag{3.12}$$

where $\mathcal{C}$ is a constant with values $\mathcal{C} = 10, 10^2, 10^3$. The modified gradient for problem (3.11) is

$$\begin{aligned}\nabla f_k^{(i)}(\mathbf{Y}_k^{(i)}) = \mathbf{Y}_k^{(i)}&\left(\mathbf{K}_k^{(i)T}(\mathcal{F}_k^i,:)\mathbf{K}_k^{(i)}(\mathcal{F}_k^i,:) + \lambda_k^{(i)}\mathbf{I}\right) \\ &- \left(\mathbf{X}_{(i)}^T(\mathcal{F}_k^i,:)\mathbf{K}_k^{(i)}(\mathcal{F}_k^i,:) + \lambda_k^{(i)}\mathbf{A}_k^{(i)}\right).\end{aligned} \tag{3.13}$$

Also, note that we set $\tilde{L}_k^{(i)} = L_k^{(i)} + \lambda_k^{(i)}$, $\tilde{\mu}_k^{(i)} = \mu_k^{(i)} + \lambda_k^{(i)}$ and we use $\frac{1}{\tilde{L}_k^{(i)}}$ to update the factor $\mathbf{A}_{k+1}^{(i)}$ in (3.7). The ASCPD algorithm follows in Algorithm 1.

The quantities $L_k^{(i)}, \mu_k^{(i)}$ are computed through an eigen/singular value decomposition, with computational cost $\mathcal{O}(R^3)$. Thus, the complexity scales cubicly in R. However, in this framework, the value of R is usually small. In cases of large values of R, more efficient methods for computing eigenvalues can be used to obtain $L_k^{(i)}$ and $\mu_k^{(i)}$.

Algorithm 1: ASCPD

Result: $\{\mathbf{A}^{(i)}\}_{i=1}^{N}$

Input: tensor $\boldsymbol{\mathcal{X}}$, $\mathbf{A}_0^{(i)} = \mathbf{Y}_0^{(i)}$, $i = 1, \ldots, N$, blocksizes $B^i = |\mathcal{F}^i|$, $i = 1, \ldots, N$.
$k = 0$;
while *terminating condition is not satisfied* **do**
  Uniformly sample i from $\{1, 2 \ldots N\}$;
  Uniformly sample B^i mode-i fibers;
  Compute stochastic gradient using (3.13);
  Compute $L_k^{(i)}$, $\mu_k^{(i)}$, and $\lambda_k^{(i)}$;
  Compute $\mathbf{A}_{k+1}^{(i)}$ using (3.7) ;
  Compute $\mathbf{Y}_{k+1}^{(i)}$ using (3.8) ;
  $k = k + 1$;
end

LS problem with constraints

We consider the case where various constraints are imposed on the latent factors. We solve a constrained version of the sketched problem (3.6). The efficient solution of such problems is possible, since there are optimal algorithms for different constraints. Proximal Gradient Descent (PGD) [25] is the mainstream algorithm for constrained optimization. Specifically, via PGD we can solve problems of the form

$$\min_{\mathbf{A}^{(i)} \in \mathbb{A}^i} f_k^{(i)}(\mathbf{A}^{(i)}) + \frac{\lambda_k^{(i)}}{2} \|\mathbf{A} - \mathbf{A}_k^{(i)}\|_F^2. \tag{3.14}$$

Note that $\mathbb{A}^i$ denotes the constraint set for each factor $\mathbf{A}^{(i)}$. Similarly to the proposed algorithm, we do not solve the problem (3.14), but we perform one step of PGD. We define $h_i(\cdot)$ as the indicator function of the set $\mathbb{A}^i$. In general, the update step of PGD is an optimization problem itself

$$\mathbf{A}_{k+1}^{(i)} = \underset{\mathbf{A}^{(i)}}{\operatorname{argmin}}\ h_i\left(\mathbf{A}^{(i)}\right) + \|\mathbf{A}^{(i)} - (\mathbf{A}_k^{(i)} - \alpha_k \nabla f_k^{(i)}(\mathbf{A}^{(i)}))\|_F^2. \tag{3.15}$$

Problem (3.15) is called proximal operator of $h_i(\cdot)$, namely

$$\mathbf{A}_{k+1}^{(i)} = \operatorname{prox}_{h_i}\left(\mathbf{A}_k^{(i)} - \alpha_k \nabla f_k^{(i)}(\mathbf{A}^{(i)})\right). \tag{3.16}$$

For example, in the case where we enforce nonnegative constraints on factors $\{\mathbf{A}^{(n)}\}_{n=1}^{N}$, the proximal operator $h_i(\cdot)$ is simply the $\max(0, \cdot)$ operator. In the ASCPD algorithm, the PGD step becomes

$$\mathbf{A}_{k+1}^{(i)} = \operatorname{prox}_{h_i}\left(\mathbf{Y}_k^{(i)} - \frac{1}{\tilde{L}_k^{(i)}} \nabla f_k^{(i)}(\mathbf{Y}_k^{(i)})\right). \tag{3.17}$$

	Ranks			
	R=20	***R=50***	***R=100***	***R=200***
NALS	6.5497e-06	1.0109e-04	2.8427e-04	4.3066e-04
BrasCPD $\alpha = 0.1$	1.5683e-06	1.9975e-05	5.6623e-05	1.4222e-04
BrasCPD optimal	2.6926e-16	9.9118e-12	2.5192e-06	7.6814e-04
AdaCPD	2.4895e-14	7.4076e-06	0.0037	0.0027
ASCPD $\mathcal{C} = 10^2$	7.3521e-16	1.0083e-15	1.2200e-15	6.7577e-11
ASCPD $\mathcal{C} = 10^3$	7.4460e-16	1.5204e-15	2.3261e-15	1.3805e-04

Table 3.1: Results for the noiseless case, for different ranks R

3.3 Numerical Experiments

In this section, we present the results of some numerical experiments for the proposed ASCPD algorithm versus its stochastic counterparts. We explore the performance of the algorithm through noiseless and very noisy settings, and present the results for different values of blocksize $|\mathcal{F}^i|$ and regularization parameter $\lambda^{(i)}$. Also, we illustrate the performance against deterministic algorithms that solve the same problem. The relative tensor reconstruction error $m_k = \frac{\|\mathcal{X}-\widehat{\mathcal{X}}\|_F}{\|\mathcal{X}\|_F}$, where $\widehat{\boldsymbol{\mathcal{X}}}$ is the estimated tensor, is used in the majority of the following experiments to evaluate the performance. Note that here k denotes a *full*-iteration, the number of iterations required to traverse the whole tensor. The results for every experiment, are extracted after 10 Monte-Carlo trials. The following experiments are implemented in MATLAB.

Noiseless Case

For the noiseless case, the performance of the algorithm is pretty impressive. However, it should be made clear that the noiseless sketched problem (3.6) is ease, due to the possible repetition of the data. For the matter of completeness, we illustrate the performance of the algorithm for the noiseless case.

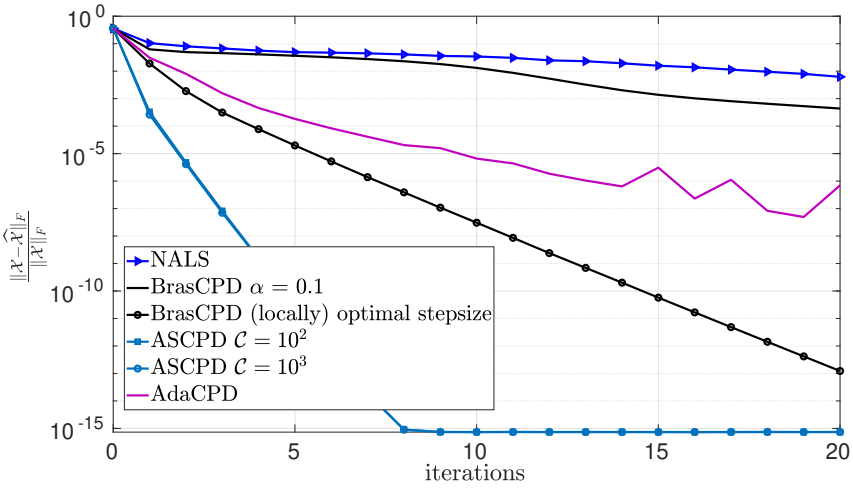
In Figures 3.1, 3.2, we illustrate the performance of the ASCPD algorithm versus the BrasCPD, AdaCPD and NALS algorithms [26] on a medium sized tensor. We set $I_1 = I_2 = I_3 = 300$, and create a tensor $\boldsymbol{\mathcal{X}}^o \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, whose latent factors are drawn from an i.i.d. uniform distribution in $[0, 1]$. We set the blocksize $|\mathcal{F}^i| = 100$ for all stochastic algorithms. BrasCPD parameters α, β are set to 0.1 and 10^{-6}, respectively. In AdaCPD, we choose $\eta = 1$ and $\beta = 10^{-6}$. We test various values for $\mathcal{C}$, namely $\mathcal{C} = 10^2, 10^3$ for the ASCPD algorithm.

Noisy Case

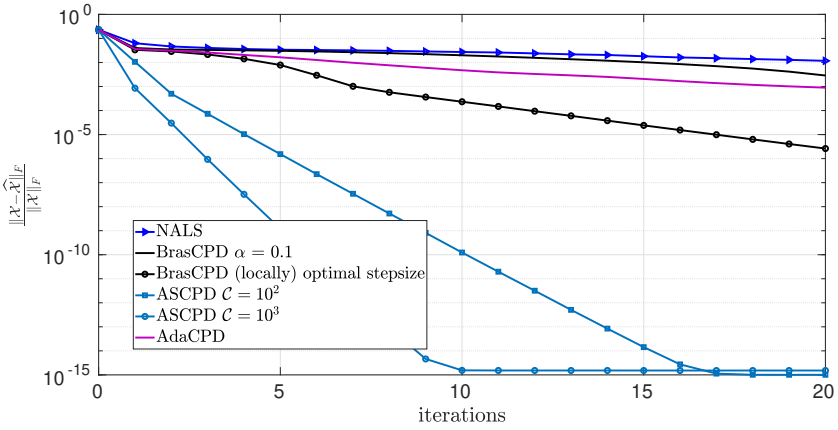
We generate a 3–rd order tensor $\boldsymbol{\mathcal{X}}^o \in \mathbb{R}_+^{I_1 \times I_2 \times I_3}$, via uniformly drawn latent factors. Throughout this section, we consider the noisy case, namely

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}}^o + \sigma_\epsilon \boldsymbol{\mathcal{E}},$$

where $\boldsymbol{\mathcal{E}} \sim \mathcal{N}(0, \sigma_\epsilon^2)$ denotes the additive noise. In our experiments, we consider different values of SNR, which is defined as



(a) $R = 20$



(b) $R = 50$

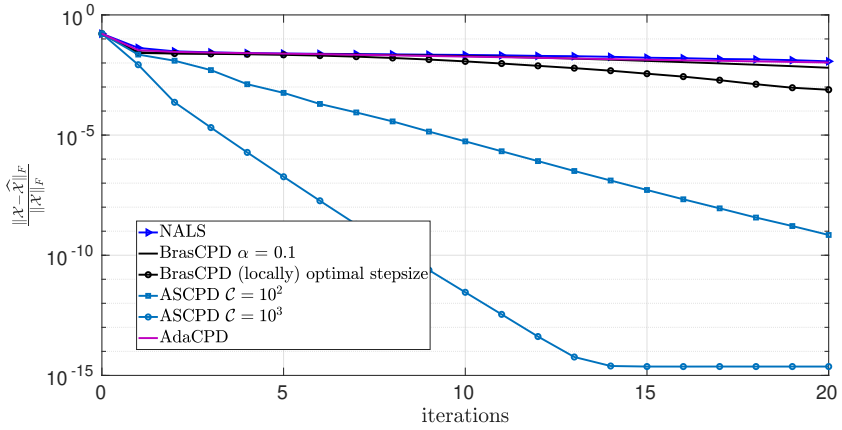
Figure 3.1: Noiseless case for tensor with $I_1 = I_2 = I_3 = 300$, and $R = 20, 50$. Also $\{\mathbb{A}^i\}_{i=1}^3 = \mathbb{R}_+$.

$$\text{SNR} = \frac{\|\boldsymbol{\mathcal{X}}^o\|_F^2}{\sigma_\epsilon^2 \|\boldsymbol{\mathcal{E}}\|_F^2}.$$

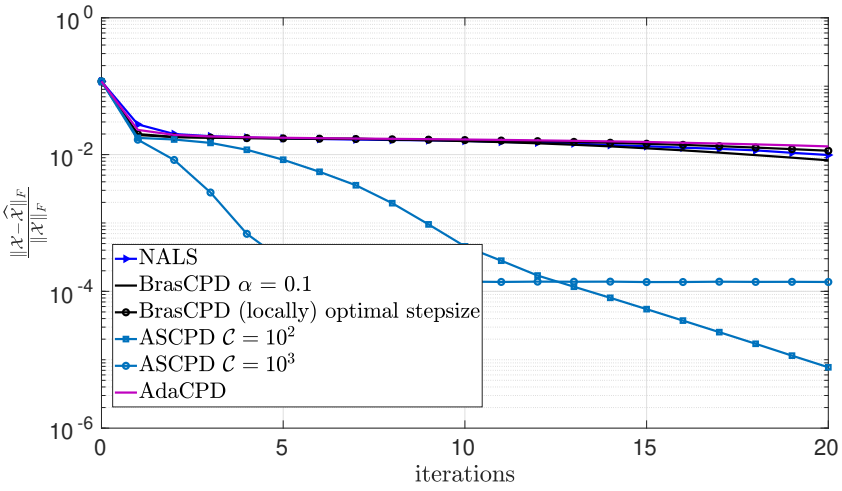
We evaluate the standard deviation for the different values of SNR via the aforementioned formula. First, we set $I_1 = I_2 = I_3 = 100$, $|\mathcal{F}^i| = 500$, $R = 50$ and we test the performance of the ASCPD algorithm versus the NALS, BrasCPD and AdaCPD algorithms for SNR 6.98dB and 20dB. For BrasCPD we set $\alpha = 0.1$ and for AdaCPD we have $\eta = 1$. For the parameter $\mathcal{C}$, we set $\mathcal{C} = 10^2,\ 10^3$. In Figure 3.3, we illustrate the results. We observed that in very noisy cases, the proximal parameter should be set to $\lambda_k^{(i)} = \frac{L_k^{(i)}}{\overline{\mathcal{C}}}$, where $\overline{\mathcal{C}} < \mathcal{C}$.

In Figure 3.4, we illustrate the results for $I_1 = I_2 = I_3 = 200$, $|\mathcal{F}^i| = 500$, $R = 100$ for SNR 10dB and 30dB. In Figure 3.5, we change the blocksize to $|\mathcal{F}^i| = 100$. In Figure 3.6, we illustrate the results for $I_1 = I_2 = I_3 = 200$, $|\mathcal{F}^i| = 500$, $R = 30$ for SNR 10dB and 30dB. In Figure 3.7, we set $|\mathcal{F}^i| = 100$ and we keep the rest of the parameters the same.

We observe that



(a) $R = 100$



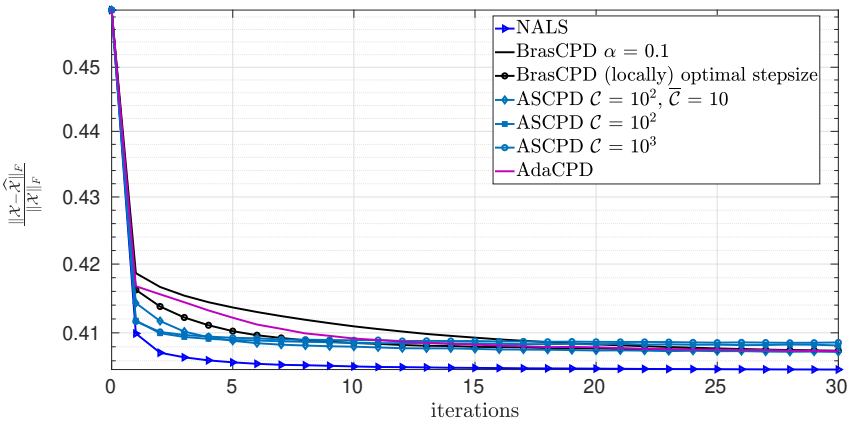
(b) $R = 200$

Figure 3.2: Noiseless case for tensor with $I_1 = I_2 = I_3 = 300$, and $R = 100, 200$. Also $\{\mathbb{A}^i\}_{i=1}^3 = \mathbb{R}_+$.

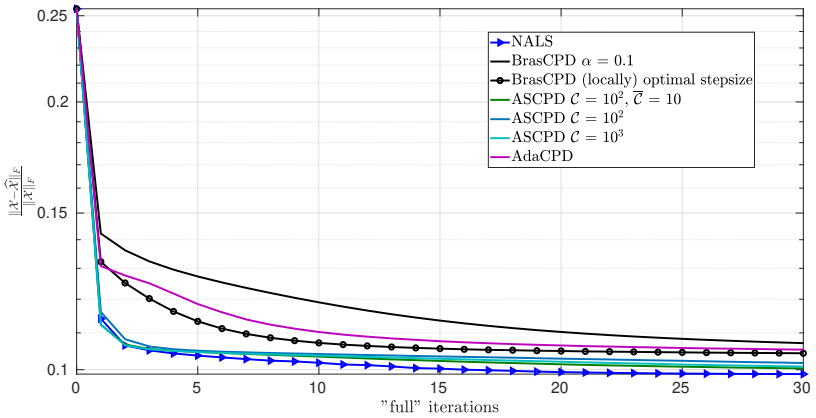
1. in the low SNR cases, the deterministic algorithm outperforms all stochastic approaches. The relative performance of AdaCPD and ASCPD depends on the block size; for "small" block sizes, AdaCPD outperforms the ASCPD, while, for "large" block sizes, the opposite happens.
2. in the high SNR cases, the ASCPD outperforms all other methods; we note that the BrasCPD with locally optimal step-size in some cases outperforms AdaCPD.

Real Data

We test the ASCPD algorithm on real datasets to evaluate its performance on real applications. Similarly to [24], we use the Indian Pine and PaviaU datasets which in fact are Hyperspectral Images (HSIs). HSI sensors collect data in a group of images, on different wavelength ranges. The resulting data-cube is a third order tensor. The Indian Pine dataset is of size $145 \times 145 \times 220$ and consists of data acquired with the AVIRIS sensor, on Indian Pines site in Indiana (USA). The PaviaU dataset has size $610 \times 340 \times 103$ and



(a) SNR = 6.98dB



(b) SNR = 20dB

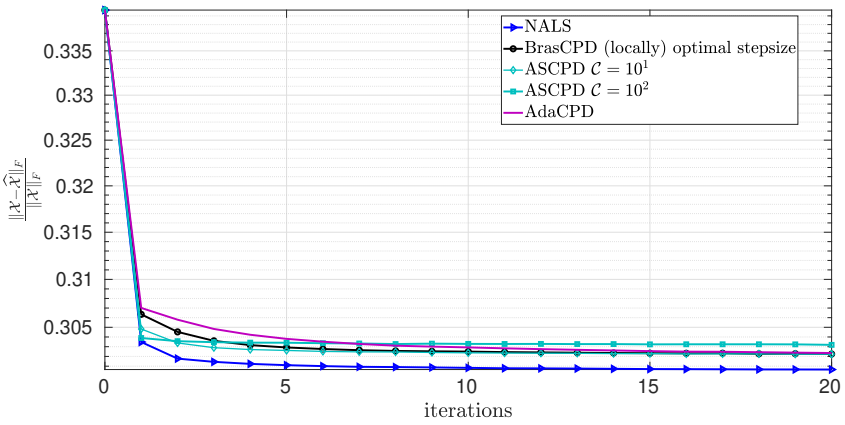
Figure 3.3: Noisy case for tensor with $I_1 = I_2 = I_3 = 100$, $R = 50$. Also $\left\{\mathbb{A}^i\right\}_{i=1}^3 = \mathbb{R}_+$.

consists of a scene of Pavia University in Italy[2]. We test the performance of BrasCPD, AdaCPD and ASCPD on both datasets. For all algorithms we use $|\mathcal{F}^i| = 500$. As a performance measure we use the metric m_k, which is evaluated at each iteration. We illustrate the results in Figures 3.8, and 3.9.

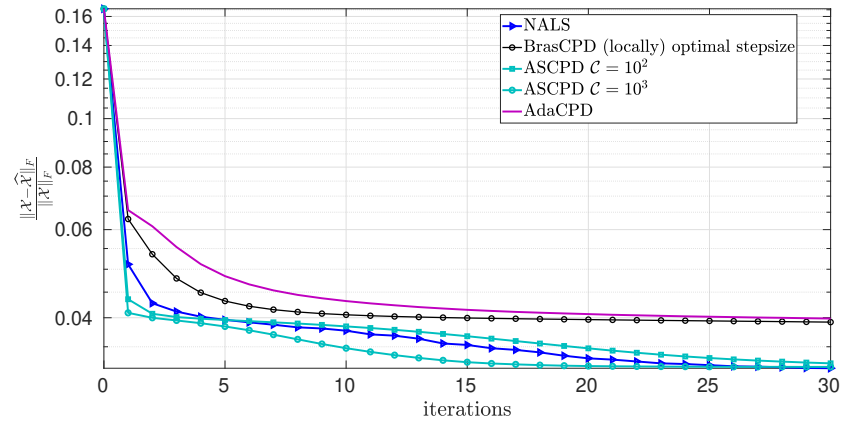
AdaCPD parameter η

We tested the performance of AdaCPD for various values of η. AdaCPD does not need manual (and sometimes exhausting) tuning of the stepsize, however, it is not immune to bad values of η. We have noticed that the parameter η ,that gives the best performance on the current setting, is higher. In Figure 3.10, we illustrate the results for different values of the parameter η for Indian Pines and PaviaU datasets. The value of β seems to not affect the results.

[2]Both datasets are available in `http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes`.



(a) SNR = 10dB

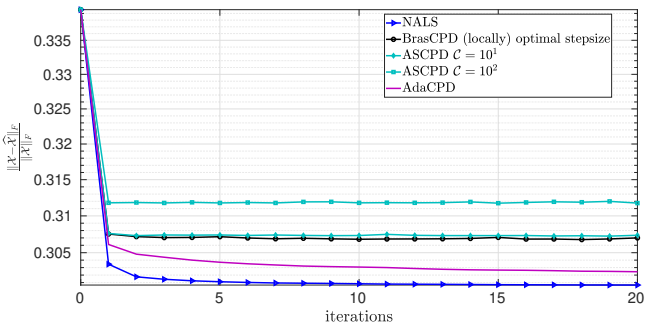


(b) SNR = 30dB

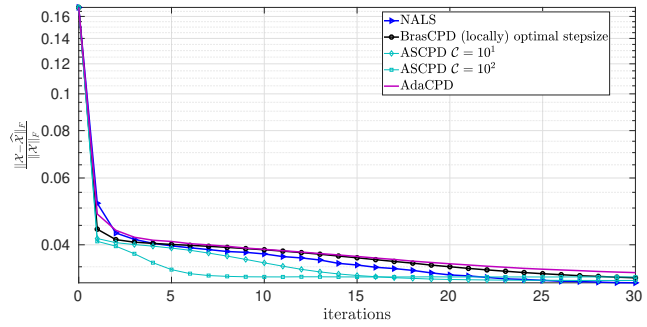
Figure 3.4: Noisy case for tensor with $I_1 = I_2 = I_3 = 200$, $R = 100$, $|\mathcal{F}^i| = 500$. Also $\{\mathbb{A}^i\}_{i=1}^3 = \mathbb{R}_+$.

3.4 Conclusions

In both noiseless and noisy cases, the tuning of the problem parameters is crucial. First, the regularization parameter $\lambda_k^{(i)}$ should admit a different computation (different value of $\mathcal{C}$) depending on the noisy level of the problem. When noise is present, in cases with very low SNR (SNR $<$ 10dB), higher values of $|\mathcal{F}^i|$ are favorable. In higher SNR cases, the parameter tuning is more flexible. We also implemented the ASCPD algorithm in C++ using the Eigen Library for fast computations [27]. Through experiments, we observed that the main bottleneck of ASCPD is no longer the MTTKRP, but the sampling of fibers from the tensor $\boldsymbol{\mathcal{X}}$. In the next Chapter we present a framework that may overcome this overhead.

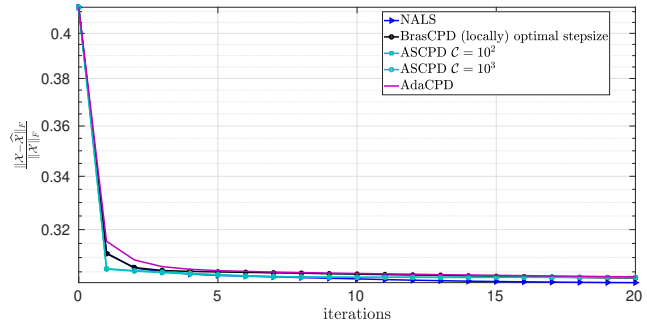


(a) SNR = 10dB

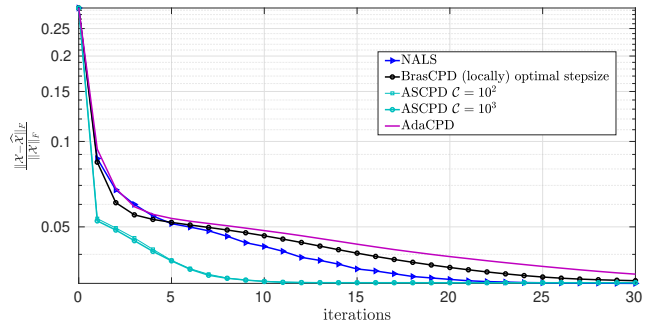


(b) SNR = 30dB

Figure 3.5: Noisy case for tensor with $I_1 = I_2 = I_3 = 200$, $R = 100$, $|\mathcal{F}^i| = 100$. Also $\{\mathbb{A}^i\}_{i=1}^3 = \mathbb{R}_+$.

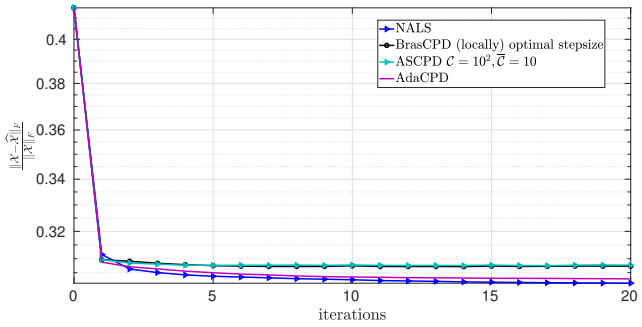


(a) SNR = 10dB

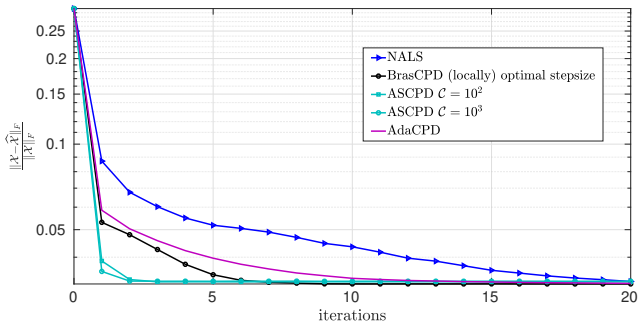


(b) SNR = 30dB

Figure 3.6: Noisy case for tensor with $I_1 = I_2 = I_3 = 200$, $R = 30$, $|\mathcal{F}^i| = 500$. Also $\{\mathbb{A}^i\}_{i=1}^3 = \mathbb{R}_+$.

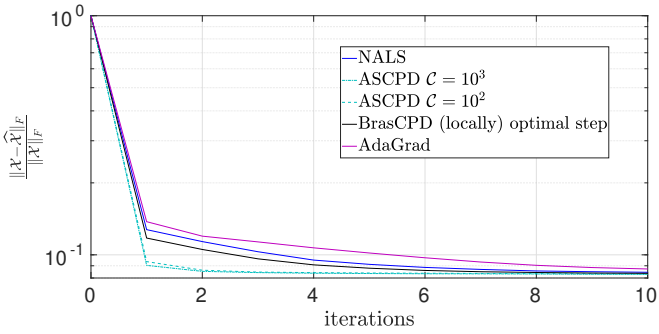


(a) SNR = 10dB

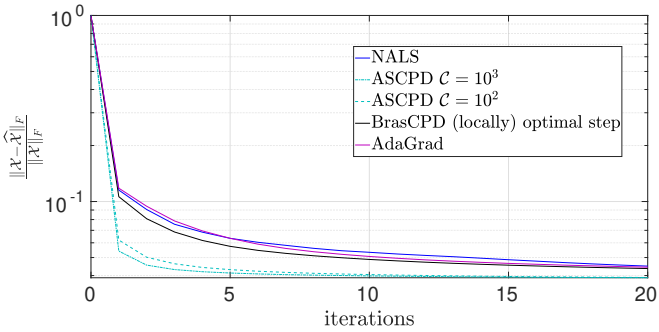


(b) SNR = 30dB

Figure 3.7: Noisy case for tensor with $I_1 = I_2 = I_3 = 200$, $R = 30$, $|\mathcal{F}^i| = 100$. Also $\{\mathbb{A}^i\}_{i=1}^3 = \mathbb{R}_+$.

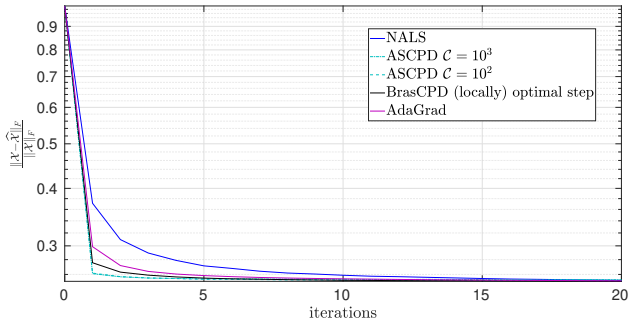


(a)$R = 10$

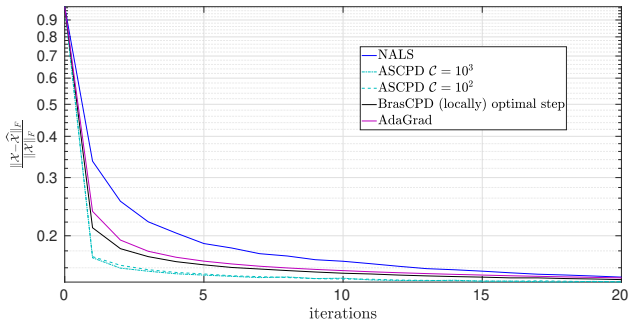


(b)$R = 100$

Figure 3.8: Indian Pines dataset for $R = 10$ (top), $R = 100$ (bottom).

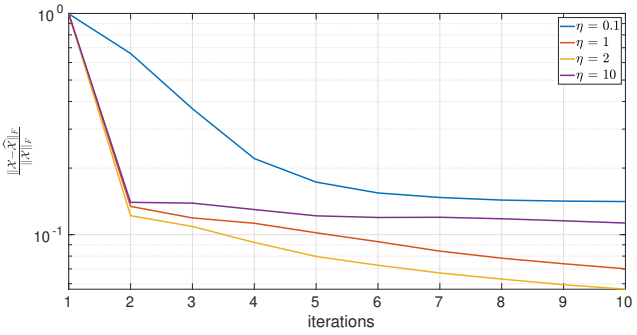


(a)$R = 50$

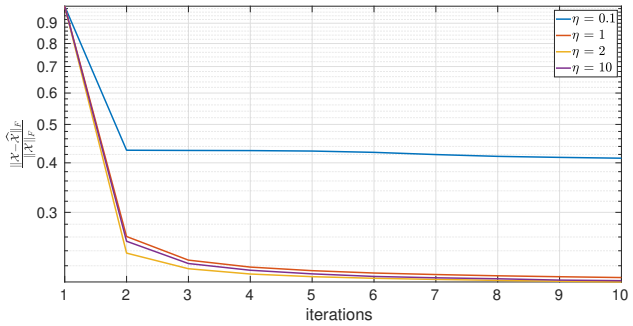


(b)$R = 200$

Figure 3.9: PaviaU dataset for $R = 50$ (top), $R = 200$ (bottom).



(a) Indian Pines dataset



(b) PaviaU dataset

Figure 3.10: AdaCPD algorithm for the real data datasets for different values of η, $\beta = 10^{-6}, R = 100, |\mathcal{F}^i| = 500$.

Chapter 4

Parallel Implementation

In this chapter, we examine a possible parallel implementation of ASCPD. We explain the reasons that may render stochastic methods not suitable for parallelization and we describe in detail an alternative method that may overcome these limitations. We highlight that we refer to thread–level parallelism using the shared memory API openMP and the Eigen Library.

4.1 Naive approach

At first glance, similarly to ALS CPD approach, we try to compute in parallel the MTTKRP. However, the "reduced" MTTKRP in ASCPD method, demands a much smaller number of arithmetic operations at each iteration. Through numerical experiments, we did not observe significant speedup using different number of threads. We conclude that, for medium scale problems the serial version of ASCPD is pretty competitive and the computation of MTTKRP in parallel may show an improvement in very large problems, where the blocksize B^i needs to be significantly high.

4.2 A multi-threaded approach

The main bottleneck of ASCPD is the sampling of the tensor $\boldsymbol{\mathcal{X}}$ as well as the computation of the cost function. Since we only obtain the tensor $\boldsymbol{\mathcal{X}}$ in the main memory, we need a more efficient way to sample the set of mode–i fibers at each iteration. Inspired by [28], we fork t threads via the OpenMP directive `#pragma omp parallel`, where each one is responsible for the sampling of $\frac{B^i}{t}$ mode–i fibers. At each iteration k, each thread computes its local Hessian matrix, $L_k^{(i)}$, $\mu_k^{(i)}$ and the momentum parameters, and then, updates its local copy of the factor $\mathbf{A}_k^{(i)}$ and the interpolation sequence $\mathbf{Y}_k^{(i)}$. The local factors of the t threads in the team, are reduced and scaled to the global variables $\left\{\mathbf{A}_p^{(i)}\right\}_{i=1}^{N}$, every p iterations. The algorithm follows in Algorithm 2.

The parameter p regulates the frequency of the reduction of the local variables. A straight-forward way to tune parameter p is to decrease its value with the increase of the working threads. Since the local B^i is smaller when the number of the available threads increases, the global variables need more frequent update.

The presented method is memory bounded though. We only have the tensor available in the main memory (the mode–1 matricization $\mathbf{X}_{(1)}$), which is stored in a column-major structure. Thus, the way we access the tensor to acquire the randomly selected fibers is not optimal for all modes. To address the possible stall of the CPU, we sort the indices that

Algorithm 2: Parallel ASCPD

Input: $\boldsymbol{\mathcal{X}}, \left\{\mathbf{A}_0^{(i)}\right\}_{i=1}^{N}, B^i, \text{with } i = 1, \dots N, R, p$

$\left\{\mathbf{Y}_0^{(i)}\right\}_{i=1}^{N} = \left\{\mathbf{A}_0^{(i)}\right\}_{i=1}^{N}$
$k = 0$
while (1) **do**
  if (term_cond is TRUE) **then**
    break
  else
    #pragma omp parallel
    Select mode i (same for every thread)
    Sample B^i/t mode–i fibers
    Compute $L_k^{(i)}, \mu_k^{(i)}$
    Update local $\mathbf{A}^{(i)}$ and local $\mathbf{Y}^{(i)}$
    if $k\%p$ **then**
      for $i = 1, \dots, N$ **do**
        global $\mathbf{A}^{(i)}$ += $\frac{1}{num\_threads} \left\{\mathbf{A}_{t,k}^{(i)}\right\}_{t=1}^{num\_threads}$
        global $\mathbf{Y}^{(i)}$ += $\frac{1}{num\_threads} \left\{\mathbf{A}_{t,k}^{(i)}\right\}_{t=1}^{num\_threads}$
  $k = k + 1$
return $\left\{\mathbf{A}_k^{(i)}\right\}_{i=1}^{N}$.

correspond to the fibers selected by each thread. In this way, the fibers to be extracted might be closer and we can exploit the system cache in a more efficient way by enforcing spatial locality.

To reduce the overhead induced by the cost function computations, we also compute it in parallel, using the memory friendly method of partial MTTKRP by Giannis Papagiannakos.

4.3 Numerical Experiments

We test our parallel approach by creating a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}_+^{800\times800\times800}$, with $R = 15, 50$. We measure the total execution time of the parallel ASCPD algorithm for different number of threads, namely $t = 1, 2, 4, 6$. We use the speedup attained by the different values of t as the metric to evaluate the performance of the parallel implementation. For the serial version, we set $B^i = 2000$. For the synchronization parameter p, we examined various values between $100 - 1000$. Through numerical experiments, we concluded that for this experiment, $p = 1000$ is a good choice. In cases where more working threads are available, the value of p should be inversely proportional to the number of threads. The results are extracted after 5 Monte-Carlo experiments. In Figures 4.1 and 4.2, we illustrate the attained speedup for $R = 15, 50$ and the total execution time of ASCPD (while loop). In Figure 4.3, we illustrate the mean execution time of the fiber–sampling step, which is the main bottleneck. In Figure 4.4, we present the results for a 4–th order tensor with

dimensions $I_1 = I_2 = I_3 = I_4 = 100$ and $R = 50$. In both cases, we observed that the attained speedup for $t > 4$ is not significant at all, in some cases the total execution time may be higher for $t = 6$. This is probably a result of the "cache thrashing". However, due to the nature of the algorithm, a possible solution is not straight-forward.

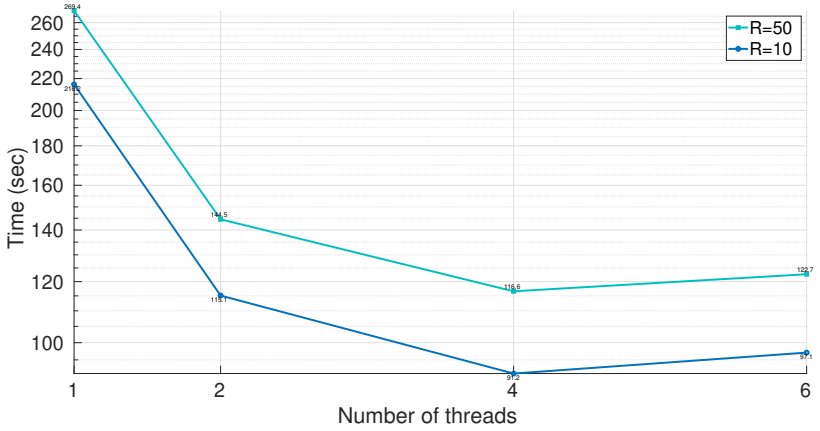


Figure 4.1: Execution time of the while loop for a tensor of dimensions $800 \times 800 \times 800$, with $R = 15, 50$, via parallel ASCPD.

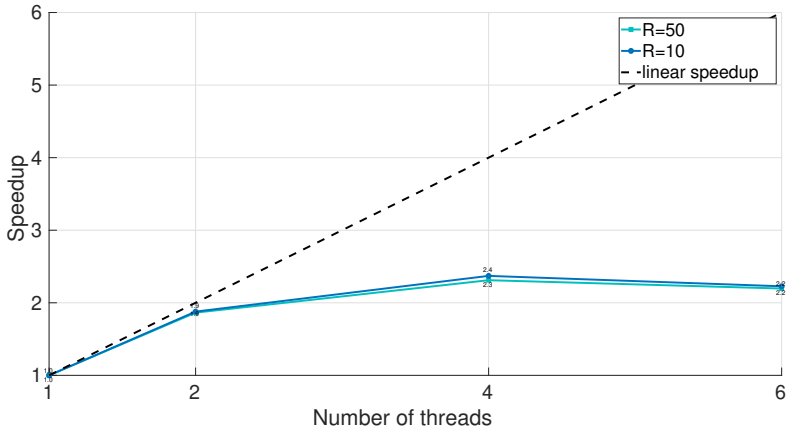


Figure 4.2: Speedup attained for a tensor of dimensions $800 \times 800 \times 800$ with $R = 15, 50$, via parallel ASCPD.

4.4 Conclusions

In this Chapter, we examined a possible parallelization of ASCPD. We mentioned a naive approach where MTTKRP can be computed in parallel as well as a multi–threaded approach where each thread contributes to the update of the factor. Through numerical experiments, we conclude that due to the nature of stochastic algorithms, parallel implementations are not as straight-forward and efficient (in terms of speedup) as their deterministic counterparts.

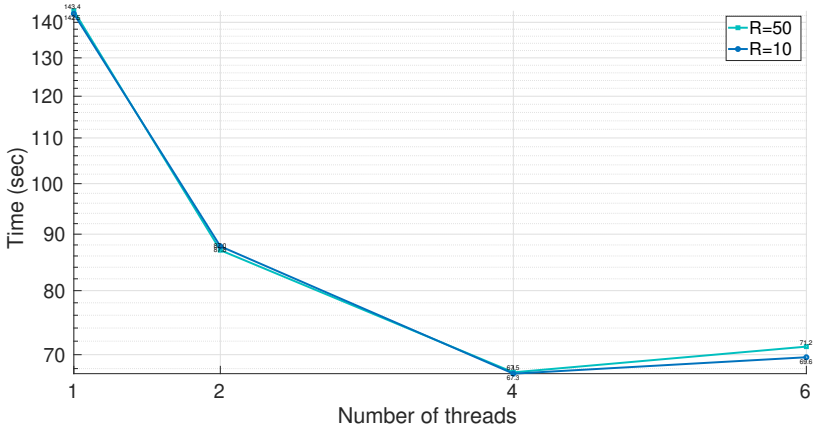


Figure 4.3: Mean execution time of the fiber–sampling step for a tensor of dimensions $800 \times 800 \times 800$, with $R = 15, 50$.

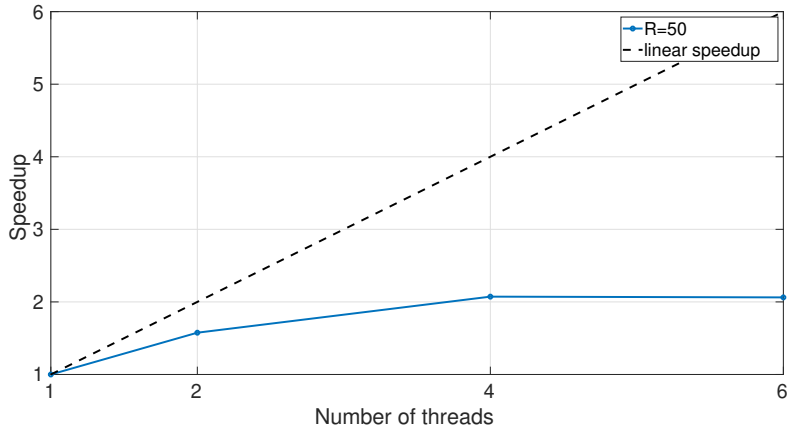


Figure 4.4: Speedup attained for a tensor of dimensions $100 \times 100 \times 100 \times 100$ with $R = 50$, via parallel ASCPD.

Chapter 5

Stochastic Tensor Completion

5.1 Tensor factorization with missing elements

Let $\boldsymbol{\mathcal{X}}^o \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ be a N-th order tensor, which admits the CP decomposition (2.7). Namely $\boldsymbol{\mathcal{X}}^o = [\![\mathbf{A}^{o(1)}, \ldots, \mathbf{A}^{o(N)}]\!]$ where $\mathbf{A}^{o(i)} = [\mathbf{a}_1^{o(i)} \ \cdots \ \mathbf{a}_R^{o(i)}] \in \mathbb{R}^{I_i \times R}$, with $i \in \mathbb{N}_N$. Also, assume a noisy tensor $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}}^o + \boldsymbol{\mathcal{E}}$, where $\boldsymbol{\mathcal{E}}$ is the additive noise. We define $\Omega \subseteq \{1 \ldots I_1\} \times \{1 \ldots I_2\} \times \cdots \times \{1 \ldots I_N\}$ as the set of indices of the observed entries of $\boldsymbol{\mathcal{X}}$. Let $\boldsymbol{\mathcal{M}}$ be a tensor of the same size as $\boldsymbol{\mathcal{X}}$, with entries $\boldsymbol{\mathcal{M}}(i_1, i_2, \ldots, i_N)$ equal to one or zero based on the availability of the corresponding element of $\boldsymbol{\mathcal{X}}$. That is

$$\boldsymbol{\mathcal{M}}(i_1, i_2, \ldots, i_N) = \begin{cases} 1, & \text{if } (i_1, i_2, \ldots, i_N) \in \Omega, \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

We consider the tensor completion problem

$$\min f_\Omega \left(\mathbf{A}^{(1)}, \mathbf{A}^{(2)} \ldots, \mathbf{A}^{(N)}\right) + \tfrac{\lambda}{2} \textstyle\sum_{i=1}^{N} \left\|\mathbf{A}^{(i)}\right\|_F^2, \tag{5.2}$$

where

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

If $\widehat{\boldsymbol{\mathcal{X}}} = [\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}]\!]$, then, for an arbitrary mode i, the corresponding matrix unfolding is given by

$$\widehat{\mathbf{X}}_{(i)} = \mathbf{A}^{(i)} \left(\mathbf{A}^{(N)} \odot \mathbf{A}^{(N-1)} \odot \cdots \odot \mathbf{A}^{(i+1)} \odot \mathbf{A}^{(i-1)} \odot \cdots \odot \mathbf{A}^{(1)}\right)^T. \tag{5.4}$$

Thus, f_Ω can be expressed as

$$f_\Omega(\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}) = \frac{1}{2} \left\| \mathbf{M}_{(i)} \circledast \left(\mathbf{X}_{(i)} - \widehat{\mathbf{X}}_{(i)}\right) \right\|_F^2, \quad i \in \mathbb{N}_N, \tag{5.5}$$

where $\mathbf{M}_{(i)}$, and $\mathbf{X}_{(i)}$ are the matrix unfoldings of $\boldsymbol{\mathcal{M}}$ and $\boldsymbol{\mathcal{X}}$, with respect to the i-th mode, correspondingly.

5.1.1 Nonnegative Matrix Completion

Similarly to AO NTF, we consider the Nonnegative Matrix Completion (NMC) problem, the building block of AO Nonnegative Tensor Completion (AO NTC). Assume matrices $\mathbf{X} \in \mathbb{R}^{P \times Q}$, $\mathbf{A} \in \mathbb{R}^{P \times R}$, $\mathbf{B} \in \mathbb{R}^{Q \times R}$. Also, let $\Omega \subseteq \{1 \ldots P\} \times \{1 \ldots Q\}$ be the set of indices of the known entries of $\mathbf{X}$, and $\mathbf{M}$ be a matrix with the same size as $\mathbf{X}$, with elements

$\mathbf{M}(i,j)$ equal to one or zero based on the availability of the corresponding element of $\mathbf{X}$. We consider the problem

$$\min_{\mathbf{A}\geq\mathbf{0}} f_\Omega(\mathbf{A}) := \frac{1}{2}\left\|\mathbf{M}\circledast\left(\mathbf{X}-\mathbf{A}\mathbf{B}^T\right)\right\|_F^2+\frac{\lambda}{2}\left\|\mathbf{A}\right\|_F^2, \tag{5.6}$$

which is of the form of (5.5) plus a regularization term. The gradient and the Hessian of f_Ω, at point $\mathbf{A}$, are given by

$$\nabla f_\Omega(\mathbf{A}) = -\left(\mathbf{M}\circledast\mathbf{X}-\mathbf{M}\circledast\left(\mathbf{A}\mathbf{B}^T\right)\right)\mathbf{B}+\lambda\mathbf{A}, \tag{5.7}$$

and

$$\nabla^2 f_\Omega(\mathbf{A}) = \left(\mathbf{B}^T\otimes\mathbf{I}_P\right)\mathtt{diag}\left(\mathtt{vec}\left(\mathbf{M}\right)\right)\left(\mathbf{B}\otimes\mathbf{I}_P\right)+\lambda\mathbf{I}_{PR}. \tag{5.8}$$

We solve problem (5.6) with a stochastic variant of the first-order optimal (Nesterov-type) algorithm. In more detail, at each iteration l of the stochastic algorithm, we define a new set of indices $\widehat{\Omega}\subset\Omega$, and a matrix $\widehat{\mathbf{M}}^l$, of the same size as $\mathbf{M}$, as

$$\widehat{\mathbf{M}}^l(i,j) = \begin{cases} 1, & \text{if } (i,j)\in\widehat{\Omega},\\ 0, & \text{otherwise.}\end{cases} \tag{5.9}$$

The acquisition of $\widehat{\Omega}$, at each iteration l, can be done in various ways. In our implementation, the subset $\widehat{\Omega}$ is created in a random manner. More specifically, at each iteration l, and for each row p, we sample from $\mathbf{M}(p,:)$ a fixed size of nonzero elements B_p^l *(blocksize per row)*. We note that $B^l=|\widehat{\Omega}|$ and that $B_p^l=\frac{B^l}{P}$.

At each iteration l of Stochastic NMC, the gradient and the Hessian of $f_{\widehat{\Omega}}$, at point $\mathbf{A}$, are given by

$$\nabla f_{\widehat{\Omega}}(\mathbf{A}) = -\left(\widehat{\mathbf{M}}^l\circledast\mathbf{X}-\widehat{\mathbf{M}}^l\circledast\left(\mathbf{A}\mathbf{B}^T\right)\right)\mathbf{B}+\lambda\mathbf{A}, \tag{5.10}$$

and

$$\nabla^2 f_{\widehat{\Omega}}(\mathbf{A}) = \left(\mathbf{B}^T\otimes\mathbf{I}_P\right)\mathtt{diag}\left(\mathtt{vec}\left(\widehat{\mathbf{M}}^l\right)\right)\left(\mathbf{B}\otimes\mathbf{I}_P\right)+\lambda\mathbf{I}_{PR}. \tag{5.11}$$

We highlight that we update each row p of the matrix $\mathbf{A}$ by computing the corresponding row of the gradient (5.10).The algorithm follows in Algorithm 3. In line 12 of Algorithm 3, we compute the "local" Hessian matrix for each iteration l and row p, namely

$$\mathbf{H}_{l,p} = \mathbf{B}^T\mathtt{diag}\left(\widehat{\mathbf{M}}^l(p,:)\right)\mathbf{B}+\lambda\mathbf{I}, \tag{5.12}$$

and its maximum eigenvalue. For notational convenience, we denote Algorithm 3 as

$$\mathbf{A}_{\text{opt}} = \text{S\_NMC}(\mathbf{X},\mathbf{M},\mathbf{B},\mathbf{A}_*,\lambda).$$

AO Stochastic NTC algorithm

In Algorithm 4, we present the Stochastic AO NTC algorithm. We start from an initial point $\left\{\mathbf{A}_0^{(i)}\right\}_{i=1}^N$ and solve, in a circular manner, Matrix Least Squares problems, based on

Algorithm 3: Stochastic Nesterov-type algorithm for NMC

Input: $\mathbf{X}, \mathbf{M} \in \mathbb{R}^{P\times Q}$, $\mathbf{B} \in \mathbb{R}^{Q\times R}$, $\mathbf{A}_* \in \mathbb{R}^{P\times R}$, λ

$\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$
$l = 0$
while (1) **do**
  if $l >=$ MAX_INNER **then**
    break
  else
    for $p = 1 \dots P$ **do**
      $\widehat{\mathbf{M}}^l(p,:) = \text{sample}(\mathbf{M}(p,:))$
      $\mathbf{W}^l(p,:) = -\left(\widehat{\mathbf{M}}^l(p,:) \circledast \mathbf{X}(p,:)\right)\mathbf{B}$
      $\mathbf{Z}^l(p,:) = \left(\widehat{\mathbf{M}}^l(p,:) \circledast \left(\mathbf{Y}_l(p,:)\mathbf{B}^T\right)\right)\mathbf{B}$
      $\nabla f_{\widehat{\Omega}}(\mathbf{Y}_l(p,:)) = \mathbf{W}^l(p,:) + \mathbf{Z}^l(p,:) + \lambda \mathbf{Y}_l(p,:)$
      $\mathbf{H}_{l,p} = \mathbf{B}^T \texttt{diag}\left(\widehat{\mathbf{M}}^l(p,:)\right)\mathbf{B} + \lambda \mathbf{I}$
      $L_{l,p} = \max\left(\text{eig}\left(\mathbf{H}_{l,p}\right)\right)$
      $q_{l,p} = \frac{\lambda}{L_{l,p}}$
      $\mathbf{A}_{l+1}(p,:) = \left(\mathbf{Y}_l(p,:) - \frac{1}{L_p}\nabla f_\Omega(\mathbf{Y}_l(p,:))\right)_+$
      $\alpha^2_{l+1,p} = (1-\alpha_{l+1,p})\alpha^2_{l,p} + q_{l,p}\alpha_{l+1,p}$
      $\beta_{l+1,p} = \frac{\alpha_{l,p}(1-\alpha_{l,p})}{\alpha^2_{l,p}+\alpha_{l+1,p}}$
      $\mathbf{Y}_{l+1}(p,:) = \mathbf{A}_{l+1}(p,:) + \beta_{l+1,p}\left(\mathbf{A}_{l+1}(p,:) - \mathbf{A}_l(p,:)\right)$
    $l = l + 1$
return $\mathbf{A}_l$.

the previous estimates. We denote as $\mathbf{K}^{(i)} = \left(\mathbf{A}_k^{(N)} \odot \cdots \odot \mathbf{A}_k^{(i+1)} \odot \mathbf{A}_{k+1}^{(i-1)} \odot \cdots \odot \mathbf{A}_{k+1}^{(1)}\right)$.

Algorithm 4: Stochastic AO NTC

Input: $\boldsymbol{\mathcal{X}}$, Ω, $\left\{\mathbf{A}_0^{(i)}\right\}_{i=1}^N$, *rank* R.

$k = 0$
while (1) **do**
  for $i = 1, 2, \dots N$ **do**
    $\mathbf{A}_{k+1}^{(i)} = \text{S\_NMC}\left(\mathbf{X}_{(i)}, \mathbf{M}_{(i)}, \mathbf{K}^{(i)}, \mathbf{A}_k^{(i)}, \lambda\right)$
  if (term_cond is TRUE) **then** break; **endif**
  $k = k + 1$
return $\left\{\mathbf{A}_k^{(i)}\right\}_{i=1}^N$.

To update the matrix $\mathbf{A}_k^{(i)}$, we use the S_NMC($\mathbf{X}_{(i)}, \mathbf{M}_{(i)}, \mathbf{K}^{(i)}, \mathbf{A}_k^{(i)}, \lambda$) method, where we compute the following quantities.

Analysis of major computations

Computation of $\mathbf{W}^l(p,:)$

For every mode $i = 1 \dots N$, the p-th row of $\mathbf{W}_{(i)}$, which will be denoted as i_i from now on, can be computed as

$$\mathbf{W}^l_{(i)}(i_i,:) = -\left(\widehat{\mathbf{M}}^l(i_i,:) \circledast \mathbf{X}_{(i)}(i_i,:)\right)\mathbf{K}^{(i)}. \tag{5.13}$$

A direct implementation of this multiplication may be prohibitive since, in many applications, the values of I_i for $i = 1, 2, \dots N$ may be of the order of millions or even billions. In order to reduce the computational complexity, we use only the randomly sampled entries of $\mathbf{X}_{(i)}(i_i,:)$ and the respective rows of $\mathbf{K}^{(i)}$. More specifically, let the randomly sampled entries

$$\left(i_i, i_1^q + \cdots + i_{i-1}\prod_{m=1\dots i-2} I_m + i_{i+1}\prod_{m=1\dots i-1} I_m + \cdots + i_N^q \prod_{m=1\dots N-1}^{m\neq i} I_m\right) \in \widehat{\Omega}, \tag{5.14}$$

for $q = 1, \dots, B^l_{i,i_i}$ that correspond to $\mathbf{X}_{(i)}(i_i,:)$. For notational convenience, we set $c^l_q = \left(i_1^q + \cdots + i_{i-1}\prod_{m=1\dots i-2} I_m + i_{i+1}\prod_{m=1\dots i-1} I_m + \cdots + i_N^q \prod_{m=1\dots N-1}^{m\neq i} I_m\right)$. The c^l_q-th row of the Khatri-Rao product corresponds to

$$\mathbf{P}(c^l_q,:) = \mathbf{A}^{(N)}_k\left(i^q_N,:\right) \circledast \cdots \circledast \mathbf{A}^{(i+1)}_k\left(i^q_{i+1},:\right) \circledast \mathbf{A}^{(i-1)}_{k+1}\left(i^q_{i-1},:\right) \circledast \cdots \circledast \mathbf{A}^{(1)}_{k+1}\left(i^q_1,:\right). \tag{5.15}$$

Thus, the computation of the i_i-th row of $\mathbf{W}^l_{(i)}$ reduces to

$$\mathbf{W}^l_{(i)}\left(i_i,:\right) = \sum_{q=1}^{B^l_{i,i_i}} \mathbf{X}(i_i, c^l_q)\mathbf{P}(c^l_q,:). \tag{5.16}$$

For each row $\mathbf{W}^l_{(i)}(i_i,:)$ the computational complexity is $\mathcal{O}\left(B^l_{i,i_i} \cdot R\right)$. Thus, the overall complexity to compute $\mathbf{W}^l_{(i)}$ is $\mathcal{O}\left(B^l \cdot R\right)$.

Computation of $\mathbf{Z}^l(p,:)$

Following similar arguments, it can be shown that for each l-th inner iteration, the i_i-th row of $\mathbf{Z}_{(i)}$ can be computed as

$$\mathbf{Z}^l_{(i)}\left(i_i,:\right) = \sum_{q=1}^{B^l_{i,i_i}} \left(\mathbf{Y}^{(i)}_l\left(i_i,:\right)\mathbf{P}(c^l_q,:)^T\right)\mathbf{P}(c^l_q,:). \tag{5.17}$$

Similarly, analogous quantities can be computed for the update of the other factors. Each computation of $\mathbf{Z}^l_{(i)}\left(i_i,:\right)$, requires $\mathcal{O}\left(B^l_{i,i_i} \cdot R\right)$ arithmetic operations. The total complexity to compute the matrix $\mathbf{Z}^l_{(i)}$ is $\mathcal{O}\left(B^l \cdot R\right)$.

Computation of the "local" Hessian matrix and its maximum eigenvalue

In the sequel, we describe the computation of the Hessian matrix and its maximum eigenvalue. More specifically, we have

$$\mathbf{H}_{l,i_i}^{(i)} = \mathbf{K}^{(i)T}\texttt{diag}\left(\widehat{\mathbf{M}}^l(i_i,:)\right)\mathbf{K}^{(i)} + \lambda\mathbf{I}_R. \tag{5.18}$$

However, similarly to the previous quantities, we do not use the whole $\mathbf{K}^{(i)}$. In more detail, we compute

$$\mathbf{H}_{l,i_i}^{(i)} = \sum_{q=1}^{B_{i,i_i}^l} \mathbf{P}(c_q^l,:)^T\mathbf{P}(c_q^l,:) + \lambda\mathbf{I}_R, \tag{5.19}$$

which requires $\mathcal{O}(B_{i,i_i}^l \cdot R^2)$ operations. Therefore, the overall complexity to compute the matrix $\mathbf{H}_{l,i_i}^{(i)}$ for every row i_i, is $\mathcal{O}(B^l \cdot R^2)$. Instead of using the "classic" Eigenvalue Decomposition (which requires $\mathcal{O}(R^3)$ operations), we use the Power Iteration method to compute the maximum eigenvalue of the Hessian matrix $\mathbf{H}_{l,i_i}^{(i)}$, in order to determine the parameter L_{l,i_i}. At each iteration of the Power Iteration we perform $\mathcal{O}(R^2)$ arithmetic operations. The Power Iteration converges linearly to the "dominant" eigenvector, and the error asymptotically decreases by a ratio $\frac{\lambda_1}{\lambda_2}$, where $\lambda_1 > \lambda_2$ are the the most dominant eigenvalues [29]. We note that in our experiments we observed that $\lambda_1 \gg \lambda_2$ and $k_p < R$, where k_p denotes the number of iterations of the Power Iteration method. We conclude that the complexity of line 12 in Algorithm 3 over all P, is $\mathcal{O}\left((I_i + B^l) \cdot R^2\right)$.

Computation of the momentum parameter β

We also examine a variant of the Algorithm 3, where the momentum parameter β_{l,i_i} is computed as

$$\beta_{l,i_i} = \frac{1 - \sqrt{q_{l,i_i}}}{1 + \sqrt{q_{l,i_i}}}. \tag{5.20}$$

The difference in the performance using (5.20) and the one in lines 15-16 of Algorithm 3 is insignificant. However, in our experiments we use relation (5.20), since it is less complex.

5.1.2 Parallel Implementation of Stochastic AO NTC

In this Section, we proceed with the analysis of a parallel implementation of Stochastic AO NTC. We employ the OpenMP API which is suitable for our multi-threading approach. The update of each row can be done independently, therefore, lines 8-17 of Algorithm 3, can be computed separately by each available thread. More specifically, OpenMP provides the directive `#pragma omp parallel for`, which forks a team of threads to execute a block of code, that begins with a for loop. Iterations are divided among the available threads. Usually, block partitioning is selected, which means that, for a total of n iterations and t number of threads, each thread executes $\frac{n}{t}$ iterations. We provide a high level algorithmic sketch of our implementation.

Algorithm 5: Parallel Stochastic Nesterov-type algorithm for NMC

Input: $\mathbf{X}, \mathbf{M} \in \mathbb{R}^{P\times Q}$, $\mathbf{B} \in \mathbb{R}^{Q\times R}$, $\mathbf{A}_* \in \mathbb{R}^{P\times R}$, λ

1 $\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$
2 $l = 0$
3 **while** (1) **do**
4 **if** $l >=$ MAX_INNER **then**
5 break
6 **else**
7 #pragma omp parallel
8 **for** $p = 1 \dots P$ **do**
9 lines 9-17 of Algorithm 3 ...
10 $l = l + 1$
11 **return** $\mathbf{A}_l$.

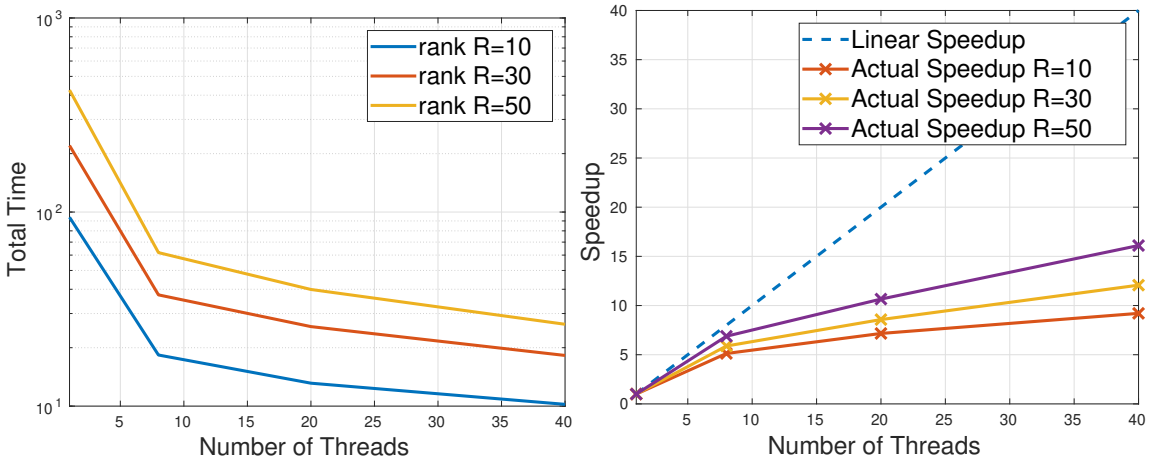


Figure 5.1: Execution time(Left), Attained Speedup(Right) for Chicago Crime dataset.

5.2 Numerical Experiments

We consider the real sparse dataset Chicago Crime, which concerns crime reports in the city of Chicago starting from January 1st 2001 up to December 11th, 2017 [30]. Data are arranged in a 4–th order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}_+^{6,186\times 24\times 77\times 32}$ with 5330673 non-zeros. Each mode of the Chicago crime tensor represents a specific feature. In more detail, the modes correspond to `day-hour-community-crimetype`, where `community` is one of the communities of Chicago, and the non-zeros represent the number of reports of a specific type of crime. In Figure 5.1, we plot the execution time and the attained speedup for the Chicago Crime dataset for $t = 1, 8, 20, 40$.

5.2.1 Model Selection problem

Real Data

We test our algorithm on the MovieLens 10M dataset [31], which contains 10000054 ratings of movies by 71567 users. More specifically, we arrange the MovieLens dataset in a 3–rd order tensor of size $71567 \times 65133 \times 730$, with the last dimension denoting the timestamp

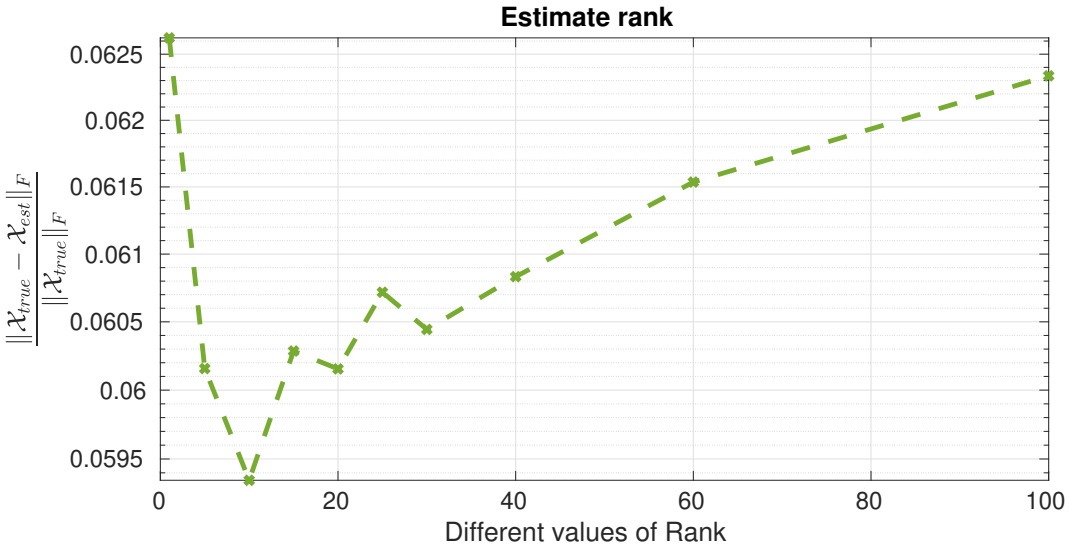


Figure 5.2: RFE vs different values of Rank.

of the rating organized in seven-day-wide ranges.

We use the Stochastic AO NTC in order to choose the appropriate model for our data. We test our method using different values for the rank R. Namely, inspired by the work in [32] and [33], we set $R = 1, 5, 10, 15, 20, 25, 30, 40, 60, 100$. In more detail, we first split the MovieLens dataset into two sets. The first one is the train set, which we will use to train our model for the different values of rank, and the test set, to assess our predictions. We keep the convention of 80% - 20% for the train and test sets. The outer iterations are set to $k = 20$, where as MAX_INNER $= 1$, to prevent the overfitting on the sampled entries. Also, we set $B^l = 0.3\,|\Omega|$ and $\lambda = 0.001$. As a metric we use the relative factorization error (RFE)

$$\text{RFE} = \frac{\|\boldsymbol{\mathcal{X}}_{true} - \boldsymbol{\mathcal{X}}_{est}\|_F}{\|\boldsymbol{\mathcal{X}}_{true}\|_F}, \tag{5.21}$$

where both $\boldsymbol{\mathcal{X}}_{true}, \boldsymbol{\mathcal{X}}_{est}$ have non zero values in the positions indicated by the test set. In Figure 5.2, we illustrate the results of the aforementioned model selection process. We observe that $R = 10$, gives the lowest RFE on the test set, thus we can claim that $R = 10$ is a good choice for our data.

Synthetic Data

Similarly, we evaluate the performance of our stochastic algorithm on the aforementioned model selection problem for synthetic data. We create a nonnegative sparse tensor $\boldsymbol{\mathcal{X}}$ of dimensions $500 \times 500 \times 500$ with true rank $R = 10$ and 0.1% nonzero entries. In addition, noise is added to the tensor entries, resulting in a SNR = 15dB noisy setting. We solve the NTC problem using Algorithm 5 by setting $R = 1, 5, 8, 10, 12, 15, 20, 35, 50$. We set $\lambda = 0.01$, the maximum number of outer iterations k is set to 300, MAX_INNER $= 1$ and $B^l = 0.5\,|\Omega|$. We again split to 80%-20% for the train and the test set. RFE is used again as the performance metric. In Figure 5.3, we illustrate the results.

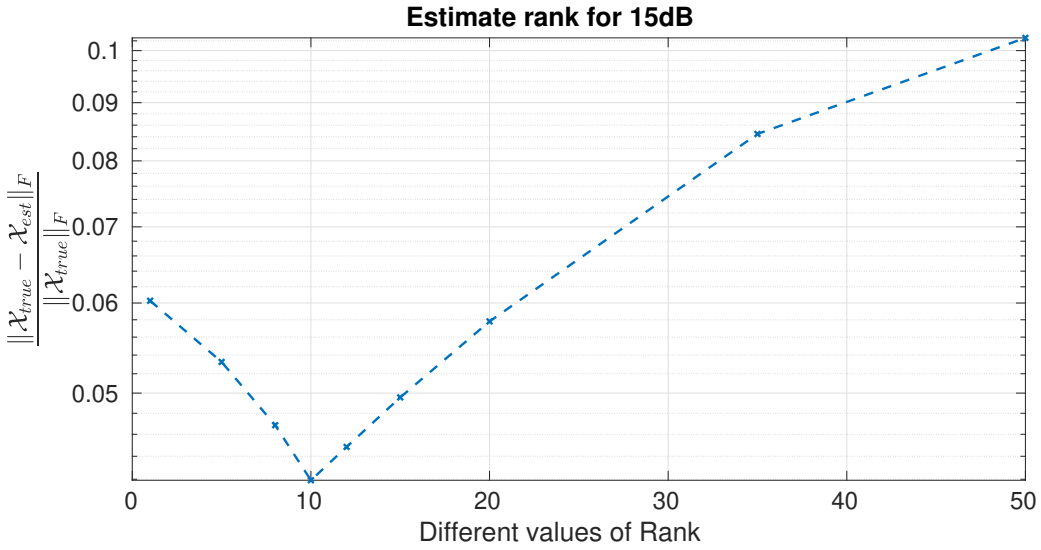


Figure 5.3: RFE vs different values of Ranks.

5.3 Conclusions

We considered the Nonnegative Tensor Completion problem and an efficient stochastic method to solve it. Through different problem settings and numerical experiments, we conclude that our stochastic approach of NTC can be a competitive method in the fields of Optimization and Machine Learning.

Chapter 6

Conclusion and Future Work

In this work, we explored the efficiency of Stochastic Optimization in conjunction to Tensor Factorization/Completion problems. For dense tensors, we proposed a new stochastic method where Nesterov acceleration was employed to improve the performance of the existing stochastic algorithms. Furthermore, we considered a stochastic variation of NAG to solve the NTC problem and its parallel implementation using the OpenMP API. Through numerical experiments, we showed the efficiency of both algorithms on NTF/NTC problems.

In Chapter 4, we explored the limitations of a parallel implementation of ASCPD. Also, we considered an alternative of ASCPD which can be solved in parallel, with the same accuracy as the serial ASCPD. However, since randomized problems are memory-bounded, the parallel version seems to not attend the same scalability as its deterministic counterparts. The development of a better parallel scheme of ASCPD is a future work subject.

Bibliography

[1] C. F. Beckmann and S. M. Smith, "Tensorial extensions of independent component analysis for multisubject fmri analysis," *Neuroimage*, vol. 25, no. 1, pp. 294–311, 2005.

[2] S. Rabanser, O. Shchur, and S. Günnemann, "Introduction to tensor decompositions and their applications in machine learning," *arXiv preprint arXiv:1711.10781*, 2017.

[3] A. M. S. Ang, J. E. Cohen, N. Gillis, and L. T. K. Hien, "Accelerating block coordinate descent for nonnegative tensor factorization," *arXiv preprint arXiv:2001.04321*, 2020.

[4] N. Vervliet, A. Themelis, P. Patrinos, and L. De Lathauwer, "A quadratically convergent proximal algorithm for nonnegative tensor decomposition," *arXiv preprint arXiv:2003.03502*, 2020.

[5] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan, "Large-scale parallel collaborative filtering for the netflix prize," in *International conference on algorithmic applications in management*. Springer, 2008, pp. 337–348.

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

[7] 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.

[8] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[9] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018.

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

[11] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in neural information processing systems*, 2013, pp. 315–323.

[12] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Advances in neural information processing systems*, 2014, pp. 1646–1654.

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[14] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[15] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of machine learning research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[16] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing finite sums with the stochastic average gradient," *Mathematical Programming*, vol. 162, no. 1-2, pp. 83–112, 2017.

[17] I. Gitman, H. Lang, P. Zhang, and L. Xiao, "Understanding the role of momentum in stochastic gradient methods," in *Advances in Neural Information Processing Systems*, 2019, pp. 9630–9640.

[18] Z. Allen-Zhu, "Katyusha: The first direct acceleration of stochastic gradient methods," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 8194–8244, 2017.

[19] C. I. Kanatsoulis and N. D. Sidiropoulos, "Large-scale canonical polyadic decomposition via regular tensor sampling," in *2019 27th European Signal Processing Conference (EUSIPCO)*, 2019, pp. 1–5.

[20] N. Vervliet and L. De Lathauwer, "A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 2, pp. 284–295, 2015.

[21] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "Parcube: Sparse parallelizable tensor decompositions," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2012, pp. 521–536.

[22] N. D. Sidiropoulos, E. E. Papalexakis, and C. Faloutsos, "Parallel randomly compressed cubes: A scalable distributed architecture for big tensor decomposition," *IEEE Signal Processing Magazine*, vol. 31, no. 5, pp. 57–70, 2014.

[23] C. Battaglino, G. Ballard, and T. G. Kolda, "A practical randomized cp tensor decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 39, no. 2, pp. 876–901, 2018.

[24] X. Fu, S. Ibrahim, H.-T. Wai, C. Gao, and K. Huang, "Block-randomized stochastic proximal gradient for low-rank tensor factorization," *IEEE Transactions on Signal Processing*, 2020.

[25] N. Parikh, S. Boyd *et al.*, "Proximal algorithms," *Foundations and Trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[26] A. P. Liavas, G. Kostoulas, G. Lourakis, K. Huang, and N. D. Sidiropoulos, "Nesterov-based alternating optimization for nonnegative tensor factorization: Algorithm and parallel implementation," *IEEE Transactions on Signal Processing*, vol. 66, no. 4, pp. 944–953, 2017.

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

[28] P. Jiang and G. Agrawal, "Adaptive periodic averaging: A practical approach to reducing communication in distributed learning," 2020.

[29] S. F. Quarteroni A., Sacco R., *Numerical Mathematics*, vol. 37, ch. Approximation of Eigenvalues and Eigenvectors (5.3.1).

[30] S. Smith, J. W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, and G. Karypis. (2017) FROSTT: The formidable repository of open sparse tensors and tools. [Online]. Available: http://frostt.io/

[31] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, Dec. 2015. [Online]. Available: https://doi.org/10.1145/2827872

[32] G. Lourakis and A. Liavas, "Nesterov-based alternating optimization for nonnegative tensor completion: Algorithm and parallel implementation," 06 2018, pp. 1–5.

[33] L. Karlsson, D. Kressner, and A. Uschmajew, "Parallel algorithms for tensor completion in the cp format," *Parallel Computing*, vol. 57, pp. 222–234, 2016.