TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Efficient Optimization Algorithms for Large Tensor Processing and Applications

by

Ioannis Marios Papagiannakos

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

ELECTRICAL AND COMPUTER ENGINEERING

February 2022

THESIS COMMITTEE

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

Abstract

We consider the problem of nonnegative tensor completion. We adopt the alternating optimization framework and solve each nonnegative matrix least-squares with missing elements problem via a stochastic variation of the accelerated gradient algorithm, where we propose and experimentally test the efficiency of various step-sizes. We develop a parallel shared-memory implementation of our algorithm using the multi-threaded API OpenMP, which attains significant speedup. We test the effectiveness and the performance of our algorithm using both real-world and synthetic data. We focus on real-world applications that can be interpreted as nonnegative tensor completion problems. We believe that our approach is a very competitive candidate for the solution of very large nonnegative tensor completion problems.

Acknowledgements

This thesis would not have been possible without the support of many people.

First of all, I would like to thank my thesis supervisor, Professor Athanasios Liavas, for his encouragement and continuous guidance throughout this work. Also thanks to my committee members, Professor George N. Karystinos and Associate Professor Vasilis Samoladas for accepting to evaluate the work presented in this thesis as members of my thesis committee.

Also, my friends and colleagues Chris Kolomvakis, Nina Siaminou, Paris Karakasis and Margarita Psychountaki, for all the assistance and support they provided during my thesis.

I would like to thank my family, Anna Maria, Elpida, and Nikos for their support and encouragement throughout my study years. Also, I want to thank my friends and especially George K., George X., Vagelis K., and Evi S. for their moral support and all the fun we had throughout these years.

Finally, part of this work was partially supported by the European Regional Development Fund of the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship, and Innovation, under the call RESEARCH - CREATE - INNOVATE (project code : $\mathrm{T1E\Delta K-03360}$).

Table of Contents

List of Figures

List of Abbreviations

ALS	Alternating Least Squares
API	Application Programming Interface
AO	Alternating Optimization
CANDECOMP	Canonical Decomposition
CPD	Canonical Polyadic Decomposition
i.i.d.	independent and identically distributed
MLS	Matrix Least-Squares
NMLS	Nonnegative Matrix Least-Squares
NMNLSME	Nonnegative Matrix Least-Squares with Missing Elements
NLP	Natural Language Processing
NTC	Nonnegative Tensor Completion
NTF	Nonnegative Tensor Factorization
OpenMP	Open Multi-Processing
PARAFAC	Parallel Factor Analysis
RFE	Relative Factorization Error
RIFE	Relative Incomplete Factorization Error
RMSE	Root Mean Square Error
SGD	Stochastic Gradient Descend

Chapter 1

Introduction

1.1 Problem Description (Tensor factorization)

Tensors are mathematical structures that can be described as multidimensional arrays of numerical values and, therefore, generalize matrices to multiple dimensions. Tensors and tensor decompositions are important tools that can model multi–way data dependencies. [1], [2], [3], [4]. Tensor factorization (or decomposition) into latent factors is very important for numerous tasks, such as feature selection, dimensionality reduction, compression, data visualization, and interpretation. Tensor factorizations are usually computed as solutions of optimization problems [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. Tensor Completion (TC) arises in many modern applications such as machine learning, signal processing, and scientific computing. We focus on the CPD model and consider the nonnegative tensor completion (NTC) problem, using as quality metric the Frobenius norm of the difference between the true and the estimated tensor. We adopt the Alternating Optimization (AO) framework, that is, we work in a circular manner and update each factor by keeping all other factors fixed. We update each factor by solving a nonnegative matrix completion (NMC) problem via a stochastic variant of the accelerated (Nesterov–type) gradient [5].

1.1.1 Notation

Vectors, matrices, and tensors are denoted by small, capital, and calligraphic capital letters, respectively; for example, $\mathbf{x}$, $\mathbf{X}$, and $\mathcal{X}$. $\mathbb{R}^{I_1\times\cdots\times I_N}$ and $\mathbb{R}_+^{I_1\times\cdots\times I_N}$ denote, respectively, the set of $(I_1 \times \cdots \times I_N)$ real and nonnegative tensors. The elements of tensor $\mathcal{X}$ are denoted as $\mathcal{X}(i_1,\ldots,i_N)$. In many cases, we use Matlab–like notation, for example, $\mathbf{A}(j,:)$ denotes the j-th row of matrix $\mathbf{A}$. The transpose of a matrix $\mathbf{A}$ is denoted by $\mathbf{A}^T$. $\mathbf{I}_P$ denotes the $(P \times P)$ identity matrix [1], and $\mathbf{O}_{M,N}$ denotes the zero matrix (matrix in which all of the entries are 0) of dimensions $M \times N$. $\|\cdot\|_F$ denotes the Frobenius norm of the tensor or matrix argument, $(\mathbf{A})_+$ denotes the projection of matrix $\mathbf{A}$ onto the set of elementwise nonnegative matrices, and $\mathbf{A} \geq \mathbf{0}$ denotes a matrix $\mathbf{A}$ with nonnegative elements. Finally, $\mathbb{N}_N$ denotes the set $\{1,\ldots,N\}$.

The outer product of vectors $\mathbf{a}$ and $\mathbf{b}$ is denoted as $\mathbf{a}\circ\mathbf{b}$. The Kronecker, Khatri–Rao, and Hadamard products of matrices $\mathbf{A}$ and $\mathbf{B}$ (with appropriate dimensions) are denoted, respectively, as $\mathbf{A}\otimes\mathbf{B}$, $\mathbf{A}\odot\mathbf{B}$, and $\mathbf{A}\circledast\mathbf{B}$. The extension of the notation of these operations

[1]If the dimension becomes clear from the context, we omit the subscript.

to more than two arguments is obvious.

1.1.2 Basic definitions

Definition 1.1 *The **Kronecker product** of matrices* $\mathbf{A} \in \mathbb{R}^{I \times R}$ *and* $\mathbf{B} \in \mathbb{R}^{J \times P}$ *is denoted as* $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{IJ \times RP}$ [3]*, and is computed as follows*

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} \mathbf{A}(1,1)\mathbf{B} & \dots & \mathbf{A}(1,J)\mathbf{B} \\ \vdots & \ddots & \vdots \\ \mathbf{A}(I,1)\mathbf{B} & \dots & \mathbf{A}(I,J)\mathbf{B} \end{bmatrix} \in \mathbb{R}^{IJ \times RP}. \tag{1.1}$$

Definition 1.2 *The **Khatri–Rao (or column–wise Kronecker) product** of matrices* $\mathbf{A} \in \mathbb{R}^{I \times R}$ *and* $\mathbf{B} \in \mathbb{R}^{J \times R}$ *is denoted as* $\mathbf{A} \odot \mathbf{B} \in \mathbb{R}^{IJ \times R}$*. The Khatri–Rao product is computed as*

$$\mathbf{A} \odot \mathbf{B} = \Big[\mathbf{A}(:,1) \otimes \mathbf{B}(:,1) \dots \mathbf{A}(:,R) \otimes \mathbf{B}(:,R)\Big] \in \mathbb{R}^{IJ \times R}. \tag{1.2}$$

Definition 1.3 *The **Hadamard (or element–wise) product** of matrices* $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{I \times R}$ *is denoted as* $\mathbf{A} \circledast \mathbf{B} \in \mathbb{R}^{I \times R}$*. The Hadamard product of an element in* i^{th} *row and* j^{th} *column is computed as*

$$[\mathbf{A} \circledast \mathbf{B}](i,j) = \mathbf{A}(i,j) \cdot \mathbf{B}(i,j) \tag{1.3}$$

1.1.3 Structure

This thesis is organized as follows. At first, in Chapter 2 we introduce some background on Matrix Least Squares problems. We briefly present the unconstrained problem, the Nonnegative Matrix Least Squares problem and focus on Nonnegative Matrix Least Squares with Missing Elements (NMLSME), which is the building block for the solution of the Nonnegative Tensor Completion (NTC) problem. We analyze a first–order accelerated gradient method to get an approximate solution.

In Chapter 3 we discuss the Stochastic Gradient Descend method and we propose a stochastic variant for the NMLSME, using various step–sizes.

In Chapter 4 we discuss the CP Decomposition. We start by giving some mathematical background on tensors, and proceed to present the unconstrained and nonnegative PARAFAC models. Then, we focus on the NTC problem, where we solve it through our stochastic NMLSME. We also propose a method for factorizing supersymmetric and nonnegative incomplete tensors.

In Chapter 5 we propose a parallel, multi–threaded, scheme for the stochastic NMLSME method via the OpenMP API.

Chapter 6 is dedicated to real–world applications which can be interpreted as NTC problems. We focus on two applications, the restoration of images with missing pixels and the extraction of word embeddings in the Natural Language Process.

Next, in Chapter 7 we test the effectiveness of our proposed methods on synthetic and real–world datasets. In cases where it is possible, we compare them with other competitive methods and present the respective results.

Finally, in Chapter 8, we conclude the thesis and propose future work.

Chapter 2

Matrix Least Squares Problem

We begin with the matrix least–squares problem, which will be the main building block for the tensor factorization.

2.1 Matrix Least Squares (MLS)

Let matrices $\mathbf{X} \in \mathbb{R}^{I\times J}$ and $\mathbf{B} \in \mathbb{R}^{J\times R}$, and consider the MLS problem

$$\min_{\mathbf{A}\in\mathbf{R}} f(\mathbf{A}) = \frac{1}{2}\,\|\mathbf{X} - \mathbf{A}\mathbf{B}^T\|_F^2, \tag{2.1}$$

where matrix $\mathbf{A} \in \mathbb{R}^{I\times R}$. The cost function f is convex, and therefore, the existence of a global minimizer of f, $\mathbf{A}_{opt}$, is guaranteed. The gradient and the Hessian of function f, at point $\mathbf{A}$, are given by

$$\nabla f(\mathbf{A}) = -\left(\mathbf{X} - \mathbf{A}\mathbf{B}^T\right)\mathbf{B} \tag{2.2}$$

and

$$\nabla^2 f(\mathbf{A}) := \frac{\partial^2 f(\mathbf{A})}{\partial \mathrm{vec}(\mathbf{A})\partial \mathrm{vec}(\mathbf{A})^T} = \mathbf{B}^T\mathbf{B} \otimes \mathbf{I}_P. \tag{2.3}$$

If we do not impose any constraints on matrix $\mathbf{A}_{opt}$, then from (2.2), it must satisfy the following equation

$$\nabla f(\mathbf{A}_{opt}) = \mathbf{0} \to -(\mathbf{X} + \mathbf{A}_{opt}\mathbf{B}^T)\mathbf{B} = \mathbf{0}. \tag{2.4}$$

Equations (2.4) are equivalent to

$$\mathbf{X}\mathbf{B} = \mathbf{A}_{opt}\mathbf{B}^T\mathbf{B} \tag{2.5}$$

which are known as normal equations. If $\mathbf{B}^T\mathbf{B}$ is invertible, the solution $\mathbf{A}_{opt}$ is given by

$$\mathbf{A}_{opt} = \mathbf{X}\mathbf{B}\left(\mathbf{B}^T\mathbf{B}\right)^{-1}. \tag{2.6}$$

2.2 Nonnegative Matrix Least Squares (NMLS)

2.2.1 Preliminaries

In the case where we impose nonnegative constraints on matrix $\mathbf{A}_{opt} = \mathbb{R}_+^{I\times R}$, the solution cannot be expressed in closed form, and thus, we must resort to an iterative algorithm. We use an accelerated gradient method because it is very efficient in practice and is suitable

Algorithm 1: Accelerated gradient algorithm for L-smooth μ-strongly convex problems (using Nesterov's constant scheme II).

```
   Input: $\mathbf{x}_0 \in \mathbb{R}^N$, $\mu$, $L$. Set $\mathbf{y}_0 = \mathbf{x}_0$, $q = \frac{\mu}{L}$ and choose $a_0 \in (0,1)$
1  $k$-th iteration
2      $\mathbf{x}_{k+1} = \left(\mathbf{y}_k - \frac{1}{L}\nabla f(\mathbf{y}_k)\right)_{\mathbb{X}}$
3      Compute $a_{k+1} \in (0,1)$ from $a_{k+1}^2 = (1-a_{k+1})a_k^2 + q a_{k+1}$ and set $\beta_k = \frac{a_k(1-a_k)}{(a_k^2+a_{k+1})}$
4      $\mathbf{y}_{k+1} = \mathbf{x}_{k+1} + \beta_k(\mathbf{x}_{k+1} - \mathbf{x}_k)$
```

Algorithm 2: Accelerated gradient algorithm for L-smooth μ-strongly convex problems (using Nesterov's constant scheme III).

```
   Input: $\mathbf{x}_0 \in \mathbb{R}^N$, $\mu$, $L$. Set $\mathbf{y}_0 = \mathbf{x}_0$, $\beta = \frac{\sqrt{\frac{L}{\mu}}-1}{\sqrt{\frac{L}{\mu}}+1}$.
1  $k$-th iteration
2      $\mathbf{x}_{k+1} = \left(\mathbf{y}_k - \frac{1}{L}\nabla f(\mathbf{y}_k)\right)_{\mathbb{X}}$
3      $\mathbf{y}_{k+1} = \mathbf{x}_{k+1} + \beta(\mathbf{x}_{k+1} - \mathbf{x}_k)$
```

for parallel implementation [6].

We begin by introducing the class of L-smooth μ-strongly convex optimization problems. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a smooth (differentiable up to some desired order) convex function and $\mathbb{X}$ a closed convex subset of $\mathbb{R}^n$. We aim to solve the problem

$$\min_{\mathbf{x}\in\mathbb{X}} f(\mathbf{x}), \tag{2.7}$$

within accuracy $\epsilon > 0$, that is, to find a point $\mathbf{x}_{opt} \in \mathbb{X}$ such that $f(\mathbf{x}_{opt}) - f^* \leq \epsilon$, where $f^* := \min\limits_{\mathbf{x}\in\mathbb{X}} f(\mathbf{x})$.

Let $0 < \mu \leq L < \infty$. A smooth convex function f is called L-smooth μ-strongly convex if [5, p. 65]

$$\mu\mathbf{I} \preceq \nabla^2 f(\mathbf{x}) \preceq L\mathbf{I}, \quad \forall\, \mathbf{x} \in \mathbb{R}^n. \tag{2.8}$$

The information complexity of black–box first–order methods for this class of problems is $O\left(\sqrt{\frac{L}{\mu}}\log\frac{1}{\epsilon}\right)$, where $\frac{L}{\mu}$ is the condition number of the problem [5, Theorem 2.2.2]. A first–order optimal algorithm appears in Algorithms 1 and 2. Algorithm 1 becomes 2 if we choose $a_0 = \sqrt{\frac{\mu}{L}}$. Then $a_k = \sqrt{\frac{\mu}{L}}$ and $\beta_k = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$. Algorithm 1 works for $\mu = 0$ [5, p.80].

We note that $(\mathbf{x})_{\mathbb{X}}$ denotes the projection of vector $\mathbf{x}$ onto set $\mathbb{X}$ (see also [5, p. 80]). The projection onto set $\mathbb{R}_+$ is easy to compute using the max operator

$$(\mathbf{A})_+ = \max(\mathbf{A}, \mathbf{O}_{I,R}).$$

If the projection onto set $\mathbb{X}$ is easy to compute, then this algorithm is both theoretically optimal and very efficient in practice.

2.2.2 Nesterov–type algorithm for NMLS with proximal term

We present an optimal first–order algorithm for the solution of L-smooth μ-strongly convex NMLS problems. For the nonnegative case, problem (2.1) is expressed as follows. Let matrices $\mathbf{X} \in \mathbb{R}^{I \times J}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, $\mathbf{A} \in \mathbb{R}^{I \times R}$, appropriately chosen $\lambda > 0$ [6]. and the following NMLS problem

$$\min_{\mathbf{A} \geq \mathbf{0}} f(\mathbf{A}) = \frac{1}{2} \|\mathbf{X} - \mathbf{A}\mathbf{B}^T\|_F^2 + \frac{\lambda}{2} \|\mathbf{A} - \mathbf{A}_*\|_F^2, \tag{2.9}$$

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

$$\nabla f_{\mathrm{P}}(\mathbf{A}) = -\left(\mathbf{X} - \mathbf{A}\mathbf{B}^T\right)\mathbf{B} + \lambda(\mathbf{A} - \mathbf{A}_*). \tag{2.10}$$

We choose λ based on the eigenvalue decomposition of matrix $\mathbf{B}^T\mathbf{B}$. Thus we have $L := \max(\mathrm{eig}(\mathbf{B}^T\mathbf{B}))$ and $\mu := \min(\mathrm{eig}(\mathbf{B}^T\mathbf{B}))$, and denote this functional dependence as $\lambda = g(L, \mu)$. If $\frac{\mu}{L} \ll 1$, then we set $\lambda \approx 10\mu$, significantly improving the conditioning of the problem by putting large weight on the proximal term. However, in this case, we expect that the optimal point will be biased towards $\mathbf{A}_*$. Otherwise, we set $\lambda \lessapprox \mu$, putting small weight on the proximal term and permitting significant progress towards the computation of $\mathbf{A}$ that satisfies approximate equality $\mathbf{X} \approx \mathbf{A}\mathbf{B}^T$ as accurately as possible.

An accelerated gradient algorithm for the solution of the NMLS problem with proximal term (2.9) is given in (3), based on Algorithm 2.

Algorithm 3: Accelerated gradient algorithm for NMLS problems with proximal term.

```
   Input: $\mathbf{X} \in \mathbb{R}^{I \times J}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, $\mathbf{A}_* \in \mathbb{R}^{I \times R}$
1  $L = \max(\mathrm{eig}(\mathbf{B}^T\mathbf{B}))$, $\mu = \min(\mathrm{eig}(\mathbf{B}^T\mathbf{B}))$
2  $\lambda = g(L, \mu)$, $\beta = \frac{\sqrt{\frac{L+\lambda}{\mu+\lambda}}-1}{\sqrt{\frac{L+\lambda}{\mu+\lambda}}+1}$
3  $\mathbf{W} = -\mathbf{X}\mathbf{B} - \lambda\mathbf{A}_*,\ \mathbf{Z} = \mathbf{B}^T\mathbf{B} + \lambda\mathbf{I}$
4  $\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$
5  $k = 0$
6  while (terminating condition is FALSE) do
7      $\nabla f_{\mathrm{P}}(\mathbf{Y}_k) = \mathbf{W} + \mathbf{Y}_k\mathbf{Z}$
8      $\mathbf{A}_{k+1} = \left(\mathbf{Y}_k - \frac{1}{L+\lambda}\nabla f_{\mathrm{P}}(\mathbf{Y}_k)\right)_+$
9      $\mathbf{Y}_{k+1} = \mathbf{A}_{k+1} + \beta\left(\mathbf{A}_{k+1} - \mathbf{A}_k\right)$
10     $k = k + 1$
11 return $\mathbf{A}_k$.
```

2.2.3 Complexities

The computational complexity of Algorithm 3 is as follows. Matrices $\mathbf{W}$ and $\mathbf{Z}$ are computed once per algorithm call and cost, respectively, $O(IJR)$ and $O(JR^2)$ arithmetic operations. Quantities L and μ are also computed once per algorithm call and in the worst case demand $O(R^3)$ operations. In every iteration with cost $O(IR^2)$, $O(IR)$, and $O(IR)$

arithmetic operations, $\nabla f_{\mathrm{P}}(\mathbf{Y}_k)$, $\mathbf{A}_k$, and $\mathbf{Y}_k$ are updated, respectively. If $\min(I, J) \gg R$, then the computation of the matrix product $\mathbf{XB}$ is the most demanding operation.

2.3 Nonnegative Matrix Least Squares with Missing Elements (NMLSME)

We now proceed to the next topic, which is the Nonnegative Matrix Least Squares with Missing Elements (NMLSME). The solution of the NMLSME problem will be our building block for the solution of the Tensor Completion problem. Let $\mathbf{X} \in \mathbb{R}^{I\times J}$, $\mathbf{A} \in \mathbb{R}^{I\times R}$, and $\mathbf{B} \in \mathbb{R}^{J\times R}$. Let $\Omega \subseteq \mathbb{N}_I \times \mathbb{N}_J$ be the set of indices of the known elements of $\mathbf{X}$ and let $\mathbf{M}$ 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 $\mathbf{X}$.

$$f_\Omega(\mathbf{A}) = \frac{1}{2}\left\|\mathbf{M} \circledast \left(\mathbf{X} - \mathbf{A}\mathbf{B}^T\right)\right\|_F^2 + \frac{\lambda}{2}\left\|\mathbf{A}\right\|_F^2. \tag{2.11}$$

We consider the problem

$$\min_{\mathbf{A}\geq\mathbf{0}} f_\Omega(\mathbf{A}) \tag{2.12}$$

The gradient and the Hessian of f_Ω, at point $\mathbf{A}$, are given by

$$\nabla f_\Omega(\mathbf{A}) = -\left(\mathbf{M} \circledast \mathbf{X} - \mathbf{M} \circledast \left(\mathbf{A}\mathbf{B}^T\right)\right)\mathbf{B} + \lambda\mathbf{A} \tag{2.13}$$

and

$$\nabla^2 f_\Omega(\mathbf{A}) = \left(\mathbf{B}^T \otimes \mathbf{I}_P\right)\operatorname{diag}\left(\operatorname{vec}\left(\mathbf{M}\right)\right)\left(\mathbf{B} \otimes \mathbf{I}_P\right) + \lambda\mathbf{I}_{PR}. \tag{2.14}$$

Algorithm 4: Nesterov–type algorithm for the nonnegative MLSME problem.

```
   Input: $\mathbf{X}, \mathbf{M} \in \mathbb{R}^{P\times Q}$, $\mathbf{B} \in \mathbb{R}^{Q\times R}$, $\mathbf{A}_* \in \mathbb{R}^{P\times R}$, $\lambda$, $\mu$, $L$
1  $\mathbf{W} = -(\mathbf{M} \circledast \mathbf{X})\mathbf{B}$
2  $C = \frac{L+\lambda}{\mu+\lambda}$, $\beta = \frac{\sqrt{C}-1}{\sqrt{C}+1}$
3  $\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$
4  $l = 0$
5  while (terminating condition is FALSE) do
6      $\mathbf{Z}_l = \left(\mathbf{M} \circledast \left(\mathbf{Y}_l\mathbf{B}^T\right)\right)\mathbf{B}$
7      $\nabla f_\Omega(\mathbf{Y}_l) = \mathbf{W} + \mathbf{Z}_l + \lambda\mathbf{Y}_l$
8      $\mathbf{A}_{l+1} = \left(\mathbf{Y}_l - \frac{1}{L+\lambda}\nabla f_\Omega(\mathbf{Y}_l)\right)_+$
9      $\mathbf{Y}_{l+1} = \mathbf{A}_{l+1} + \beta\left(\mathbf{A}_{l+1} - \mathbf{A}_l\right)$
10     $l = l + 1$
11 return $\mathbf{A}_l$.
```

We are interested only in the nonnegative case, where we solve the MLSME problem using the Nesterov–type algorithm of 4. We observe that this algorithm is much more complicated than 3, mainly because of the computations in line 6, which must be repeated in every iteration.

A key point of the algorithm is the assignment of values to parameters μ and L. If we denote the optimal values as μ^* and L^*, then it turns out that $\mu^* + \lambda$ and $L^* + \lambda$ are, respectively, equal to the smallest and the largest eigenvalue of $\nabla^2 f_\Omega$. As the size of the problem grows, the computation of μ^* and L^* becomes very demanding.

A simple approximation is to set $\mu = 0$ and $L = \max(\mathrm{eig}(\mathbf{B}^T\mathbf{B}))$, which in the cases of small R can be easily computed. We have observed that, in practice, our choice for μ is very accurate for very sparse problems, while our choice for L is an easily computed upper bound for L^*. More recently in [7], we proposed a more computationally demanding but more effective approach, as presented in Algorithm 4. We compute the rows $\mathbf{A}_{l+1}(j,:)$ and $\mathbf{Y}_{l+1}(j,:)$, for $j \in \mathbb{N}_P$, using $\mu = 0$, $L_j = \max(\mathrm{eig}(\mathbf{G}_j))$, where $\mathbf{G}_j$ is the second derivative of $f_\Omega(\mathbf{A})$ with respect to $\mathbf{A}(j,:)$, and is defined as

$$\mathbf{G}_j := \mathbf{B}^T \mathrm{diag}(\mathbf{M}(j,:))\,\mathbf{B}. \tag{2.15}$$

We also use the respective β_j for each row. It is important to mention that this adds an overhead to the algorithm since the computation of $\mathbf{G}_j$, for $j \in \mathbb{N}_P$, is not directly required in Algorithm 4, but leads to very useful step sizes, which will be discussed in the next chapter. Also, in the next chapter, we will present a stochastic variant of this algorithm.

2.3.1 Complexities

The computational complexity of Algorithm 4 is as follows. Matrix $\mathbf{W}$ demands $O(|\Omega|R)$ arithmetic operations once per algorithm call. Matrix $\mathbf{Z}$ also requires $O(|\Omega|R)$ arithmetic operations per inner iteration l. In every iteration, matrices $\nabla f_{\mathsf{P}}(\mathbf{Y}_k)$, $\mathbf{A}_k$, and $\mathbf{Y}_k$ are updated with cost of $O(IR)$ arithmetic operations. Finally, computation of matrix $\mathbf{G}_j$ requires $O(|\Omega|R^2)$ arithmetic operations.

Chapter 3

Stochastic Gradient Descent Methods

3.1 Stochastic Gradient Descent algorithm (SGD)

Stochastic approximation methods are a family of iterative methods, typically used for root–finding problems or for optimization problems, and were introduced in the early '50s by [8]. Even though it is known for a long time, this method has recently gained great popularity, since it is the preferred optimization method in modern machine learning. In practice, they only require the gradient for one training example (a small "mini–batch" of examples) in each iteration and thus can be used with large datasets, due to their lower computational complexity.

We aim to minimize a differentiable function f assuming access to noisy stochastic gradients (noisy estimates of the full gradient) of the function. We focus on functions f with a finite–sum structure, expressed as

$$f(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{x}), \tag{3.1}$$

where n is the size of the training set and the function f_i is the loss function for a training point i. We assume that f is lower–bounded by some value f^* and that f is L-smooth implying that the gradient ∇f is L-Lipschitz continuous [9].

The problem of minimizing a function $f : \mathbb{R}^n \to \mathbb{R}$. The commonly used procedure to optimize f is to iteratively adjust $\mathbf{x}_t \in \mathbb{R}^n$ using the average gradient information $\nabla f_{i,t}(\mathbf{x}_t)$ obtained on a relatively small t-th batch of $b \leq n$ datapoints. The Stochastic Gradient Descent (SGD) procedure then becomes an extension of the Gradient Descent (GD) to stochastic optimization of the loss function f_i as follows:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t \nabla f_{i,t}(\mathbf{x}_t), \tag{3.2}$$

where η_t is the learning rate. If we consider the second–order information we have

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta_t {\mathbf{H}_{i,t}}^{-1} \nabla f_{i,t}(\mathbf{x}_t). \tag{3.3}$$

We note that theoretical analysis for the SGD algorithm is beyond the scope of this thesis.

Algorithm 5: RACDM

```
  Input: $\eta_0$, $\mathbf{g} \in \mathbb{R}^R$, $\mathbf{a}_0 \in \mathbb{R}^R_+$.
1 $t = 1, \eta_t = \eta_0$, $\mathbf{a}_t = (\mathbf{a}_0 - \eta_0 \mathbf{g})_+$
2 do
3     Compute $\nabla f_i(\mathbf{a}_t)$
4     $\eta_{t+1} = \frac{1}{2}\eta_t$
5     $\mathbf{a}_{t+1} = (\mathbf{a}_t - \eta_{t+1}\nabla f_i(\mathbf{a}_t))_+$
6     $t = t + 1$
7 while $\mathbf{g}^T \nabla f_i(\mathbf{a}_t) < 0$
8 return $\eta_t$, $\mathbf{a}_t$.
```

3.2 First–Order Methods

First–order methods are popular due to their simplicity and optimal complexity.

We start with a method proposed in [10, p. 223]. The authors propose the step–size $\eta = \frac{1}{\|\nabla f_i\|\sqrt{k}}$, where $k \in \mathbb{N}_N$ [1]. We will refer to this method as "SimpleGrad".

We continue with a modified version of RACDM [11], where we ignore step 3 of the algorithm. We start from an initial value $\eta = \eta_0$, we compute the initial gradient $\mathbf{g} = \nabla f_i(\mathbf{a}_0)$, and then we perform a gradient step

$$a_{t+1} = (a_t - \eta_{t+1}\nabla f_i(\mathbf{a}_t))_+ \,. \tag{3.4}$$

We compute the gradient at the new point, $\nabla f_i(\mathbf{a}_{t+1})$. If

$$\nabla f_i(\mathbf{a}_0)^T \nabla f_i(\mathbf{a}_{t+1}) < 0, \tag{3.5}$$

then we set $\eta_{t+1} = \frac{\eta_t}{2}$ and repeat the procedure until relation (3.5) does not hold true. In Algorithm 5 we present our modified version.

The last first–order method is called Armijo line–search technique [12], denoted as "ARMIJO–LS". It was suggested in [13]. More specifically, starting from an initial value η_0, we are looking for a step–size η_t such that

$$f_i(\mathbf{a}_t) > f_i(\mathbf{a}_0) - \gamma\eta_t\|\mathbf{g}\|^2 \tag{3.6}$$

where $\gamma > 0$ is a hyper–parameter. If relation (3.6) does not hold true, then we backtrack by a constant factor $\delta \in (0, 1)$, that is, we set $\eta_{t+1} = \delta\,\eta_t$, until the line–search succeeds. In Algorithm 6 we present our version.

3.3 Second–Order Method

Incorporating the second–order curvature information has been shown to improve convergence [14]. However, one of the major drawbacks of second–order methods is their need for high computational and memory resources.

We can use the second–order information by computing the Hessian matrix $\mathbf{H}$. In [15],

[1] To be exact, k =ao_iter, where ao_iter denotes the iterations of AO and will be discussed in more detail later.

Algorithm 6: ArmijoLS

```
   Input: $\eta_0$, $\mathbf{g} \in \mathbb{R}^R$, $\mathbf{a}_0 \in \mathbb{R}^R_+$.
1  $t = 1, \mathbf{a}_t = \mathbf{a}_0$, $\eta_t = \eta_0$, $\gamma > 0$, $\delta \in (0, 1)$
2  Compute $f_i(\mathbf{a}_0)$
3  do
4      $\eta_{t+1} = \delta\eta_t$
5      $\mathbf{a}_{t+1} = (\mathbf{a}_t - \eta_{t+1}\mathbf{g})_+$
6      Compute $f_i(\mathbf{a}_{t+1})$
7      $t = t + 1$
8  while $f_i(\mathbf{a}_t) > f_i(\mathbf{a}_0) - \gamma\eta_t\|\mathbf{g}\|^2$
9  return $\eta_t$, $\mathbf{a}_t$.
```

we proposed a new method, called "1 over L", where we use the step–size $\eta = \frac{1}{L}$. L is the maximum eigenvalue of the local estimation of the Hessian matrix and is computed similarly as we described in section 2.3. We note that $\mathbf{H}$ is used only for the determination of L. A very interesting topic is the development of algorithms that fully exploit the Hessian matrix.

We may now proceed to the next section, where we present the accelerated stochastic gradient for NMLSME of section 2.3. We use the above methods to select the most efficient step–size.

3.4 Accelerated stochastic gradient for NMLSME

We solve problem (2.12) via the stochastic variant of the accelerated (Nesterov–type) gradient algorithm which appears in Algorithm 10.

During each iteration of the while–loop, we use a subset of the available entries of matrix $\mathbf{X}$. More specifically, at iteration l, we define a set of indices $\widehat{\Omega}_l \subset \Omega$ and a matrix $\widehat{\mathbf{M}}_l$, of the same size as $\mathbf{M}$, as

$$\widehat{\mathbf{M}}_l(i,j) = \begin{cases} 1, & \text{if } (i,j) \in \widehat{\Omega}_l, \\ 0, & \text{otherwise.} \end{cases} \tag{3.7}$$

We create set $\widehat{\Omega}_l$ randomly. We define the block–size $B_l := |\widehat{\Omega}_l|$ and select $c := \frac{B_l}{|\Omega|} < 1$. For row p of matrix $\mathbf{X}$, for $p = 1, \dots, P$, we sample, uniformly at random, a blocksize $B_{l,p} := \lfloor c\|\mathbf{M}(p,:)\|_0 \rfloor$ that contains the nonzero elements of $\mathbf{X}(p,:)$. We note that if $B_{l,p} = 0$, then we skip the p-th row.

We perform an accelerated gradient step using only the elements of $\mathbf{X}$ whose indices appear in $\widehat{\Omega}_l$. Thus, our cost function becomes $f_{\widehat{\Omega}_l}$ with gradient and Hessian similar to those in (2.13) and (2.14), with the only difference being that $\mathbf{M}$ is replaced by $\widehat{\mathbf{M}}_l$.

We find it convenient to compute the gradient and update the matrix variable in a row–wise fashion, using one of the above step–sizes.

More specifically, in the case where we use the "1 over L" method, we make use of the local estimation of the Hessian matrix of $f_{\widehat{\Omega}_l}$. We denote it as $\mathbf{H}_{l,p}$, and is computed with respect to the p-th row of $\mathbf{A}$. Quantity L_p is the largest eigenvalue of $\mathbf{H}_{l,p}$ and in that case, step–size $\eta_l(p)$ is equal to $\frac{1}{L_p}$. If we use the "SimpleGrad" method, the step–size is defined

Algorithm 7: RACDM for NMLSME

```
   Input: $\eta_0$, $\mathbf{g} \in \mathbb{R}^R$, $\mathbf{a}_0 \in \mathbb{R}_+^R$, $\mathbf{x}, \mathbf{m} \in \mathbb{R}_+^Q$, $\mathbf{B} \in \mathbb{R}_+^{Q\times R}$, $\lambda$.
 1 $t = 1, \eta_t = \eta_0$, $\mathbf{a}_t = (\mathbf{a}_0 - \eta_0 \mathbf{g})_+$
 2 do
 3     $\mathbf{w} = -(\mathbf{m} \circledast \mathbf{x})\mathbf{B}$
 4     $\mathbf{z} = \left(\mathbf{m} \circledast \left(\mathbf{a}_t \mathbf{B}^T\right)\right)\mathbf{B}$
 5     $\nabla f_{\widehat{\Omega}_l}(\mathbf{a}_t) = \mathbf{w} + \mathbf{z} + \lambda \mathbf{a}_t$
 6     $\eta_{t+1} = \frac{1}{2}\eta_t$
 7     $\mathbf{a}_{t+1} = (\mathbf{a}_t - \eta_{t+1}\nabla f_{\widehat{\Omega}_l}(\mathbf{a}_t))_+$
 8     $t = t + 1$
 9 while $\mathbf{g}^T \nabla f_{\widehat{\Omega}_l}(\mathbf{a}_t) < 0$
10 return $\mathbf{a}_t$, $\eta_t$.
```

Algorithm 8: ArmijoLS for NMLSME

```
   Input: $\eta_0$, $\mathbf{g} \in \mathbb{R}^R$, $\mathbf{a}_0 \in \mathbb{R}_+^R$, $\mathbf{x}, \mathbf{m} \in \mathbb{R}_+^Q$, $\mathbf{B} \in \mathbb{R}_+^{Q\times R}$, $\lambda$.
 1 $t = 1, \mathbf{a}_t = \mathbf{a}_0$, $\eta_t = \eta_0$, $\gamma = 0.1$, $\delta = 0.5$
 2 Compute $f_{\widehat{\Omega}_l}(\mathbf{a}_0)$ using $\mathbf{m}$, $\mathbf{x}$ and $\mathbf{B}$ according to (2.11)
 3 do
 4     $\eta_{t+1} = \delta \eta_t$
 5     $\mathbf{a}_{t+1} = (\mathbf{a}_t - \eta_{t+1}\mathbf{g})_+$
 6     Compute $f_{\widehat{\Omega}_l}(\mathbf{a}_{t+1})$ using $\mathbf{m}$, $\mathbf{x}$ and $\mathbf{B}$ according to (2.11)
 7     $t = t + 1$
 8 while $f_{\widehat{\Omega}_l}(\mathbf{a}_t) > f_{\widehat{\Omega}_l}(\mathbf{a}_0) - \gamma\eta_t\|\mathbf{g}\|^2$
 9 return $\mathbf{a}_t$, $\eta_t$.
```

as $\eta_l(p) = \frac{1}{\|\nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l(p,:))\|\sqrt{k}}$. Finally, in both "RACDM" and "ARMIJO–LS", Algorithms 7 and 8, we initialize step–size $\eta_0(p)$ using the quantity $\frac{1}{L_p}$, instead of a constant variable as the original authors propose. This adds an extra computational cost but we observed that it performs sufficiently good.

For the proposed line–search method "ARMIJO-LS", we used $\gamma = 0.1$ and $\delta = 0.5$, as we show in line 1 of Algorithm 8.

In order to choose the appropriate method, we use the variable

$$\mathit{opt} = \{1\text{:“1 over L”}, 2\text{:“SimpleGrad”}, 3\text{:“RACDM”}, 4\text{:“ARMIJO-LS”}\}.$$

After finding the desirable step–size, we update the row of the matrix $\mathbf{A}$ as shown in Algorithm 9. We can choose to perform an accelerated gradient step as shown in line 18, using either the constant step scheme II of[5, p. 80] (lines 12-13) or the constant step scheme III of [5, p. 81] (line 15) or we may not perform an accelerated gradient step.

3.5 Complexities

The most demanding computations required for the gradient $f_{\widehat{\Omega}_l}$ are the computation of $\mathbf{W}_l(p,:)$ (requires $O(B_{l,p}R)$ arithmetic operations per row or $O(B_lR)$ in total) and the computation of $\mathbf{Z}_l(p,:)$ (requires $O(B_{l,p}R)$ arithmetic operations per row or $O(B_lR)$ in total). After the computation of these matrices, $\nabla f_{\widehat{\Omega}_l}$ requires $O(PR)$ arithmetic operations.

Each of the presented step–sizes has a different complexity. "SimpleGrad" requires

Algorithm 9: Step size selection algorithm (choose_stepsize)

```
Input: opt, $\mathbf{g} \in \mathbb{R}^R$, $\mathbf{a}_0, \mathbf{y} \in \mathbb{R}_+^R$, $\mathbf{x}, \mathbf{m} \in \mathbb{R}_+^Q$, $\mathbf{B} \in \mathbb{R}_+^{Q \times R}$, $k$, $\lambda$, $\eta_0$.
1  if (opt == 1) then
2      $\mathbf{H} = \mathbf{B}^T \mathrm{diag}(\mathbf{m}) \mathbf{B} + \lambda \mathbf{I}_R$
3      $L = \max(\mathrm{eig}(\mathbf{H}))$
4      $\eta_{new} = \frac{1}{L}$
5      $\mathbf{a}_{new} = (\mathbf{y} - \eta \mathbf{g})_+$
6  else if (opt == 2) then
7      $\eta_{new} = \frac{1}{\|\mathbf{g}\| \sqrt{k}}$
8      $\mathbf{a}_{new} = (\mathbf{y} - \eta \mathbf{g})_+$
9  else if opt == 3 then
10     $(\mathbf{a}_{new}, \eta_{new}) = RACDM(\eta_0, \mathbf{g}, \mathbf{a}_0, \mathbf{x}, \mathbf{m}, \mathbf{B}, \lambda)$
11 else if opt == 4 then
12     $(\mathbf{a}_{new}, \eta_{new}) = ArmijoLS(\eta_0, \mathbf{g}, \mathbf{a}_0, \mathbf{x}, \mathbf{m}, \mathbf{B}, \lambda)$
13 return $\mathbf{a}_{new}, \eta_{new}$.
```

only the computation of the above gradient $f_{\widehat{\Omega}_l}$. "1 over L" has an additional cost for the computation of L_p, via the power method, which requires $O(R^2)$ arithmetic operations (in total, $O(PR^2)$) per iteration. In both "RACDM" and "ARMIJO–LS" we use L_p only for the step–size initialization during the first iteration. "RACDM" has an additional complexity, since it requires the computation of the gradient iteratively, with a cost of $O(B_{l,p}R)$ arithmetic operations per row. On the other side, "ARMIJO–LS" requires the computation of the cost function $f_{\widehat{\Omega}_l}$ iteratively, demanding in total $O(B_l)$ operations.

The computation of each of the matrices $\nabla f_{\widehat{\Omega}_l}$, $\mathbf{A}_{l+1}$ and $\mathbf{Y}_{l+1}$ requires $O(PR)$ arithmetic operations.

Finally, for notational convenience, we denote Algorithm 10 as

$$(\mathbf{A}_{\text{opt}}, \eta_{\text{new}}) = \text{S\_NMLSME}(\mathbf{X}, \mathbf{M}, \mathbf{B}, \mathbf{A}_*, \lambda, \eta_0, \text{opt}, \text{ao\_iter}).$$

Algorithm 10: Accelerated stochastic gradient for NMLSME (S_NMLSME)

```
   Input: $\mathbf{X}, \mathbf{M} \in \mathbb{R}_+^{P\times Q}$, $\mathbf{B} \in \mathbb{R}_+^{Q\times R}$, $\mathbf{A}_* \in \mathbb{R}_+^{P\times R}$, $\lambda$, $\eta_\mathbf{0} \in \mathbb{R}^\mathbf{P}$, opt, ao_iter.
1  $\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$
2  $l = 0$, $\alpha_{0,:} = 1$
3  while ($l$ <MAX_INNER) do
4      for $p = 1 \dots P$, in parallel do
5          $\widehat{\mathbf{M}}_l(p,:) = \text{sample}(\mathbf{M}(p,:))$
6          $\mathbf{W}_l(p,:) = -\left(\widehat{\mathbf{M}}_l(p,:) \circledast \mathbf{X}(p,:)\right)\mathbf{B}$
7          $\mathbf{Z}_l(p,:) = \left(\widehat{\mathbf{M}}_l(p,:) \circledast \left(\mathbf{Y}_l(p,:)\mathbf{B}^T\right)\right)\mathbf{B}$
8          $\nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l(p,:)) = \mathbf{W}_l(p,:) + \mathbf{Z}_l(p,:) + \lambda \mathbf{Y}_l(p,:)$
9          $(\mathbf{A}_{l+1}(p,:), \eta_{l+1}(p))$ = choose_stepsize(opt , $\nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l(p,:))$, $\mathbf{A}_l(p,:)$, $\mathbf{Y}_l(p,:)$,
           $\mathbf{X}(p,:)$, $\widehat{\mathbf{M}}_l(p,:)$, $\mathbf{B}$, ao_iter, $\lambda$, $\eta_l(p)$)
10         $q_p = \lambda\, \eta_{l,p}$
11         if *scheme II is used* then
12             choose $\alpha_{l+1,p} \in (0,1)$ from $\alpha_{l+1,p}^2 = (1-\alpha_{l+1,p})\alpha_{l,p}^2 + q_p\alpha_{l+1,p}$
13             $\beta_{l,p} = \frac{\alpha_{l,p}(1-\alpha_{l,p})}{\alpha_{l,p}^2 + \alpha_{l+1,p}}$
14         else if *scheme III is used* then
15             $\beta_{l,p} = \frac{1-\sqrt{q_p}}{1+\sqrt{q_p}}$
16         else
17             $\beta_{l,p} = 0$
18         $\mathbf{Y}_{l+1}(p,:) = \mathbf{A}_{l+1}(p,:) + \beta_{l,p}\left(\mathbf{A}_{l+1}(p,:) - \mathbf{A}_l(p,:)\right)$
19     $l = l + 1$
20 return $(\mathbf{Y}_l, \eta_l)$.
```

Chapter 4

Tensor Factorization and Completion

4.1 Introduction on Tensor Factorization and Completion

Definition 4.1 *The* ***outer product*** *of two vectors* $\mathbf{a} \in \mathbb{R}^I$ *and* $\mathbf{b} \in \mathbb{R}^J$ *is denoted as* $\mathbf{a} \circ \mathbf{b} \in \mathbb{R}^{I \times J}$ *and gives a rank–one matrix. Likewise, a* ***3–way outer product*** *of any three vectors,* $\mathbf{a} \in \mathbb{R}^I$, $\mathbf{b} \in \mathbb{R}^J$, $\mathbf{c} \in \mathbb{R}^K$ *is denoted as* $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c} \in \mathbb{R}^{I \times J \times K}$ *and gives a rank–one tensor with elements* $(\mathbf{a} \circ \mathbf{b} \circ \mathbf{c})(i,j,k) = a(i)b(j)c(k)$.

Definition 4.2 *The* ***order*** *of a tensor is the number of dimensions that it has. More precisely, scalars can be described as zeroth–order tensors, vectors as first–order tensors, matrices as second–order tensors, and any tensor having order n>2 (e.g.* $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$*) will be referred to as nth–order tensor.* In Fig. 4.1) we illustrate a third–order tensor.

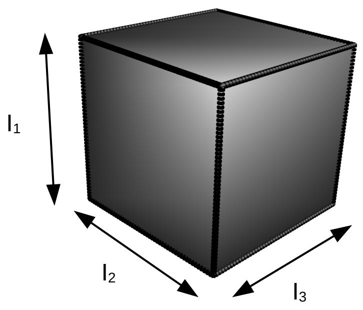


Figure 4.1: A third–order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$

Definition 4.3 *The* ***rank*** *of a tensor* $\mathcal{X}$ *is denoted as rank(*$\mathcal{X}$*) and defines the minimum number of rank–one tensors which are needed to produce* $\mathcal{X}$ *as their sum. For example, let* $\mathcal{X}$ *be a third–order tensor with rank(*$\mathcal{X}$*)* $= R$*, then*

$$\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r. \tag{4.1}$$

Definition 4.4 *In general, we can extract lower–order tensors from a nth–order tensor. In our case, from a third–order tensor, we can extract a first and second–order one (vectors and matrices correspondingly). More precisely, if we fix all but one indices, a* ***fiber*** *is created, otherwise, if we fix all but two indices, we create a* ***slice****. From a third–order tensor* $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$*, fibers are given as* $\mathbf{x}_{:jk}$*,* $\mathbf{x}_{i:k}$ *and* $\mathbf{x}_{ij:}$*, and slices are given as* $\mathbf{X}_{::k}$*,* $\mathbf{X}_{:j:}$ *and* $\mathbf{X}_{i::}$*.*

Definition 4.5 *The **Mode–n Matricization** of $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ is denoted as*

$$\mathbf{X}_{(1)} \in \mathbb{R}^{I_1 \times I_2 I_3}, \mathbf{X}_{(2)} \in \mathbb{R}^{I_2 \times I_1 I_3}, \mathbf{X}_{(3)} \in \mathbb{R}^{I_3 \times I_1 I_2}, \tag{4.2}$$

and defines the operation that reorders a tensor into a matrix, by turning the mode–n fibers (if we fix all but one indices) of tensor $\boldsymbol{\mathcal{X}}$ into the columns of matrix $\mathbf{X}_{(n)}$.

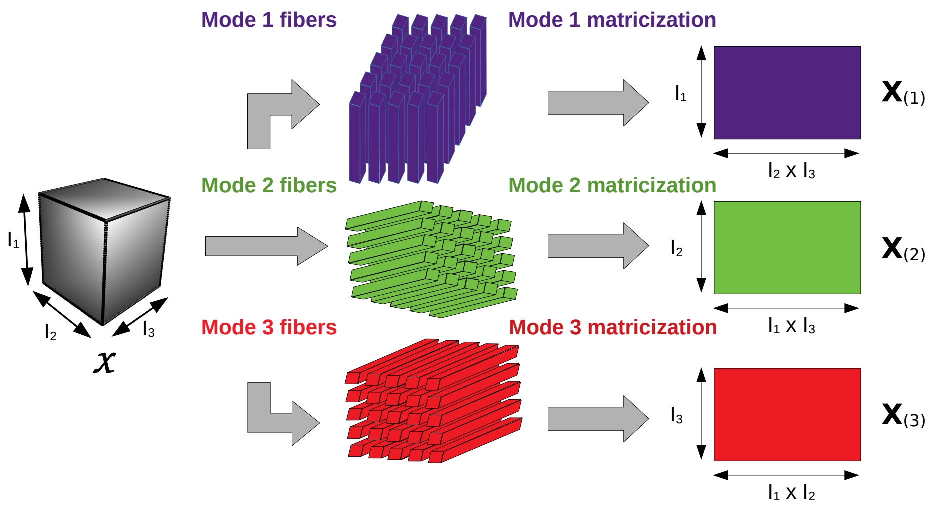


Figure 4.2: Mode–n Matricization of $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$

Definition 4.6 Finally, *the **Frobenius Norm** of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_1 \times I_3}$ is defined as*

$$||\mathcal{X}||_F = \sqrt{\sum_{i=1}^{I_1} \sum_{j=1}^{I_2} \sum_{k=1}^{I_3} \mathcal{X}(i,j,k)^2} \quad . \tag{4.3}$$

4.2 The PARAFAC Model

Let an N-mode tensor $\mathcal{X}^o \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ admit a PARAFAC (or CP Decomposition) of the form

$$\mathcal{X}^o = \sum_{r=1}^{R} \mathbf{u}_r^{(1)} \circ \mathbf{u}_r^{(2)} \circ \cdots \circ \mathbf{u}_r^{(N)} = [\![\mathbf{U}^{o(1)}, \mathbf{U}^{o(2)}, \ldots, \mathbf{U}^{o(N)}]\!], \tag{4.4}$$

where $\mathbf{U}^{o(i)} = [\mathbf{u}_1^{o(i)} \ldots \mathbf{u}_R^{o(i)}] \in \mathbb{R}^{I_i \times R}$, for $i = 1, \ldots, N$.

We observe the noisy tensor $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}}^o + \boldsymbol{\mathcal{E}}$, where $\boldsymbol{\mathcal{E}}$ is the additive noise. Estimates of $\mathbf{U}^{o(i)}$ can be obtained by computing matrices $\mathbf{U}^{(i)} \in \mathbb{R}^{I_i \times R}$, for $i \in \mathbb{N}_N$, that solve the optimization problem

$$\min_{\{\mathbf{U}^{(i)} \in \mathbb{U}^{(i)}\}_{i=1}^{N}} f_{\mathcal{X}}\left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}\right),$$

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

$$f_{\mathcal{X}}\left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}\right) = \frac{1}{2}\left\|\mathcal{X} - [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]\right\|_F^2. \tag{4.5}$$

This problem is nonconvex and, thus, difficult to solve, in general.

If $\mathcal{Y} = [\![\mathbf{U}^{(1)}, \dots, \mathbf{U}^{(N)}]\!]$, then its i-th mode matrix unfolding is given by [3]

$$\mathbf{Y}_{(i)} = \mathbf{U}^{(i)} \left(\mathbf{U}^{(N)} \odot \cdots \odot \mathbf{U}^{(i+1)} \odot \mathbf{U}^{(i-1)} \odot \cdots \odot \mathbf{U}^{(1)}\right)^T. \tag{4.6}$$

If we define the matrix

$$\mathbf{K}^{(i)} := \left(\mathbf{U}^{(N)} \odot \cdots \odot \mathbf{U}^{(i+1)} \odot \mathbf{U}^{(i-1)} \odot \cdots \odot \mathbf{U}^{(1)}\right), \tag{4.7}$$

then relation (4.8) can be written as

$$\mathbf{Y}_{(i)} = \mathbf{U}^{(i)} \mathbf{K}^{(i)^T}. \tag{4.8}$$

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

$$f_{\mathcal{X}}\left(\mathbf{U}^{(1)}, \dots, \mathbf{U}^{(N)}\right) = \frac{1}{2}\left\|\mathbf{X}_{(i)} - \mathbf{U}^{(i)}\mathbf{K}^{(i)^T}\right\|_F^2, \quad i \in \mathbb{N}_N. \tag{4.9}$$

These expressions form the basis of the AO approach for tensor decomposition, i.e., for fixed matrix factors $\mathbf{U}^{(j)}$, with $j \neq i$, we update $\mathbf{U}^{(i)}$ by solving an MLS problem, and this process is repeated circularly until convergence.

4.3 Nonnegative Tensor Factorization (NTF)

In many applications, we are interested in tensor decompositions whose factors should comply with constraints emerging from underlying models or for interpretability reasons.

Let an N-mode tensor $\mathcal{X}^o \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ admit a PARAFAC (or CP Decomposition) of the form

$$\mathcal{X}^o = \sum_{r=1}^{R} \mathbf{u}_r^{(1)} \circ \mathbf{u}_r^{(2)} \circ \cdots \circ \mathbf{u}_r^{(N)} = [\![\mathbf{U}^{o(1)}, \mathbf{U}^{o(2)}, \dots, \mathbf{U}^{o(N)}]\!], \tag{4.10}$$

where $\mathbf{U}^{o(i)} \in \mathbb{U}^{(i)}$ a certain set constraint and $i = 1, \dots, N$. In the unconstrained case, $\mathbb{U}^{(i)} \in \mathbb{R}^{I_i \times R}$, while in the nonnegative case, $\mathbb{U}^{(i)} \in {\mathbb{R}_+}^{I_i \times R}$. We focus on the Nonnegative Tensor Factorization (NTF), where all factors are nonnegative. In the AO framework, we update each factor separately and in a circular manner, i.e., for $i \in \mathbb{N}_N$, we update factor $\mathbf{U}^{(i)} \in {\mathbb{R}_+}^{I_i \times R}$ with all the other factors being fixed. Let us assume that during the $(k+1)$-st outer iteration, we have computed $\mathbf{U}_{k+1}^{(1)}, \dots, \mathbf{U}_{k+1}^{(i-1)}, \mathbf{U}_k^{(i)}, \dots, \mathbf{U}_k^{(N)}$. In order to update nonnegative factor $\mathbf{U}_k^{(i)}$, we solve the problem

$$\mathbf{U}_{k+1}^{(i)} = \underset{\mathbf{U}^{(i)} \in \mathbb{R}_+^{(i)}}{\operatorname{argmin}} \left\|\mathbf{X}_{(i)} - \mathbf{U}^{(i)}\mathbf{K}_k^{(i)T}\right\|_F^2. \tag{4.11}$$

The update is given by

$$\mathbf{U}_{k+1}^{(i)} = \texttt{NMLS}(\mathbf{X}_{(i)}, \mathbf{K}_k^{(i)}, \mathbf{U}_k^{(i)}), \tag{4.12}$$

4.4 Nonnegative Tensor Completion (NTC)

In many real–world problems, we observe a small subset of the elements of tensor $\mathcal{X}$, indexed by $\Omega \subseteq \mathbb{N}_{I_1} \times \cdots \times \mathbb{N}_{I_N}$. Let $\mathcal{M}$ be a binary tensor with the same size as $\mathcal{X}$ whose elements are defined as

$$\mathcal{M}(i_1, i_2, \ldots, i_N) = \begin{cases} 1, & \text{if } (i_1, i_2, \ldots, i_N) \in \Omega, \\ 0, & \text{otherwise.} \end{cases} \tag{4.13}$$

The number of nonzero elements of $\mathcal{X}$ is equal to $\texttt{nnz} := |\Omega|$. The NTC problem can be expressed as

$$\min f_\Omega \left(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\right) + \frac{\lambda}{2} \sum_{i=1}^{N} \|\mathbf{U}^{(i)}\|_F^2, \tag{4.14}$$

where

$$f_\Omega \left(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\right) = \frac{1}{2} \left\| \mathcal{M} \circledast \left(\mathcal{X} - [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!]\right)\right\|_F^2. \tag{4.15}$$

If $\mathcal{Y} = [\![\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}]\!]$, then

$$f_\Omega \left(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\right) = \frac{1}{2} \left\|\mathbf{M}_{(i)} \circledast \left(\mathbf{X}_{(i)} - \mathbf{Y}_{(i)}\right)\right\|_F^2, \quad i \in \mathbb{N}_N, \tag{4.16}$$

where $\mathbf{M}_{(i)}$, $\mathbf{X}_{(i)}$, and $\mathbf{Y}_{(i)}$ are, respectively, the matrix unfoldings of $\mathcal{M}$, $\mathcal{X}$, and $\mathcal{Y}$ with respect to the i-th mode. Similarly to the NTF case, these expressions form the basis of the AO method for NTC. If we consider factor $\mathbf{U}^{(i)}$ as a variable, with all the other factors being fixed, then we can update $\mathbf{U}^{(i)}$ by solving the problem

$$\min_{\mathbf{U}^{(i)} \in \mathbb{U}^{(i)}} \left\|\mathbf{M}_{(i)} \circledast \left(\mathbf{X}_{(i)} - \mathbf{U}^{(i)}\mathbf{K}^{(i)T}\right)\right\|_F^2 + \frac{\lambda}{2} \|\mathbf{U}^{(i)}\|_F^2. \tag{4.17}$$

As in the NTF problem, we start from initial points $\mathbf{U}_0^{(i)}$, for $i = 1, \ldots, N$, and solve, in a circular manner, NMLSME problems, based on the previous estimates.

4.5 NTC via stochastic NMLSME

Instead of using the NMLSME, which was presented in section 2.3, we will use the S_NMLSME method of section 3.4 to solve the NTC problem, as we present in Algorithm 11.

4.6 Pseudosymmetric NTC via stochastic NMLSME

In some applications, we are interested in the factorization of nonnegative and supersymmetric [1] incomplete tensors. For example, let tensor $\boldsymbol{\mathcal{S}}$ be the triple product of one single

[1]A tensor is supersymmetric if it is invariant under any permutation of indices $i_1, i_2, \ldots, i_N$. For example, for any element of a supersymmetric third–order tensor it holds that $\boldsymbol{\mathcal{X}}(i,j,k)$=$\boldsymbol{\mathcal{X}}(i,k,j)$=$\boldsymbol{\mathcal{X}}(j,i,k)$=$\boldsymbol{\mathcal{X}}(j,k,i)$=$\boldsymbol{\mathcal{X}}(k,i,j)$=$\boldsymbol{\mathcal{X}}(k,j,i)$ (N! permutations in total).

Algorithm 11: AO algorithm for NTC via S_NMLSME

```
   Input: $\mathcal{X}$, $\Omega$, $\mathbf{U}_0^{(i)} \in \mathbb{U}^i$, $i = 1, \ldots, N, \lambda$, rank $R$, opt.
1  $k = 0$
2  Initialize $\eta_0^{(i)}$ according to variable opt
3  while (1) do
4      for $i = 1, 2, \ldots N$ do
5          $(\mathbf{U}_{k+1}^{(i)}, \eta_{k+1}^{(i)}) = \text{S\_NMLSME}\left(\mathbf{X}_{(i)}, \mathbf{M}_{(i)}, \mathbf{K}_k^{(i)}, \mathbf{U}_k^{(i)}, \lambda, \eta_k^{(i)}, \text{opt}, k\right)$
6      if (terminating condition is TRUE) then break; endif
7      $k = k + 1$
8  return $\mathbf{U}_k^{(i)}, i = 1, \ldots, N$.
```

factor $\mathbf{U} \in \mathbb{R}^{I \times R}$ such that

$$\boldsymbol{\mathcal{S}} = [\![\mathbf{U}, \mathbf{U}, \mathbf{U}]\!]. \tag{4.18}$$

Using a supersymmetric binary tensor $\boldsymbol{\mathcal{M}}$ of the same dimensions, we have

$$\widehat{\boldsymbol{\mathcal{S}}} = \boldsymbol{\mathcal{S}} \circledast \boldsymbol{\mathcal{M}} \tag{4.19}$$

One can ignore the symmetry of $\widehat{\boldsymbol{\mathcal{S}}}$ and use Algorithm 11. The resulting factors will be close but not equal. To get the same resulting factors, we followed a naive approach. Instead of updating all factors in a circular manner, we update only the first one, $\mathbf{U}^{(1)}$, and we force the remaining factors to be equal. We refer to this method as "Pseudosymmetric" AO NTC, and works as in Algorithm 12. We start from an initial point $\left\{\mathbf{U}_0^{(i)} \in \mathbb{R}_+{}^{I_i \times R}\right\}_{i=1}^N$ and solve, in a circular manner, MLSME problems via S_NMLSME method, based on the previous estimates.

Algorithm 12: Pseudosymmetric AO NTC via S_NMLSME

```
   Input: $\mathcal{X}$, $\Omega$, $\left\{\mathbf{U}_0^{(i)} \in \mathbb{R}_+{}^{I_i \times R}\right\}_{i=1}^N$, $\lambda$, rank $R$, opt.
1  $k = 0$
2  Initialize $\eta_0^{(i)}$ according to variable opt
3  while (1) do
4      $(\mathbf{U}_{k+1}^{(1)}, \eta_{k+1}^{(1)}) = \text{S\_NMLSME}\left(\mathbf{X}_{(1)}, \mathbf{M}_{(1)}, \mathbf{K}_k^{(1)}, \mathbf{U}_k^{(1)}, \lambda, \eta_k^{(1)}, \text{opt}, k\right)$
5      for $i = 2, \ldots N$ do
6          $\mathbf{U}_{k+1}^{(i)} = \mathbf{U}_{k+1}^{(1)}$
7      if (term_cond is TRUE) then break; endif
8      $k = k + 1$
9  return $\left\{\mathbf{U}_k^{(i)}\right\}_{i=1}^N$.
```

Chapter 5

Parallel Implementations

In this chapter we present a parallel scheme for the Tensor Completion. There exist other parallel implementations, like [7], but in this thesis, a multi–threaded approach is presented. This parallel scheme is implemented using the OpenMP API in a shared memory environment.

5.1 Multi–threaded Implementation using OpenMP

Multi–threading is a model of program execution that allows for multiple threads to be created within a process, executing independently (each one is distributed to its own CPU core) but concurrently sharing process resources.

OpenMP (MP stands for *multiprocessing*) is an API developed for shared–memory parallel programming. OpenMP is directive–based programming model designed as an extension of C and C++ [16]. We selected OpenMP rather than other interfaces, like POSIX threads (*Pthreads*), since OpenMP is of higher level, allowing us to parallelize tasks and assign them to threads easier.

5.2 Parallel implementation of NMLSME and S_NMLSME

As we have shown in Sections 2.3, 3.4, we update each row of factor $\mathbf{U}^{(i)}$ separately. Both NMLSME and S_NMLSME can be parallelized in the same manner, but in this thesis we will focus only on the parallel implementation of method S_NMLSME. Since each row can be updated separately and independently, it is assigned as a task to a corresponding available thread. Therefore, we can solve lines 6-18 of Algorithm 10 in parallel.

We note that that the complexity, and thus the execution time, varies from task to task, and depends on the way that the nonzeros are distributed. Therefore, we must consider the scheduling of the threads.

We have experimented with various scheduling types and we found out that the most suitable is the dynamic type. In a nutshell, this type of scheduling ensures that each thread executes a chunk of iterations and then requests another chunk until there are no more chunks available.

In Algorithm 13, we provide a high level algorithmic sketch of the proposed parallel and accelerated stochastic gradient for NMLSME.

Algorithm 13: Parallel accelerated stochastic gradient for NMC

```
   Input: $\mathbf{X}, \mathbf{M} \in \mathbb{R}_+^{P\times Q}$, $\mathbf{B} \in \mathbb{R}_+^{Q\times R}$, $\mathbf{A}_* \in \mathbb{R}_+^{P\times R}$, $\lambda, \eta_{\mathbf{0}} \in \mathbb{R}^{\mathbf{P}}$, opt, ao_iter
1  $\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$
2  $l = 0$
3  while (1) do
4      if ($l \geq$MAX_INNER) then
5          break
6      else
7          in parallel for $p = 1 \ldots P$ do
8              lines 6-18 of Algorithm 10
9          $l = l + 1$
10 return $\mathbf{A}_l$.
```

Chapter 6

Applications using Tensor Completion

6.1 Corrupted Image (Missing Values)

An interesting application is the estimation of missing data in images, following the RGB model, inspired by the works of [17] and [18].

6.1.1 Image Representations

In a nutshell, digital images can be represented using multiple color formats. We begin with black and white images, which have monochrome palettes, that only have some shades of gray, from black to white, both considered the most possible darker and lighter “grays”, respectively. For example, a 1-bit grayscale image requires only one bit per pixel. Each bit is either equal to 0 or 1, where 1 represents white pixels (light) and a value of 0 the black ones (dark). As the number of bits increases according to the exponential 2^n, more different shades of gray are displayed (more quantization levels). Usually, the maximum number of grays in ordinary monochrome systems is $2^8 = 256$, resulting in a 256-value palette. Grayscale images use only one single color plane and can be represented as a matrix $\mathbf{I} \in \mathbb{R}^{h \times w}$, where h, w are the image's height and width respectively.

On the other hand, color images use three color planes, each with n number of possible levels by component, raised to a power of 3 ($n \times n \times n = n^3$). One of the most popular image representation models is the RGB (Red – Green – Blue) model. An RGB image is a three-dimensional byte array that explicitly stores a color value for each pixel. RGB image arrays are made up of width, height, and three channels of color information. The color information is stored in three sections of a third dimension of the image. These sections are known as color channels, color bands, or color layers. One channel represents the amount of red in the image (the red channel), one channel represents the amount of green in the image (the green channel), and one channel represents the amount of blue in the image (the blue channel). Therefore, colored images can be represented as a tensor $\mathcal{I} \in \mathbb{R}^{h \times w \times 3}$, where the first two dimensions h, w are the image's height and width respectively, and the third one holds the number of channels. In Fig. 6.1 we illustrate a simplified tensor representation of an image following the RGB model.

As an example, we use an image of dimensions 1063×1599 [1], as shown in Fig. 6.2a. This 8-bit image (each pixel is represented by 8-bits or 1 byte) can be seen as a tensor

[1]Image can be found at https://images.freeimages.com/images/large-previews/7bb/building-1222550.jpg

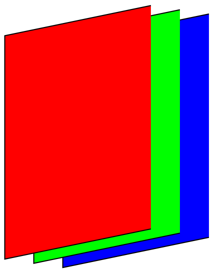


Figure 6.1: Separate Red, Green, and Blue image layers

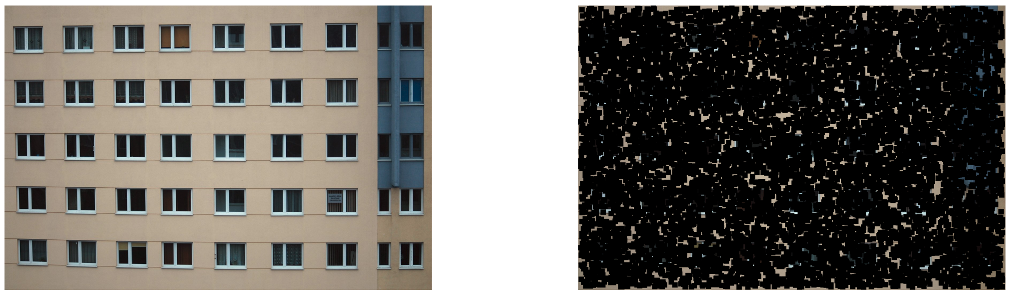
$\boldsymbol{\mathcal{I}} \in \mathbb{Z}_{+}^{1063\times1599\times3}$, with values in [0,1,. . . ,254, 255].

6.1.2 Missing values

To create an image with missing elements, we first need to generate a matrix $\mathbf{M}$ of the same first two dimensions of the image (height and width), with values 0 ("missing" value) or 1 ("known" value), which we will refer to as the mask. We begin by assigning all values of $\mathbf{M} \in \mathbb{N}^{h\times w}$ to 1. More specifically, we select in a random manner blocks of size 20×20 from $\mathbf{M}$, with all values assigned to 0, and we repeat this process until we obtain the desired sparsity level (e.g. 90% "0" / 10% "1" = 90% sparsity). Then we apply the mask $\mathbf{M}$ for each channel as follows

$$\boldsymbol{\mathcal{X}}(:,:,i) = \boldsymbol{\mathcal{I}}(:,:,i) \circledast \mathbf{M}, \text{ where } i = 1, 2, 3.$$

The resulted 90% sparse tensor $\boldsymbol{\mathcal{X}}$ is illustrated in Fig. 6.2b. We note that from now on we make the assumption that our incomplete image is represented by the tensor $\boldsymbol{\mathcal{X}}$ in $\mathbb{R}_{+}^{1063\times1599\times3}$ and not in $\mathbb{Z}_{+}^{1063\times1599\times3}$ as is the original tensor.



(a) Original (b) Corrupted (90% sparse)

Figure 6.2: Tensor Completion on images

In order to restore our incomplete image, we use the Stochastic NTC method. In the next chapter, we present our methodology in image restoration and we illustrate the resulting restored image.

6.2 Natural Language Processing

Another interesting application is Natural Language Processing (NLP). Most tasks in natural language processing involve looking at words and finding similarities and dissimilarities between them. There are many word representations in NLP, but we will focus only on the "Distributional Representation".

6.2.1 Distributional Representation

This type of representation is based on the distributional hypothesis [19], which states that words in similar contexts have similar meanings. This has given rise to many word representation methods in the NLP literature, the vast majority of whom can be described in terms of a word-context matrix $\mathbf{M}$ in which each row i corresponds to a word, each column j to a context in which the word appeared, and each matrix entry $\mathbf{M}(i, j)$ corresponds to some association measure between the word and the context. Rows of $\mathbf{M}$ represent words in the vocabulary and columns represent contexts. The context could be sliding windows over the training sentences or even documents.

Many state-of-the-art word embedding techniques involve the factorization of a co-occurrence-based matrix [20]. More recently, authors [21] have extended this approach by studying word embedding techniques that involve the factorization of co-occurrence-based tensors. Inspired by this work, we aimed to examine if our proposed Stochastic tensor decomposition method performs well in NLP tasks.

Before we proceed into details, some preliminaries must be presented.

6.2.2 Word Embedding

In general, word embedding is a term used for the representation of words for text analysis, typically in the form of a real-valued dense vector. This vector encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning. These representations referred to as "neural embeddings" or "word embedding", have been shown to perform well in a wide range of NLP tasks. In many works, authors propose to represent words as dense vectors, derived using various training methods inspired from neural-network language modeling [22].

6.2.3 n-gram

An n-gram is a contiguous sequence of n items from a given sample of text or speech corpus. The items can be either phonemes, syllables, letters, words, numbers, or base pairs according to the application. They are collected from a text or speech corpus. When $n = 1$, the 1-gram is referred to as a "unigram", for $n = 2$ is a "bigram", for size 3 is a "trigram". For $n > 3$ we have a "four-gram", a "five-gram", and so on [23].

6.2.3.1 Skip-gram

One of the most popular methods for learning representations of words is the skip-gram model. Skip-grams are a technique largely used in the field of speech processing, whereby n-grams are formed but in addition to allowing adjacent sequences of words, tokens are allowed to be "skipped" [24]. It predicts words within a certain range, called window, before and after the current word in the same sentence. A simple example of this is given below for a window of size 2.

The sentence

“Concorde makes emergency landing in Canada”

produces the following skip-grams:

- Concorde : (Concorde, makes), (Concorde, emergency),
- makes : (makes, Concorde), (makes, emergency), (makes, landing),
- emergency: (emergency, Concorde), (emergency, makes), (emergency, landing), (emergency, in),
- landing : (landing, makes), (landing, emergency), (landing, in), (landing, Canada),
- in : (in, emergency), (in, landing), (in, Canada),
- Canada : (Canada, landing), (Canada, in).

6.2.3.2 Word2Vec

In [22], Mikolov et. al. used the Skip-Gram model with Negative-Sampling (SGNS) training method which is both efficient to train and provides state-of-the-art results on various linguistic tasks. The proposed training method, called “Word2Vec” is very popular and embeddings learned through this method have proven to be successful on a variety of downstream natural language processing tasks. The training objective follows the distributional hypothesis, trying to maximize the dot-product between the vectors of frequently occurring word-context pairs, and minimize it for random word-context pairs. We will not focus on this method but we will use it to compare the resulting word embeddings.

In [20], the authors prove that, despite seeming like a local neural network, the SGNS method can be seen as an implicit weighted matrix factorization problem. Later on, we will discuss the relationship between word embeddings and matrix factorization.

6.2.4 Pointwise Mutual Information

Pointwise mutual information (PMI) is a useful property in NLP. It quantifies the likelihood of co-occurrence of two words [20]. It is defined as:

$$\mathrm{PMI}(w_1, w_2) = \log \frac{p(w_1, w_2)}{p(w_1)p(w_2)}, \tag{6.1}$$

where $p(w_1, w_2)$ is the probability that both words w_1 and w_2 occur inside a fixed window of L word in the corpus, and $p(w_1)$, $p(w_2)$ are their marginal probabilities. How do we compute the probabilities $p(w_1, w_2)$, $p(w_1)$, $p(w_2)$?

Let N the number of pairs of words observed in a corpus, $f(w) \leq N$ the number of times word w occurs, $f(w_1, w_2)$ the number of times words w_1, w_2 occur in the same window of size L. Then, the probabilities $p(w_1)$, $p(w_2)$, $p(w_1, w_2)$ are computed as follows

$$p(w_1) = \frac{f(w_1)}{N} = \frac{\sum_{j\in|V|} f(w_1, w_j)}{N}, \tag{6.2}$$

$$p(w_2) = \frac{f(w_2)}{N} = \frac{\sum_{i\in|V|} f(w_i, w_2)}{N}, \tag{6.3}$$

$$p(w_1, w_2) = \frac{f(w_1, w_2)}{N}. \tag{6.4}$$

As authors in [20] suggest, PMI is replaced with positive PMI (PPMI) metric, which is defined as:

$$\mathrm{PPMI}(w_1, w_2) := \max(0, \mathrm{PMI}(w_1, w_2)). \tag{6.5}$$

since negative PMI values can not be easily interpreted.

For an indexed vocabulary $V = \{w_1, w_2, \dots, w_{|V|}\}$, of size $|V|$, we can construct a PPMI matrix $\mathbf{M}$ with elements $\mathbf{M}(i, j) = PPMI(w_i, w_j)$.

PMI can be easily generalized in more than two variables. We consider PMIs with N variables defined as

$$\mathrm{PMI}(w_1, w_2, \dots, w_N) = \log \frac{p(w_1, w_2, \dots, w_N)}{p(w_1)p(w_2)\dots p(w_N)}, \tag{6.6}$$

where $p(w_1, w_2, \dots, w_N)$ is the probability that $w_1, w_2, \dots, w_N$ occur together in a given fixed-length context window in the corpus, regardless of their order. For $N = 3$, the probabilities $p(w_1)$, $p(w_2)$, $p(w_3)$, $p(w_1, w_2, w_3)$ are computed as in relations (6.2 - 6.4). Briefly we have

$$p(w_1) = \frac{\sum_{j,k\in|V|} f(w_1, w_j, w_k)}{N}, \tag{6.7}$$

$$p(w_2) = \frac{\sum_{i,k\in|V|} f(w_i, w_2, w_k)}{N}, \tag{6.8}$$

$$p(w_3) = \frac{\sum_{i,j\in|V|} f(w_i, w_j, w_3)}{N}, \tag{6.9}$$

$$p(w_1, w_2, w_3) = \frac{f(w_1, w_2, w_3)}{N}. \tag{6.10}$$

Therefore, for $N = 3$, we construct a PPMI tensor $\boldsymbol{\mathcal{M}}$ with elements $\boldsymbol{\mathcal{M}}(i, j, k) = PPMI(w_i, w_j, w_k)$.

In [20], they show that SGNS from [22] can be viewed as an implicit matrix factorization of the matrix $\mathbf{M}$. More specifically, it embeds both words and their contexts into a low-dimensional space $\mathbb{R}^d$, resulting in the word and context matrices $\mathbf{W} \in \mathbb{R}^{|V|\times R}$ and $\mathbf{C} \in \mathbb{R}^{|V|\times R}$ ($\mathbf{M} = \mathbf{W}\mathbf{C}^T$). We highlight that the factorization rank d is the dimension of the resulting vector representation. The rows of matrix $\mathbf{W}$ are typically used in NLP tasks, such as computing word similarities, while $\mathbf{C}$ is ignored. This can be generalized for $N > 2$. Just as the rank-R matrix decomposition is defined to be the product of two-factor matrices $\mathbf{M} = \mathbf{W}\mathbf{C}^T$, the PARAFAC rank-R tensor decomposition for a third-order tensor is defined to be the product of three-factor matrices. Since our PPMI tensor $\mathcal{M}$ is nonnegative and invariant under permutation, M is nonnegative and supersymmetric. As in [25] and [21], we will also consider symmetric CP decomposition of nonnegative tensors.

We factorize the PPMI tensor $\boldsymbol{\mathcal{M}}$ as the triple product of one single factor $\mathbf{U} \in \mathbb{R}^{|V| \times R}$ such that

$$\boldsymbol{\mathcal{M}} = [\![\mathbf{U}, \mathbf{U}, \mathbf{U}]\!],$$

where factor $\mathbf{U}$ is the resulted word embedding. Tensor $\boldsymbol{\mathcal{M}}$ is usually very sparse ($> 99\%$), nonnegative and supersymmetric. Therefore, we use the proposed pseudosymmetric NTC via stochastic NMLSME, of section 4.6, in order to obtain the desired word embedding. In the next chapter (section 7.6), we will present our results using a real-world dataset and we will compare our method with Word2Vec.

Chapter 7

Numerical Experiments

7.1 Rank estimation problem with NTC via stochastic NMLSME

7.1.1 Formulation of the problem

In real world problems where the datasets can be represented as tensors, the only available information is usually a list of nonzero entries and their respective indices. The factorization rank is not given and thus, we have to examine different values of ranks to fit our data. In order to select an appropriate factorization rank, we use the 5-Fold Cross-Validation process, as shown in Fig. 7.1. In more detail, for each one of the five different split train/test sets, we solve the Nonnegative Tensor Completion (NTC) problem using an arbitrary range of ranks. Finally, we select the factorization model with the rank that gives the lower values on the associate performance metrics. More specifically, we use the Root Mean Square Error (RMSE), namely

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{nnz_{test}} (\text{test value}_i - \text{estimated value}_i)^2}{nnz_{test}}}, \tag{7.1}$$

where

$$\text{estimated value}_i = \sum_{r=1}^{R} \left(\mathbf{U}^{(1)}_{\text{train}}(i_1, :) \circledast \mathbf{U}^{(2)}_{\text{train}}(i_2, :) \circledast \cdots \circledast \mathbf{U}^{(N)}_{\text{train}}(i_N, :) \right),$$

and the Relative Incomplete Factorization Error (RIFE)

$$\text{RIFE} = \frac{\| \boldsymbol{\mathcal{M}}_{test} \circledast (\boldsymbol{\mathcal{X}}_{test} - \boldsymbol{\mathcal{X}}_{est}) \|_F}{\| \boldsymbol{\mathcal{M}}_{test} \circledast \boldsymbol{\mathcal{X}}_{test} \|_F}, \tag{7.2}$$

where

$$\boldsymbol{\mathcal{X}}_{est} = [\![\mathbf{U}^{(1)}_{\text{train}}, \mathbf{U}^{(2)}_{\text{train}}, \ldots, \mathbf{U}^{(N)}_{\text{train}}]\!].$$

We also use the Relative Factorization Error (RFE)

$$\text{RFE} = \frac{\| \boldsymbol{\mathcal{X}}_{true} - \boldsymbol{\mathcal{X}}_{est} \|_F}{\| \boldsymbol{\mathcal{X}}_{true} \|_F}, \tag{7.3}$$

in the case where the whole dataset is known (original tensor has no missing elements).

In order to estimate the tensor's rank on both synthetic and real data, we use the Accelerated Stochastic Gradient for NTC, using the "1 over L" step–size method and

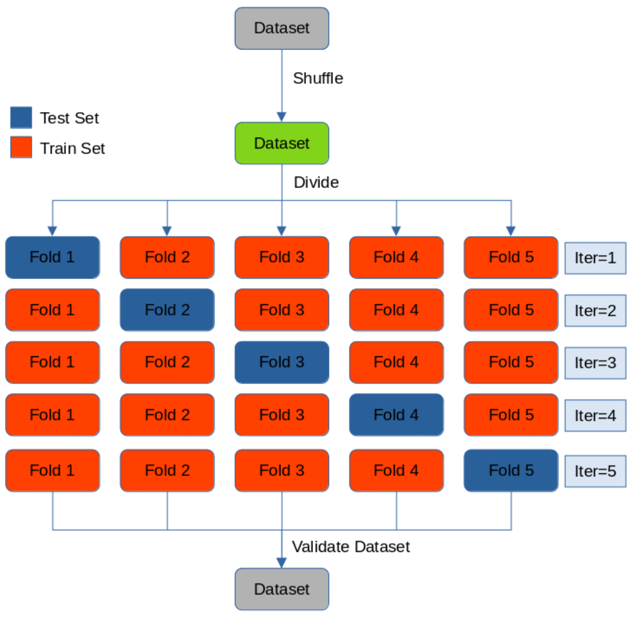


Figure 7.1: 5-Fold Cross Validation Process

with a fixed number of inner iterations and a varying blocksize $B^l = c|\Omega|$. In most of our experiments we use 1 inner iteration, starting from initial factors $\mathbf{U}^{o(i)} \in \mathbb{R}_+^{I_i \times R}$ for $i = 1, 2, 3$, with i.i.d. $\mathcal{U}[0, 1]$ elements.

7.1.2 Synthetic Noiseless Data (Rank 10, 20, 50)

We start with synthetic data. We construct three nonnegative tensors, one rank–10, one rank–20, and one rank–50 tensor, of the same size and all denoted as $\boldsymbol{\mathcal{X}}^o \in \mathbb{R}_+^{3000 \times 1700 \times 65}$. Both follow the CPD model and are constructed using true nonnegative factors. Then we apply a mask $\boldsymbol{\mathcal{M}}$ of the same size with values 0 or 1. The observed incomplete tensor is expressed as

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{M}} \circledast \boldsymbol{\mathcal{X}}^o$$

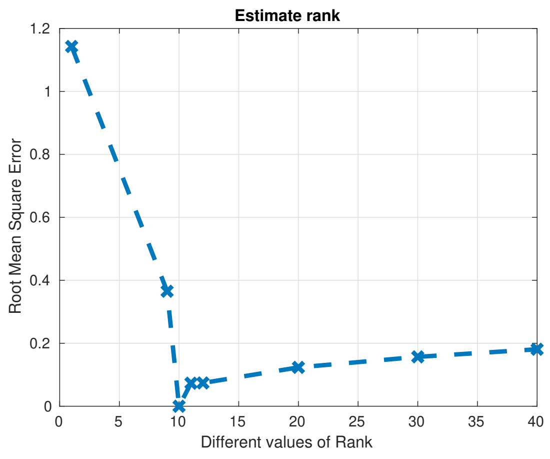
and has 5M nonzeros.

7.1.2.1 Synthetic Noiseless (Rank 10)

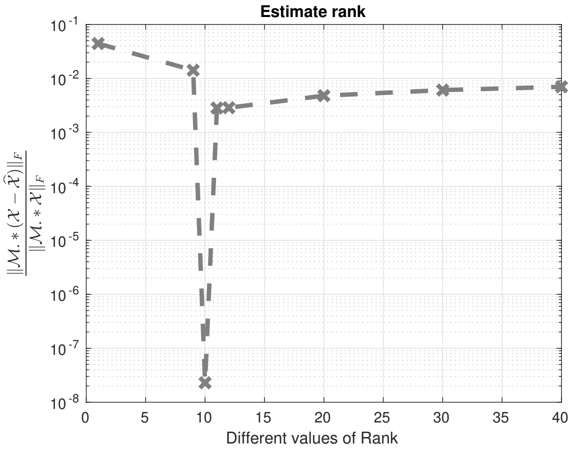
We set the number of epochs equal to 50. As epoch we denote the number of iterations required to access once all available tensor elements [1]. We set $c = 0.5$ and $\lambda = 10^{-3}$. We test for $R = 1, 9, 10, 11, 12, 20, 30, 40$. As performance metrics, we use RIFE, RMSE and RFE. In Fig. 7.2 we illustrate the results of the aforementioned model selection process.

We observe that $R = 10$ gives the lowest for all the metrics used on the test set, which is the true rank of the noiseless tensor.

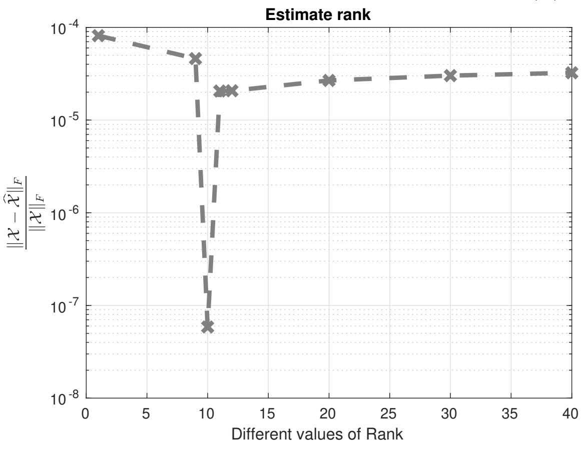
[1]For example, if the number of outer iterations is 10 and $c = 0.2$, then the number of epochs is equal to $10 * \frac{1}{0.2} = 50$.



(a) RMSE



(b) RIFE



(c) RFE

Figure 7.2: RMSE (a), RIFE (b) and RFE (c) vs Different values of rank for the synthetic (R10) noiseless dataset.

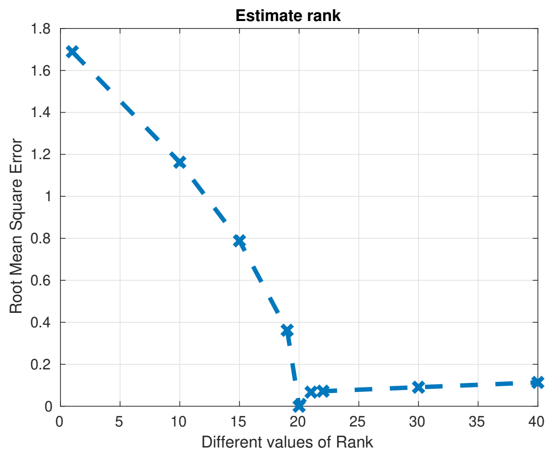
7.1.2.2 Synthetic Noiseless (Rank 20)

We set $c = 0.5$, $\lambda = 10^{-4}$ and the number of epochs equal to 200. We test for $R = 1, 10, 15, 19, 20, 21, 22, 30, 40$. As performance metrics, we use RIFE, RMSE and RFE. In Fig. 7.3 we illustrate the results of the aforementioned model selection process.

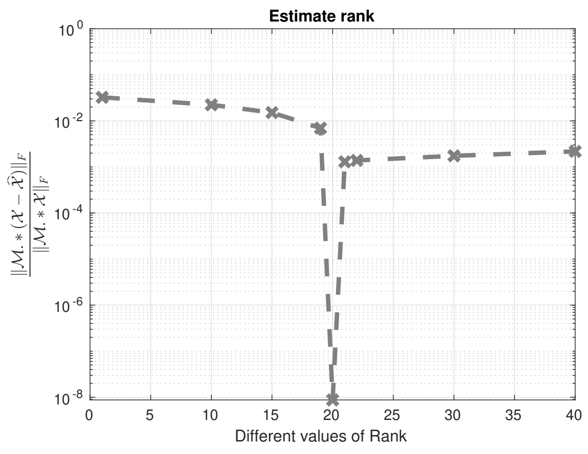
We observe that $R = 20$ gives the lowest for all the metrics used on the test set, which is the true rank of the noiseless tensor.

7.1.2.3 Synthetic Noiseless (Rank 50)

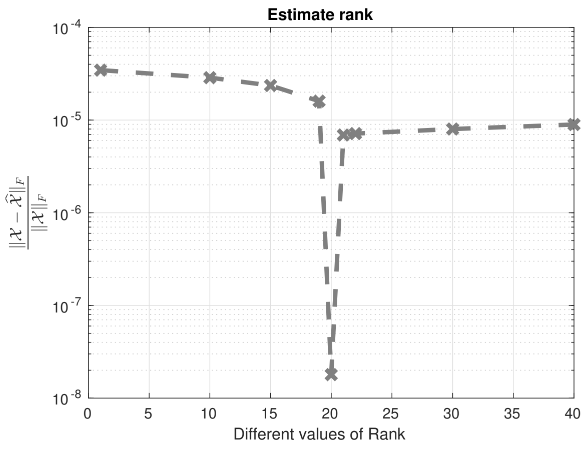
We set $c = 0.2$, number of epochs $= 100$ and $\lambda = 10^{-3}$. We test for various ranks $R = 1, 10, 40, 49, 50, 51, 52, 60$. As performance metrics, we use RIFE, RMSE and RFE. In Fig. 7.4 we illustrate the results of the aforementioned model selection process.



(a) RMSE



(b) RIFE



(c) RFE

Figure 7.3: RMSE (a), RIFE (b) and RFE (c) vs Different values of rank for the synthetic (R20) noiseless dataset.

7.1.3 Synthetic Noisy Data (Rank 10, 20, 50)

In the noisy case, we add noise $\boldsymbol{\mathcal{E}}$ on the complete known tensor $\boldsymbol{\mathcal{X}}^o$. The additive noise has i.i.d. elements $\boldsymbol{\mathcal{N}}(0, \sigma_N^2)$. The observed incomplete tensor $\boldsymbol{\mathcal{X}}$ is expressed as

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{M}} \circledast (\boldsymbol{\mathcal{X}}^o + \boldsymbol{\mathcal{E}}) .$$

We define the Signal-to-Noise ratio as

$$\text{SNR} := \frac{\|\boldsymbol{\mathcal{M}} \circledast \boldsymbol{\mathcal{X}}^o\|_F^2}{\|\boldsymbol{\mathcal{M}} \circledast \boldsymbol{\mathcal{E}}\|_F^2}. \tag{7.4}$$

We present our results for various noise levels 10 and 20 dB.

7.1.3.1 Synthetic Noisy 10dB (Rank 10)

We set $c = 0.2$, $\lambda = 10^{-3}$ and number of epochs equal to 100. We test for $R = 1, 9, 10, 11, 12, 20, 30, 40$. In Fig. 7.5 we illustrate the results of the aforementioned model selection process.

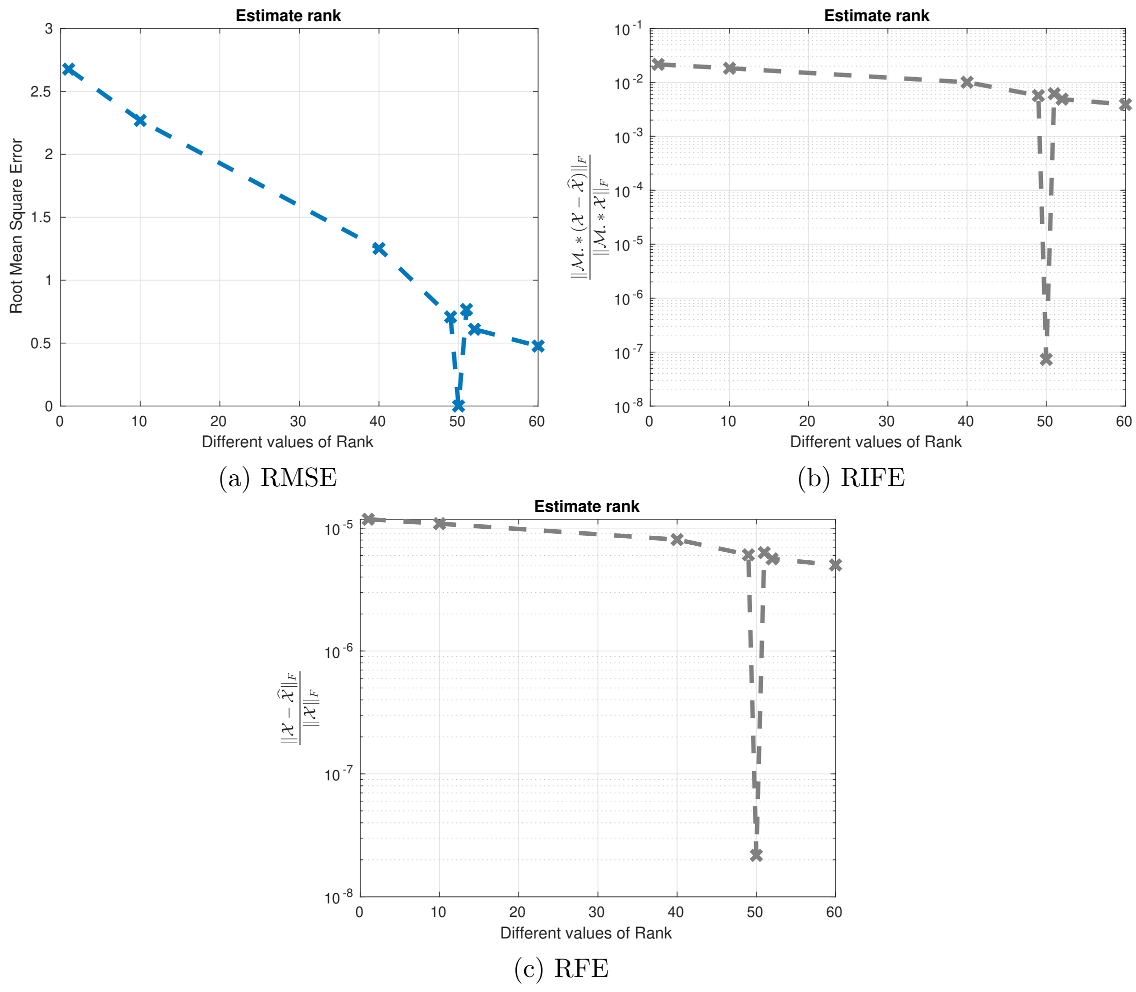


Figure 7.4: RMSE (a), RIFE (b) and RFE (c) vs Different values of rank for the synthetic (R50) noiseless dataset.

We observe that rank $R = 10$ gives the lowest values for all the metrics used on the test set, which is the true rank.

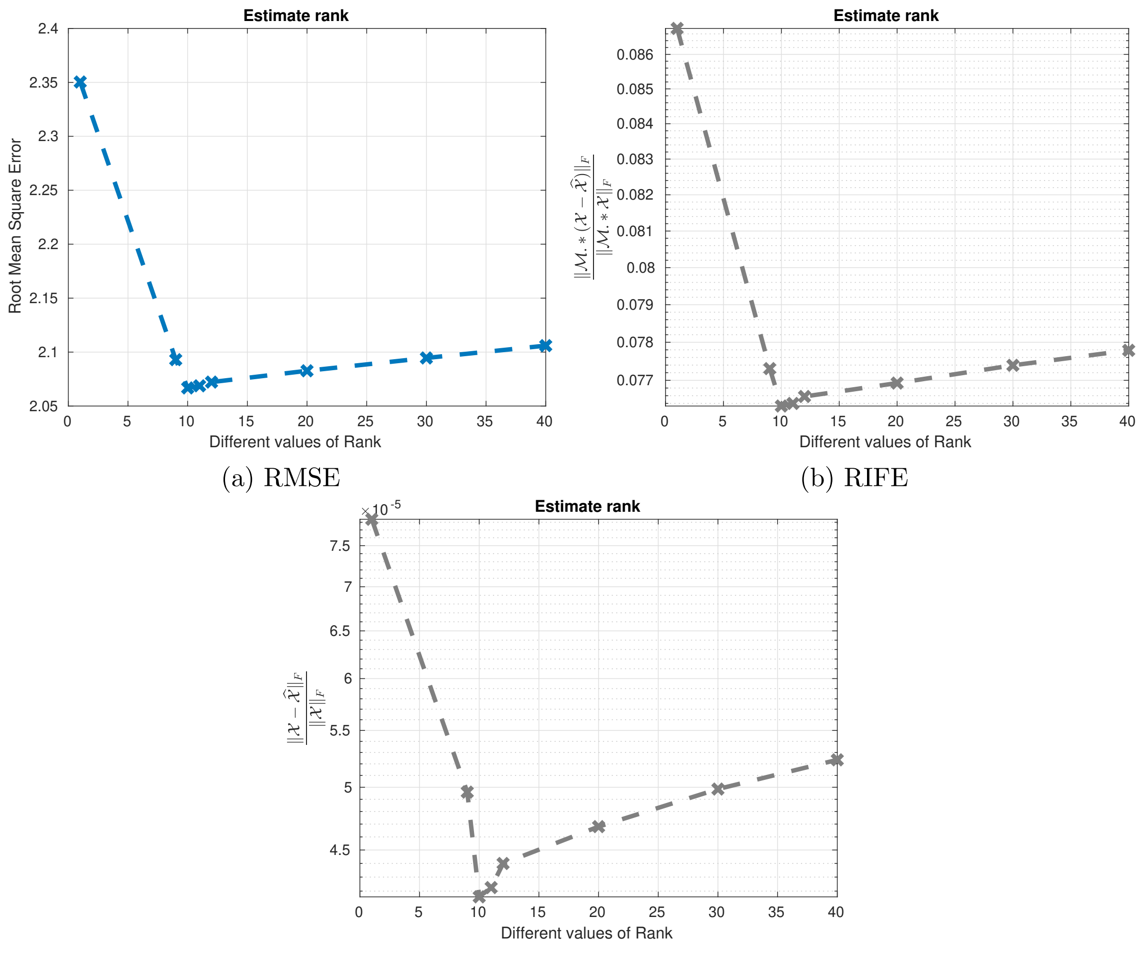
7.1.3.2 Synthetic Noisy 10dB (Rank 20)

For the rank–20 tensor, we set the number of epochs equal to 200, $c = 0.5$ and $\lambda = 10^{-3}$. We test for various ranks $R = 1, 10, 15, 19, 20, 21, 22, 30, 40$. As performance metrics, we use RIFE, RMSE and RFE. In Fig. 7.6 we illustrate the results of the aforementioned model selection process.

We observe that $R = 20$ gives the lowest for all the metrics used on the test set, which is the true rank of the noisy tensor.

7.1.3.3 Synthetic Noisy 10dB (Rank 50)

For the rank–50 tensor, we set the number of epochs equal to 100. We set $c = 0.5$ and $\lambda = 10^{-1}$. We test for various ranks $R = 1, 10, 40, 49, 50, 51, 52, 60$. As performance metrics, we use RIFE, RMSE and RFE. In Fig. 7.7 we illustrate the results of the aforementioned model selection process.



(c) RFE

Figure 7.5: RMSE (a), RIFE (b) and RFE (c) vs Different values of rank for the synthetic (R10) noisy (10 dB) dataset.

We observe that ranks $R = 1, 10, 50$ are suitable for this dataset.

7.1.3.4 Synthetic Noisy 20dB (Rank 10)

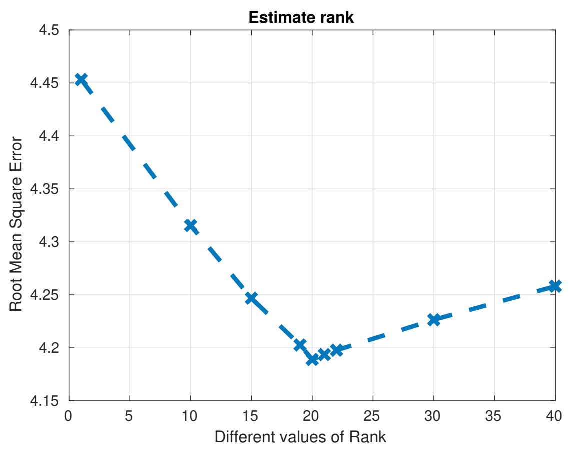
We continue with a 20dB noise level. For the rank–10 tensor we set the number of epochs to 100, $c = 0.5$ and $\lambda = 10^{-3}$. We test for $R = 1, 9, 10, 11, 12, 20, 30, 40$. In Fig. 7.8 we illustrate the results of the aforementioned model selection process.

We observe that in higher SNR the estimated rank $R = 10$ is the actual factorization rank.

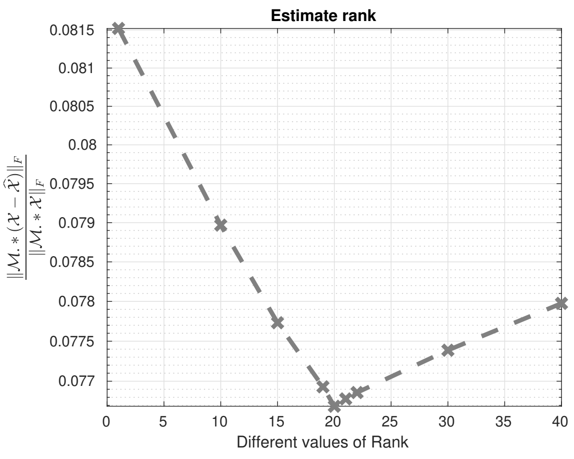
7.1.3.5 Synthetic Noisy 20dB (Rank 20)

For the rank–20 tensor, we set the number of epochs to 200, $c = 0.5$ and $\lambda = 10^{-4}$. We test for $R = 1, 10, 15, 19, 20, 21, 22, 30, 40$. In Fig. 7.9 we illustrate the results of the aforementioned model selection process.

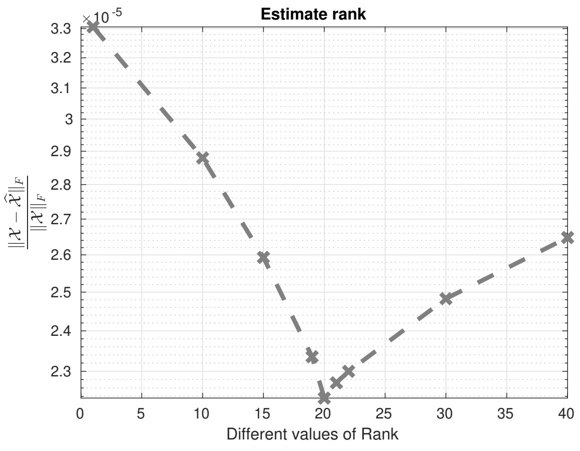
Again, we observe that in higher SNR the estimated rank $R = 20$ is the actual factorization rank.



(a) RMSE



(b) RIFE



(c) RFE

Figure 7.6: RMSE (a), RIFE (b) and RFE (c) vs Different values of rank for the synthetic (R20) noisy (10 dB) dataset.

7.1.3.6 Synthetic Noisy 20dB (Rank 50)

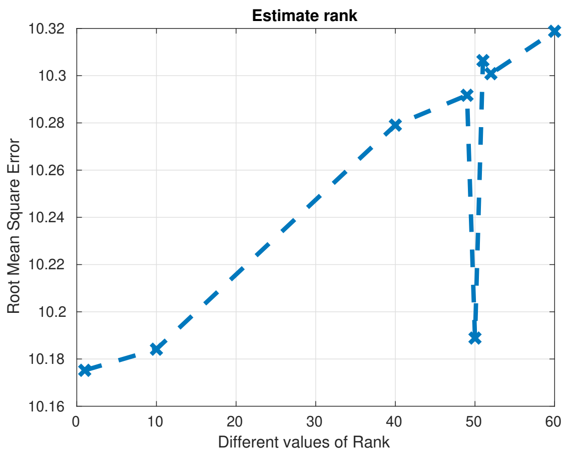
For the rank–50 tensor, we set the number of epochs to 100, $c = 0.2$ and $\lambda = 10^{-3}$. We test for $R = 1, 10, 40, 49, 50, 51, 52, 60$. In Fig. 7.10 we illustrate the results of the aforementioned model selection process.

Again, we observe that in higher SNR the estimated rank $R = 50$ is the actual factorization rank.

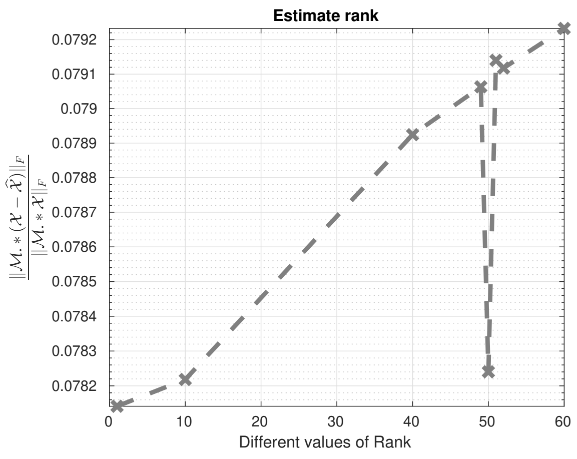
7.1.4 Real-World Data

7.1.4.1 Movielens - 10M

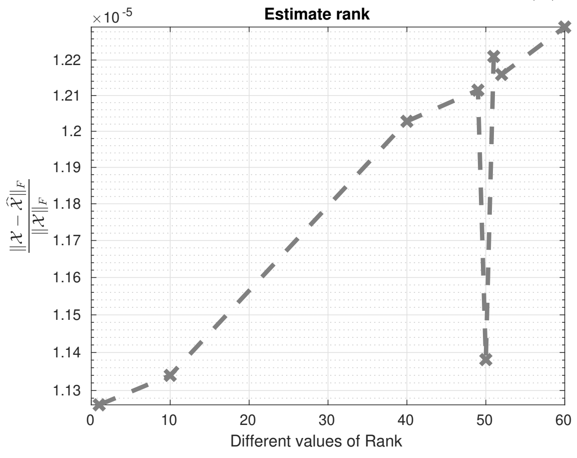
First we consider the "Movielens-10M" dataset [26]. This dataset contains 10000054 ratings (in the range $0 - 5$) applied to 65133 movie ids by 71567 users of the online movie recommender service MovieLens. Each rating contains also its (almost unique) timestamp and, therefore, we can group each rating for 730 weeks. Thus, this dataset can be represented as a third-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{71567\times65133\times730}$. We highlight that the true rank is not known, thus we need to follow the 5-Fold Cross Validation process to select the appropriate model for our data. We test our method using different values for the rank R.



(a) RMSE



(b) RIFE



(c) RFE

Figure 7.7: RMSE (a), RIFE (b) and RFE (c) vs Different values of rank for the synthetic (R50) noisy (10 dB) dataset.

Namely, inspired by the work in [27], we set $R = 1, 10, 20, 30, 40, 50$. We keep the convention of 80% - 20% for the train and test sets. The number of epochs is set to 50, whereas inner iterations are set to 1. Also, we set $c = 0.5$ and $\lambda = 0.001$. As for performance metrics, we use both the RFE and the RMSE. In Fig. 7.11 we illustrate the results of the aforementioned model selection process.

We observe that $R = 10$, gives both the lowest RMSE and RFE on the test set, thus we can claim that $R = 10$ is a good choice for our model.

7.1.4.2 Chicago Crime (Communities)

The second real-world dataset that we use is the "Chicago Crime" Dataset, publicly available in [28], that contains the crime reports in the city of Chicago, ranging from January 1st, 2001 to December 11th, 2017 (a duration of 6186 days). Non-zeros are counts and modes provide information such as time (days and hours), location (77 communities), and type of crime (32 types). Thus, we obtain a fourth order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{6186\times 24\times 77\times 32}$ with 5.3 million nonzeros. We test our method using different values for the rank R. Namely, we set $R = 1, 10, 20, 30, 40, 50$. The epochs are set to 50, $c = 0.5$ and $\lambda = 0.001$. In Fig.

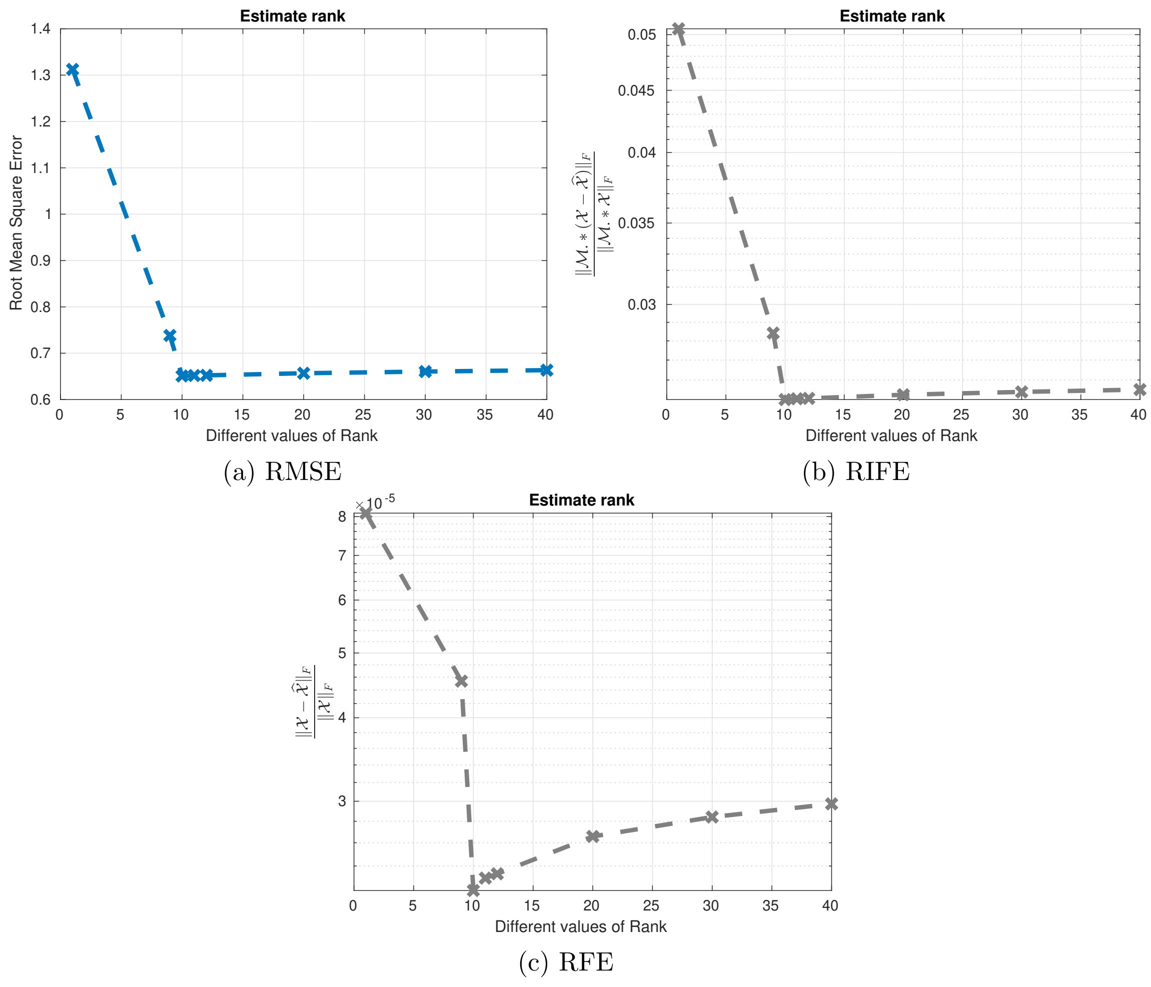


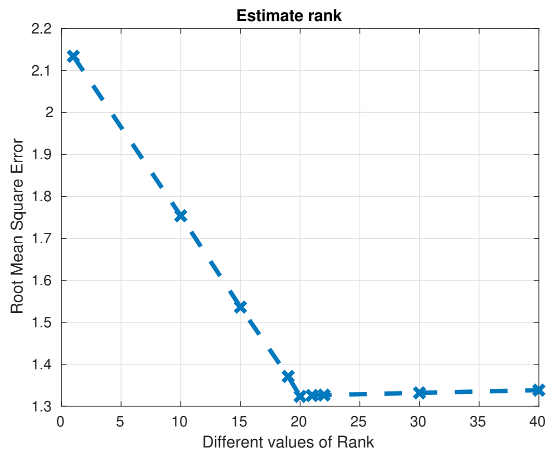
Figure 7.8: RMSE (a), RIFE (b) and RFE (c) vs Different values of rank for the synthetic (R10) noisy (20 dB) dataset.

7.12 we illustrate the results of the aforementioned model selection process.

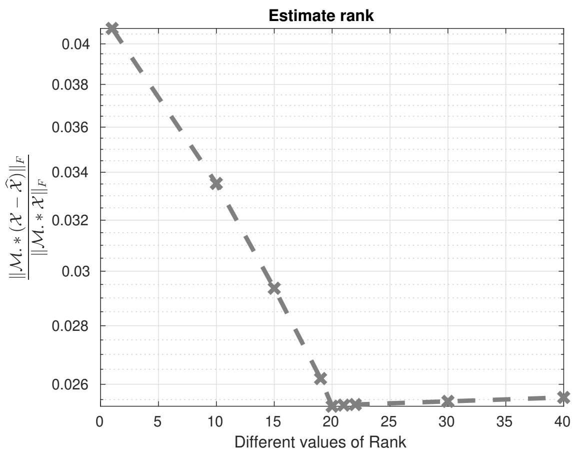
We observe that $R = 10$, gives both the lowest RMSE and RFE on the test set, thus $R = 10$ is suitable for our model. We note that in [29] they also use $R = 10$ without any further explanation.

7.1.4.3 Uber Pickups

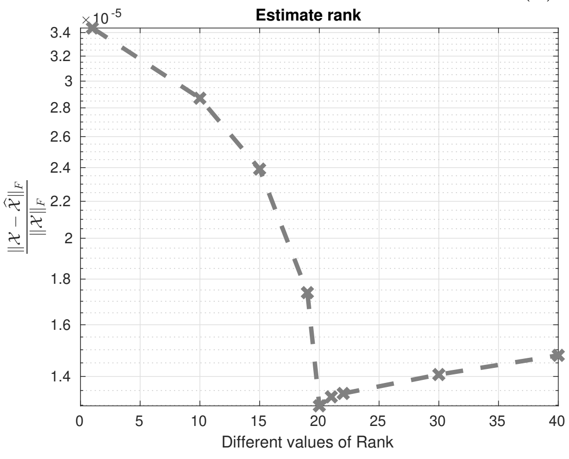
The next real-world dataset that we use is the "Uber Pickups" Dataset [28], that contains six months of Uber pickup data in New York City. Data covers April 2014 through September 2014 ($01 - 04 - 2014$ until $30 - 09 - 2014$). Non-zeros are integer counts and modes provide information such as date (183 days), hour ($0 - 23$), latitude and longitude. Latitude and Longitude values are rounded to three decimal places (i.e., 110 meters of resolution). Thus, we obtain a fourth order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{183\times 24\times 1140\times 1717}$ with 3.3 million nonzeros. We test our method using different values for the rank R. Namely, we set $R = 1, 10, 20, 25, 30, 35, 40, 45, 50, 60, 70, 90$. The epochs are set to 200, $c = 0.5$ and $\lambda = 10^{-4}$. In Fig. 7.13 we illustrate the results of the aforementioned model selection process.



(a) RMSE



(b) RIFE



(c) RFE

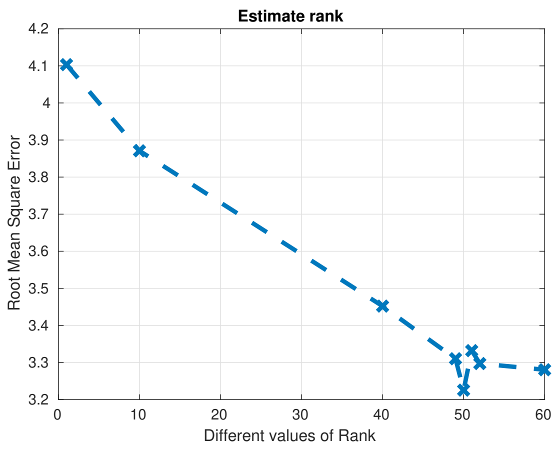
Figure 7.9: RMSE (a), RIFE (b) and RFE (c) vs Different values of rank for the synthetic (R20) noisy (20 dB) dataset.

We observe that $R = 30$ is a good approximation of the tensor's rank.

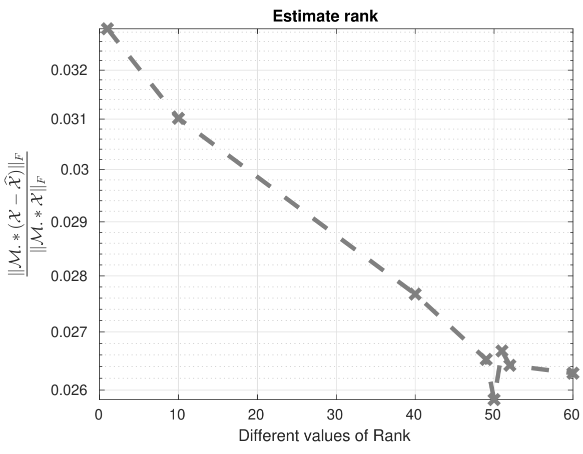
7.1.4.4 NIPS Publications

The next real-world dataset that we use is the "NIPS Publications" Dataset [28], that contains papers published in NIPS from 1987 to 2003, collected by [30]. Non-zeros are integer counts of words and each mode represent paper IDs (2482), paper authors (2862), vocabulary (14036 words) and years(17). Thus, we obtain a fourth order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{2482\times 2862\times 14036\times 17}$ with 3.1 million nonzeros. We test our method using different values for the rank R. Namely, we set $R = 1, 10, 20, 30, 40, 50, 60, 70, 90$. The epochs are set to 200, $c = 0.5$ and $\lambda = 10^{-4}$. In Fig. 7.14 we illustrate the results of the aforementioned model selection process.

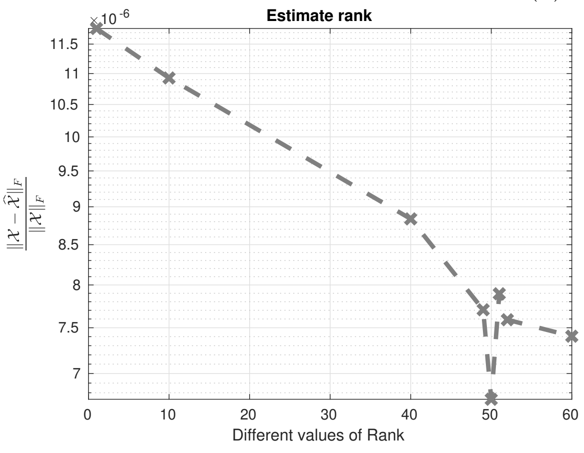
We observe that $R = 40, 70$ are good approximations of the tensor's rank. Since we prefer the lower possible rank, we conclude that rank $R = 40$ might be suitable for our model.



(a) RMSE

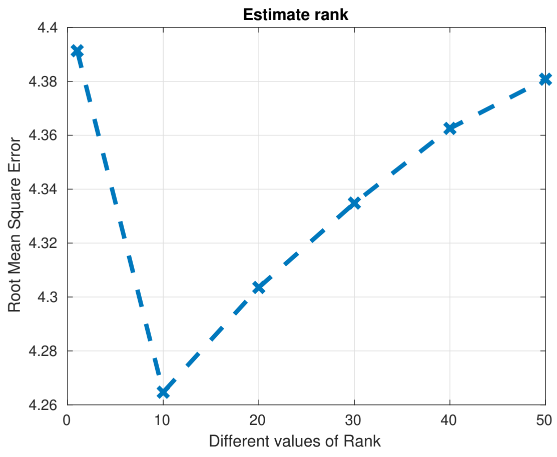


(b) RIFE

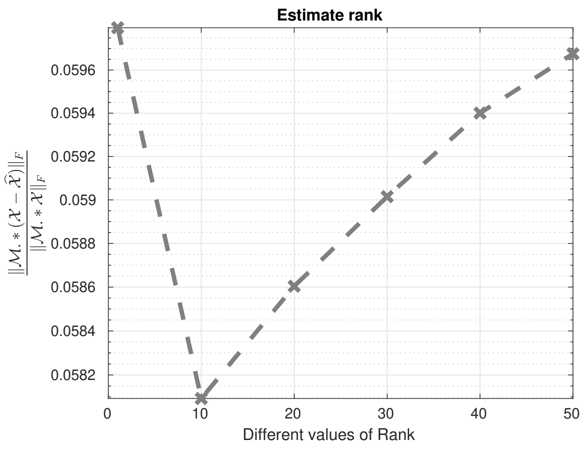


(c) RFE

Figure 7.10: RMSE (a), RIFE (b) and RFE (c) vs Different values of rank for the synthetic (R50) noisy (20 dB) dataset.

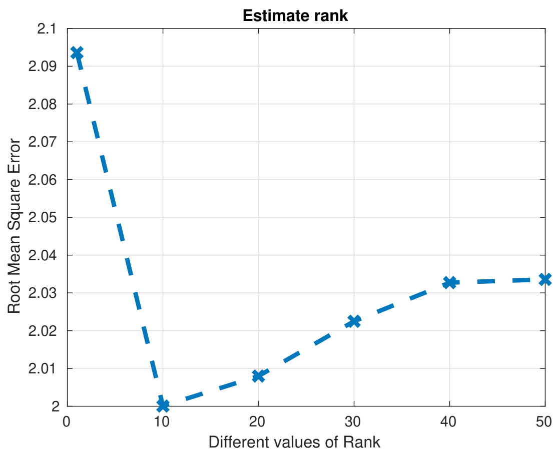


(a) RMSE

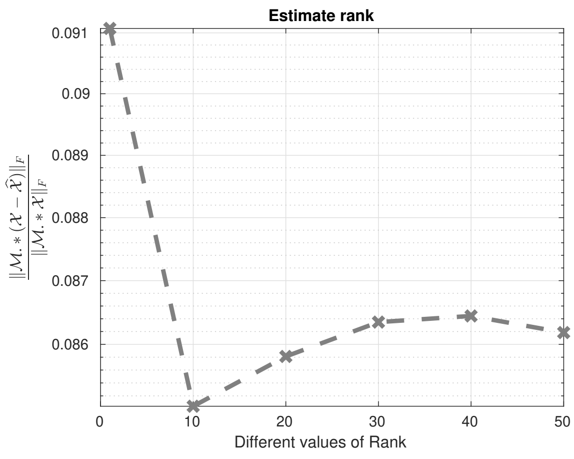


(b) RFE

Figure 7.11: RMSE (a) and RIFE (b) vs Different values of rank for the Movielens10M dataset.

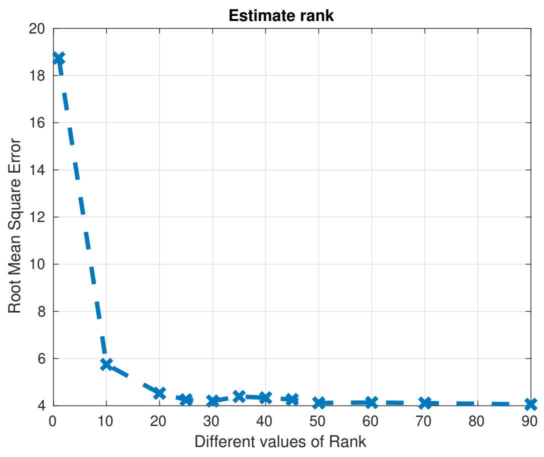


(a) RMSE

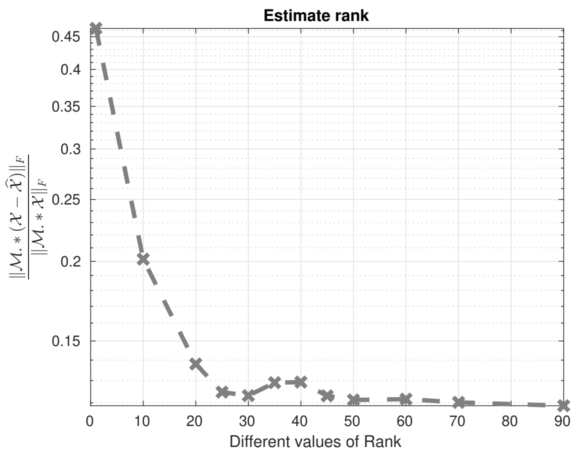


(b) RFE

Figure 7.12: RMSE (a) and RIFE (b) vs Different values of rank for the Chicago Crime dataset.

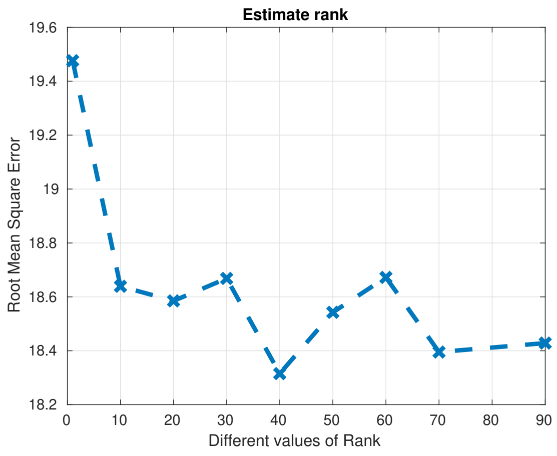


(a) RMSE

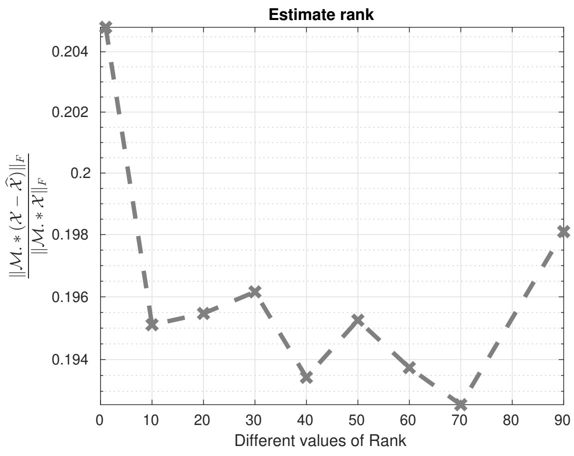


(b) RFE

Figure 7.13: RMSE (a) and RIFE (b) vs Different values of rank for the Uber Pickups dataset.



(a) RMSE

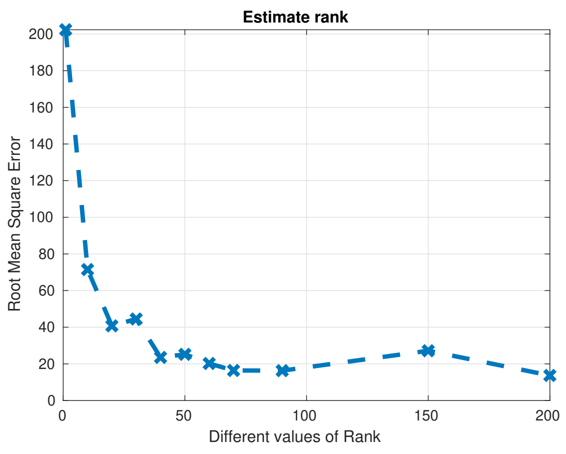


(b) RFE

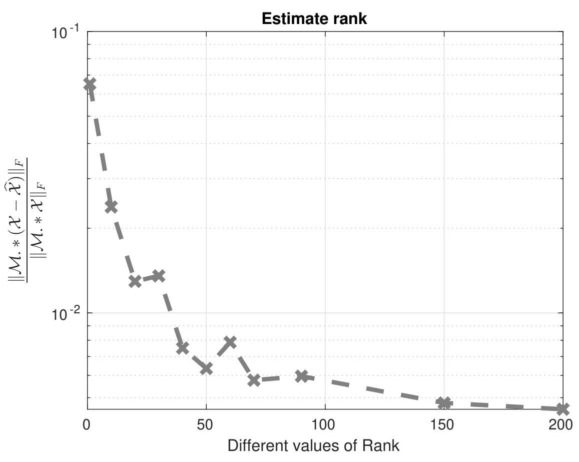
Figure 7.14: RMSE (a) and RIFE (b) vs Different values of rank for the NIPS dataset.

7.1.4.5 Corrupted Image (Missing Values)

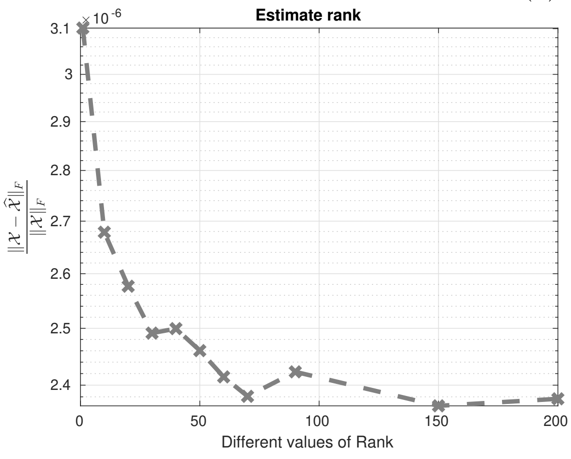
In order to approximate the best decomposition rank for our damaged image, presented in Chapter 6.1, we test our method using different values for the rank R. Namely, we set $R = 1, 10, 20, 30, 40, 50, 60, 70, 90, 150, 200$. The epochs are set to 100 and the inner iterations are set to 5. Also, we set $c = 0.2$ and $\lambda = 10^{-4}$. In Fig. 7.15 we illustrate the results of the aforementioned model selection process.



(a) RMSE



(b) RIFE



(c) RFE

Figure 7.15: RMSE (a), RIFE (b) and RFE (c) vs Different values of rank for the corrupted image dataset.

We observe that ranks $R = 50, 70, 150, 200$, give the lowest RMSE, RIFE, and RFE on the test set. Since we prefer the lower possible rank, we conclude that values $R = 50, 70$ might be suitable for our model.

7.2 Convergence speed of NTC via Stochastic NMLSME

In this section, we test the effectiveness of our algorithm in terms of convergence, for various test cases, using both synthetic and real-world data. In all of the experiments we compute averages over 5 Monte Carlo trials. We set $c = 0.2$ and 100 epochs. We compare the proposed step-sizes for 1 (1 nes. iters.) and 5 (5 nes. iters.) inner iterations, with

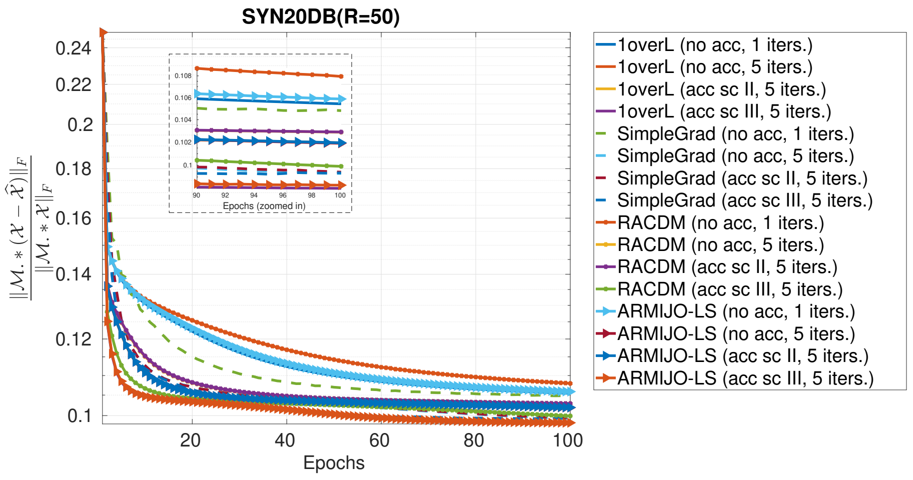


Figure 7.16: Relative factorization error vs the number of epochs for the nonnegative synthetic noisy tensor (20dB).

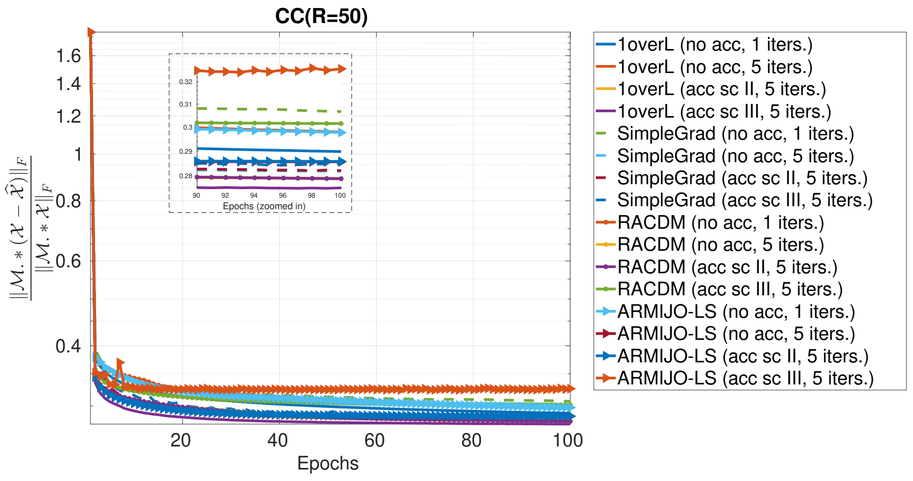


Figure 7.17: Relative factorization error vs the number of epochs for the nonnegative real-world tensor (Chicago Crime)

(acc) and without (no acc) acceleration. The results are illustrated in Fig. 7.16 - 7.20. In Fig. 7.16, both methods “1 over L” and “ARMIJO-LS”, with 5 inner iterations and scheme III, are the most effective. In Fig. 7.17, methods “1 over L” and “RACDM”, both with 5 inner iterations and scheme III, are the most competitive. Finally, in Fig. 7.18 - 7.20, we observe that methods “RACDM” and “ARMIJO-LS”, with scheme II and without acceleration, and with 5 inner iterations, achieve the lowest relative factorization error.

7.3 Execution time for parallel NTC via Stochastic NMLSME

In this section, we test the effectiveness of our parallelized algorithm in terms of execution time. This experiment is executed on a DELL PowerEdge R820 system with processor type Sandy Bridge - Intel(R) Xeon(R) CPU E5-4650v2 (4 sockets per node - 10 cores per

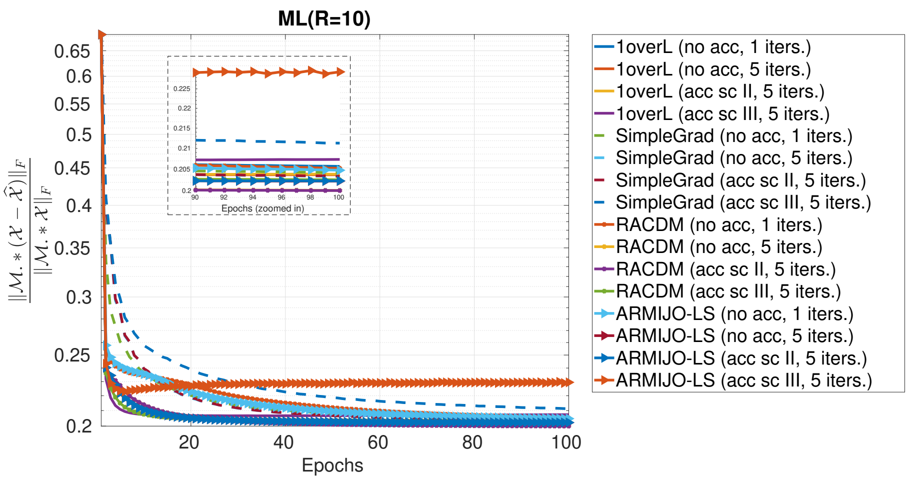


Figure 7.18: Relative factorization error vs the number of epochs for the nonnegative real-world tensor (MovieLens10M)

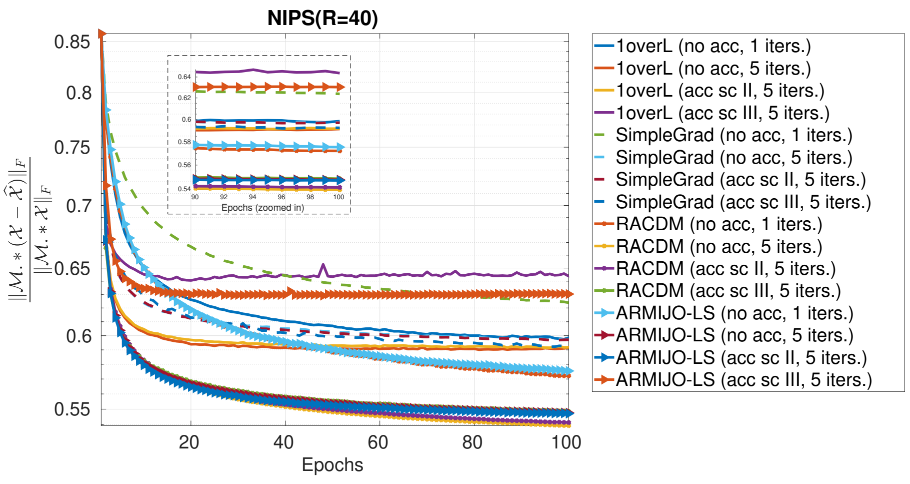


Figure 7.19: Relative factorization error vs the number of epochs for the nonnegative real-world tensor (NIPS)

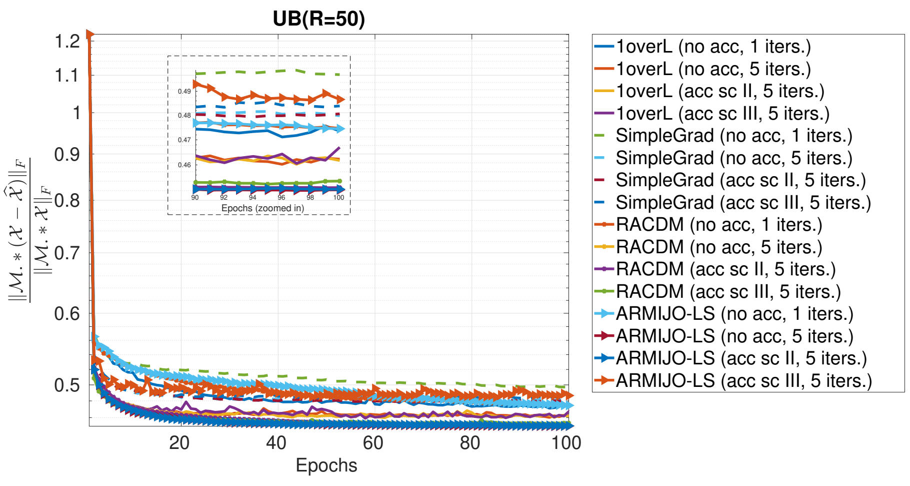


Figure 7.20: Relative factorization error vs the number of epochs for the nonnegative real-world tensor (Uber Pickups)

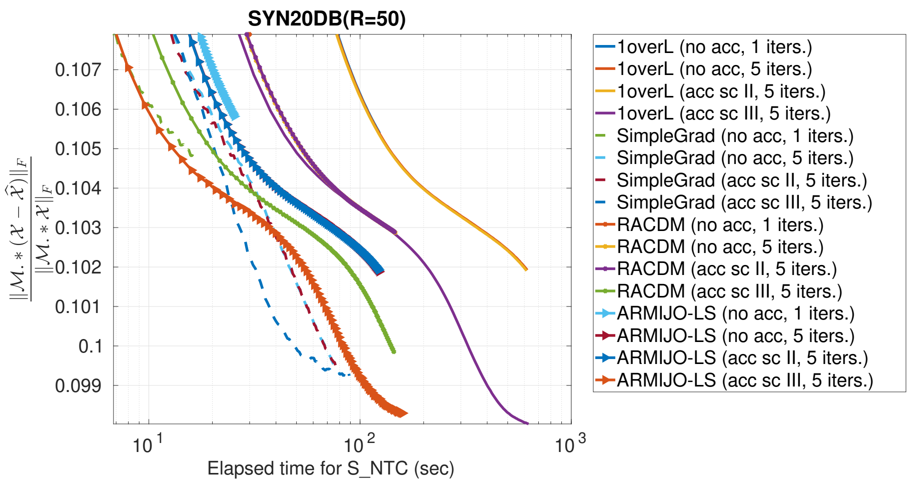


Figure 7.21: Relative factorization error vs execution time for the nonnegative synthetic noisy tensor (20dB)

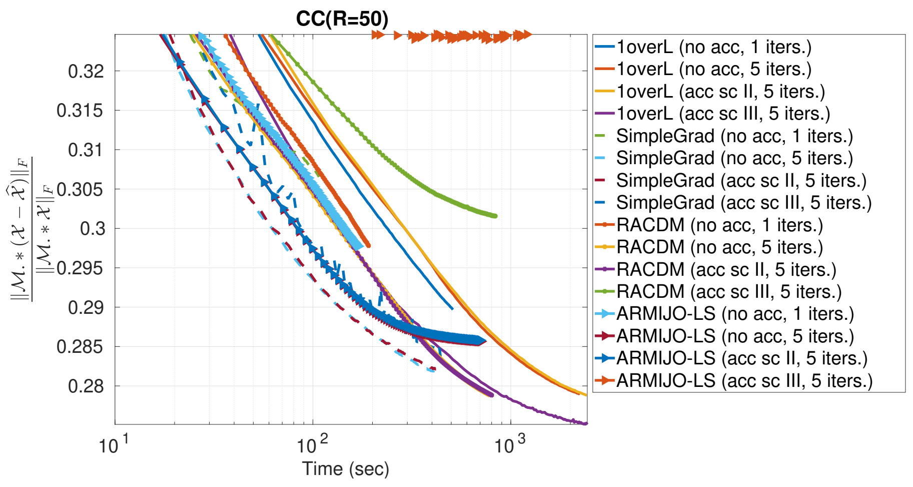


Figure 7.22: Relative factorization error vs execution time for the nonnegative real-world tensor (Chicago Crime)

socket) and 512 GB RAM per node at ARIS supercomputer [2]. We use 40 physical threads where each thread runs on a separate core with hyperthreading disabled. We set $c = 0.2$ and the number of epochs equal to 100. Again, we compare the proposed step-sizes for 1 and 5 inner iterations, with and without acceleration.

In Fig. 7.21, the most competitive method is “SimpleGrad” with 5 inner iterations and scheme III. In Fig. 7.22, method “SimpleGrad” with and without acceleration (scheme II), and 5 inner iterations is the most competitive. In Fig. 7.23, methods “SimpleGrad” (with 1 inner iteration and without acceleration) and “ARMIJO-LS” (with and without scheme II and 5 inner iterations) are the most efficient. Finally, in Fig. 7.24, we observe that methods “RACDM” and “ARMIJO-LS”, with 1 inner iteration and no acceleration, converge fast. Finally, in Fig. 7.25, methods “RACDM” (scheme II) and “ARMIJO-LS” (scheme III) are the most competitive for 5 inner iterations.

We observe that the methods “1 over L”, “RACDM” and “ARMIJO-LS” are very

[2]Greek Research and Technology Network - High Performance Computing Services https://hpc.grnet.gr

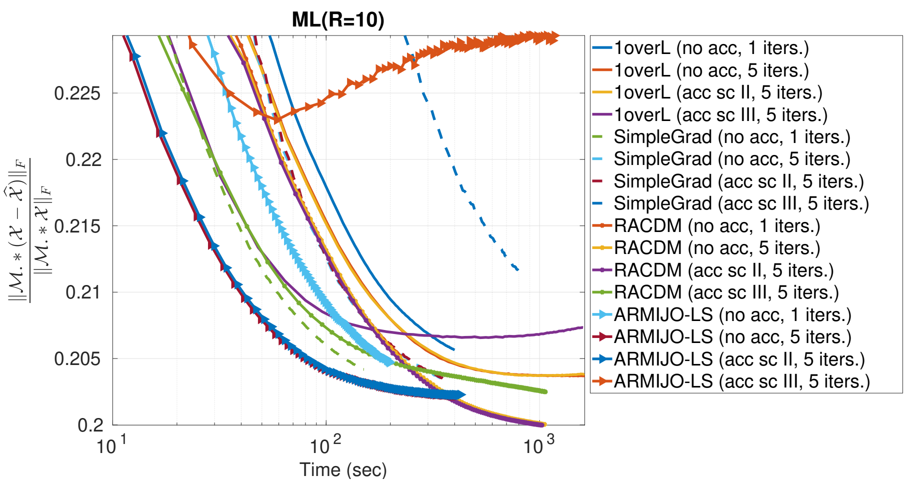


Figure 7.23: Relative factorization error vs execution time for the nonnegative real-world tensor (MovieLens10M)

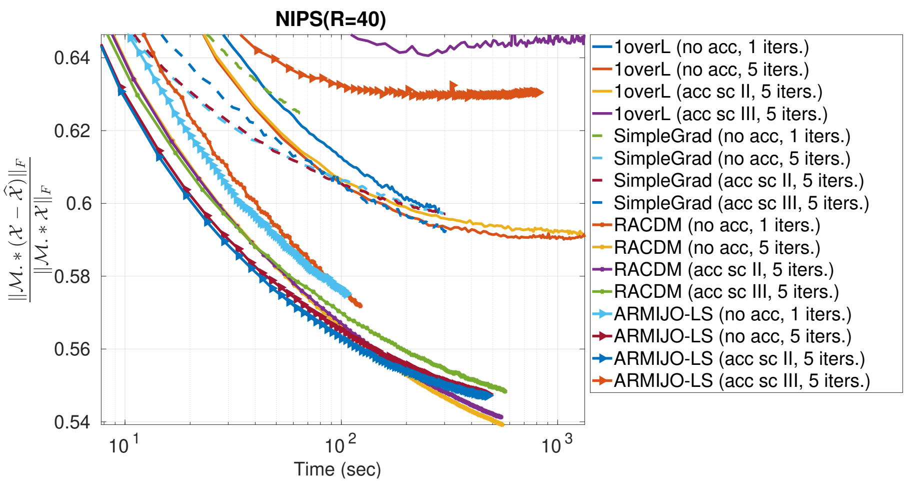


Figure 7.24: Relative factorization error vs execution time for the nonnegative real-world tensor (NIPS)

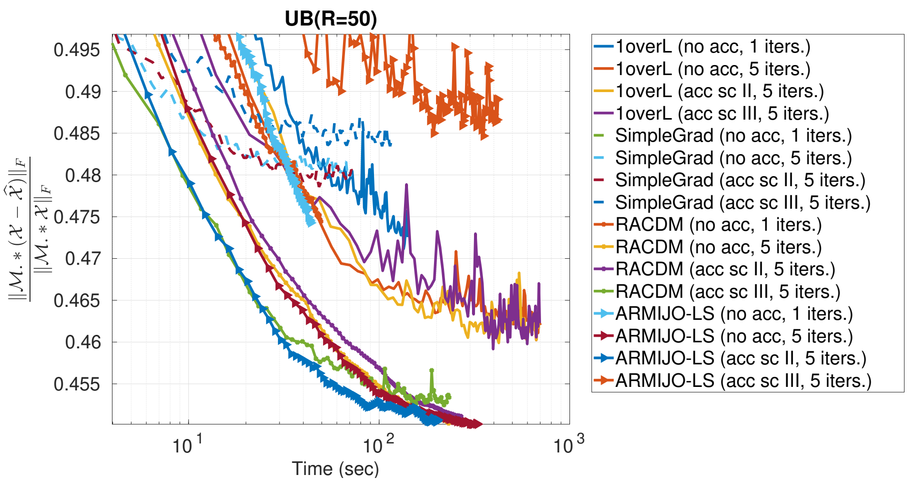


Figure 7.25: Relative factorization error vs execution time for the nonnegative real-world tensor (Uber Pickups)

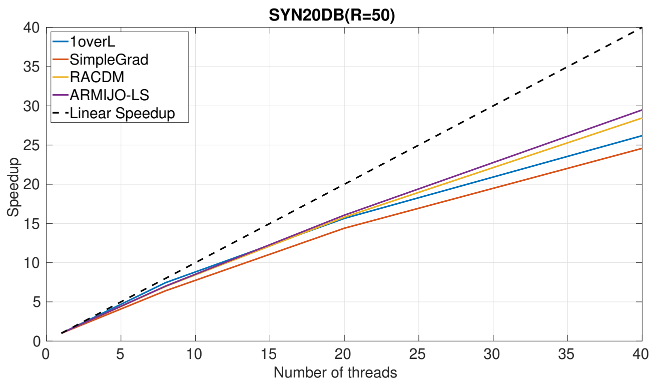


Figure 7.26: Speedup achieved vs the number of threads for the parallel implementation of stochastic NMLSME using the nonnegative synthetic tensor.

competitive concerning the convergence rate. On the other side, the method "SimpleGrad" is less accurate but is very fast in terms of execution time.

7.4 Speedups for parallel NTC via Stochastic NMLSME

In this section, we test the performance of our parallel implementation, for all the proposed step-sizes. We set the number of epochs to 10 and we test for 1 inner iteration (no acceleration). We test for $t = 1, 8, 20, 40$ physical threads.

Usually, the best our parallel program can do is to divide the work equally among the cores, while at the same time introducing no additional work for the cores. If we succeed in doing this, and we run our program with t cores, one thread on each core, then our parallel program will run t times faster than the serial runs on a single core of the same design.

To measure the effectiveness of our parallelism, we use the speedup metric, defined as

$$\text{speedup} = \frac{T_1}{T_t}, \tag{7.5}$$

where T_1 is the execution time for 1 thread (serial) and T_t is the execution time for $t > 1$ number of threads (parallel).

The optimal speedup, also known as "linear speedup" is the best possible run-time of our parallel program. If the speedup is equal to the number of threads, then we say that our parallel program has achieved linear speedup.

In Figures 7.26 and 7.27, we illustrate the achieved speedup, for the nonnegative noisy tensor and the real-world tensor from the "Uber Pickups" dataset. We observe that for the synthetic dataset, our parallel implementation is very close to the linear speedup. In the case of the "Uber pickups" dataset, we observe that our algorithm performs slightly worse due to the load imbalance among threads. More specifically, we believe that the distribution of the nonzero entries in the $3rd$ and $4th$ dimension (latitude-longitude) affects our parallel implementation.

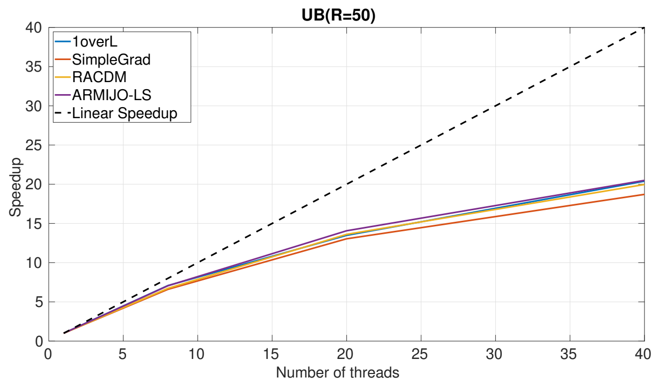


Figure 7.27: Speedup achieved vs the number of threads for the parallel implementation of stochastic NMLSME using the real-world tensor from the “Uber Pickups” dataset.

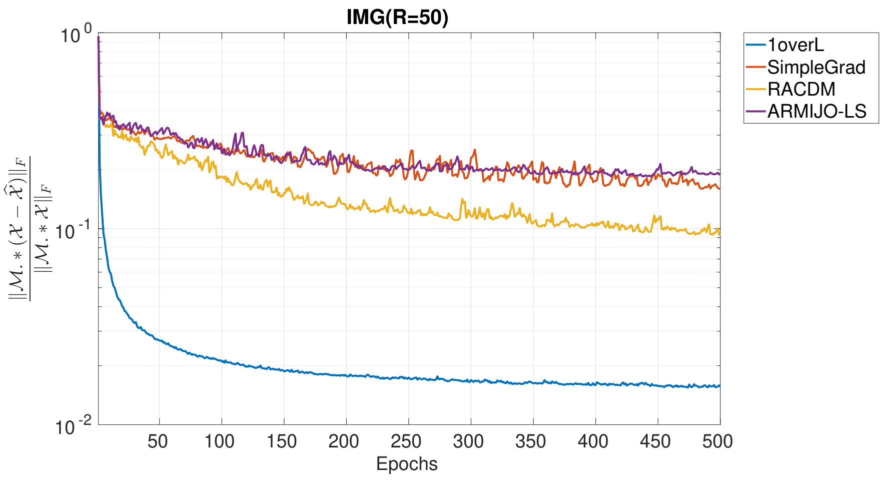


Figure 7.28: Relative cost function vs. number of epochs, using 1 inner iteration and Rank R = 50.

7.5 Stochastic NTC on corrupted image

We use the same corrupted image (Fig. 6.2b), which we illustrate again in Fig. 7.29a. Since the given incomplete tensor $\boldsymbol{\mathcal{X}}$ is in $\mathbb{R}_+^{1063\times1599\times3}$, the resulted nonnegative factors $\{\mathbf{U}_k^{(i)} \in \mathbb{R}_+^{I_i\times R}\}_{i=1}^3$ give us a full tensor $\widehat{\boldsymbol{\mathcal{X}}}$ also in $\mathbb{R}_+^{1063\times1599\times3}$.

We use the CPD factorization model, and more specifically, we use the proposed NTC via the S_NMLSME method. We start from initial factors $\mathbf{U}^{o(i)} \in \mathbb{R}_+^{I_i\times R}$ for $i = 1, 2, 3$, with i.i.d. elements in $\mathcal{U}[0, 1]$, and we set $c = 0.02$, $\lambda = 10^{-6}$, number of epochs 500, 1 inner iteration, and rank $R = 50$. From all the proposed step–sizes, the one that performs better in terms of convergence is “1 over L”, as we show in Fig. 7.28.

We generate the tensor $\widehat{\boldsymbol{\mathcal{X}}}$ from the resulted factors $\mathbf{A} = \mathbf{U}^{(1)} \in \mathbb{R}_+^{1063\times50}$, $\mathbf{B} = \mathbf{U}^{(2)} \in \mathbb{R}_+^{1599\times50}$, and $\mathbf{C} = \mathbf{U}^{(3)} \in \mathbb{R}_+^{3\times50}$, using the cpdgen() function. In order to illustrate the resulted tensor as a true image, we need to convert it back to the tensor $\widehat{\boldsymbol{\mathcal{I}}} = \text{uint8}(\widehat{\boldsymbol{\mathcal{X}}}) \in \mathbb{Z}_+^{1063\times1599\times3}$, using function uint8() of MATLAB. In Fig. 7.30 we illustrate the above methodology using a high–level block–diagram. Finally, in Fig. 7.29 we illustrate side by side the corrupted image (a) versus the restored one (b). We observe that the algorithm can reconstruct the corrupted image even for small values of c.

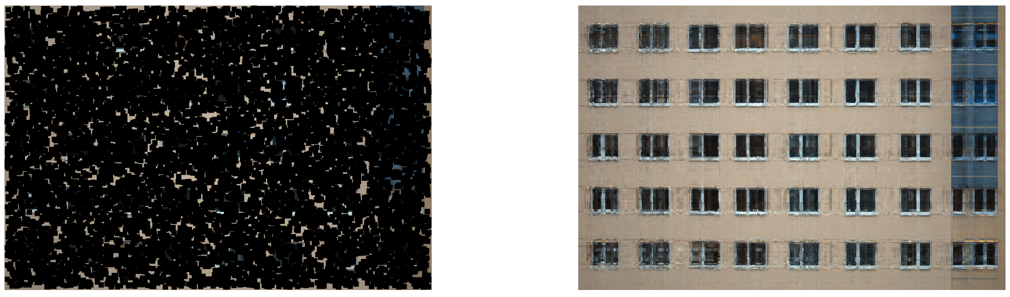

(a) Corrupted (90% sparse) (b) Restored

Figure 7.29: Tensor Completion on a corrupted image.

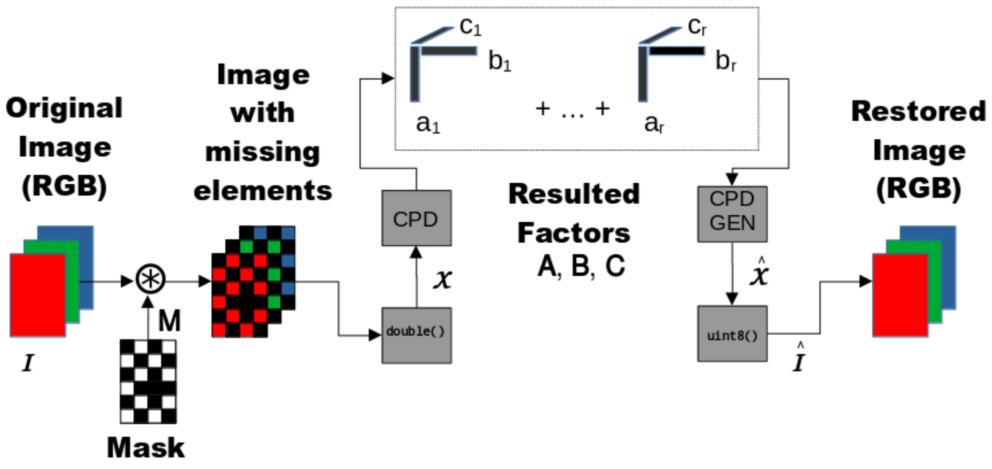


Figure 7.30: Block diagram of image corruption and restoration

7.6 Word Embeddings

In this section, we present a quantitative evaluation comparing our embeddings versus the Word2Vec method, described in section 6.2.3.2.

7.6.0.1 Data Description

To compare those two methods we used the “abcnews” dataset [3]. Sourced from the reputable Australian news source ABC (Australian Broadcasting Corporation), this dataset contains data of news headlines published for eighteen years (Start Date: 2003-02-19 - End Date: 2020-12-31). It contains historical moments of the last decade, for example Afghanistan war, financial crisis, multiple elections, ecological disasters, terrorism, etc.

7.6.0.2 Settings

We first extract the unigrams, bigrams, and trigrams from our corpus using the skip-gram model with a window of size 5. We remove all the infrequent words, by setting a

[3]it can be found in `https://www.kaggle.com/therohk/million-headlines`

minimum threshold word count equal to 20. After this process, we reduce the vocabulary size from 115876 to 19751 words. Then, we compute the supersymmetric PPMI tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}_{+}^{19751\times 19751\times 19751}$ as we showed in section 6.2.4.

To extract the embeddings we use the Pseudosymmetric AO NTC via the S_NMLSME method. We choose the step–size of the "SimpleGrad" method since it has the lowest execution time. We set the factorization rank $R = 300$ ($d = R$), which is a common value [25],[21], since is both rich enough, and does not require an excessive amount of memory. We set parameter $c = 0.001$ and we start from $\mathbf{U}^{o(1)} = \mathbf{U}^{o(2)} = \mathbf{U}^{o(3)} \in \mathbb{R}_{+}^{19751\times 300}$, with i.i.d. elements in $\mathcal{U}[0,1]$. We tested our method using 100 epochs and one inner iteration. Both the computation of the PPMI and its factorization was able to run on a laptop with 6 cores and 16 GB RAM in less than 3 hours. As for the Word2Vec method, it is implemented using the Tensorflow library [4]. The training of the model required 40 epochs and was very computational and memory demanding. It required a minimum of 32 GB RAM and took about 12 hours in total.

7.6.0.3 Vector similarity

In order to measure the similarity between two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$, we use the Euclidean distance (L2-norm)

$$dist_{L2}(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{N}(x_i - y_i)^2}. \tag{7.6}$$

7.6.0.4 Additive Compositionality

Authors in [22] demonstrate the additive compositionality of their Word2Vec vectors. To be more precise, one can sum vectors produced by their embedding to compute vectors for certain phrases rather than just vectors for words. For example, assume words w_1="Germany" and w_2="capital". Sum $\mathbf{u}_{w_1} + \mathbf{u}_{w_2}$ is close to vector representation $\mathbf{u}_{w_3}$ of word w_3="Berlin". This compositionality suggests that a non-obvious degree of language understanding can be obtained by using basic mathematical operations on the word vector representations.

7.6.0.5 Multiplicative Compositionality

We examine if our tensor-based embeddings capture 3^{rd} order relationships between words, through multiplicative compositionality, which was firstly introduced by [21]. More specifically, one can create a vector that represents a word w_1 in the context of another word w_2 by taking the elementwise product $\mathbf{u}_{w_1} \circledast \mathbf{u}_{w_2}$. This product is called "meaning vector" [21] for the word w_1. Using the third-order PPMI tensor representation, for any triplet w_i, w_j, w_k,

$$\boldsymbol{\mathcal{M}}(i,j,k) \approx \sum_{r=1}^{R} u_{ir}u_{jr}u_{kr} = (\mathbf{u}_i \circledast \mathbf{u}_j)^T \mathbf{u}_k, \tag{7.7}$$

[4]A useful tutorial of Word2Vec implementation using the Tensorflow library can be found in `https://github.com/tensorflow/docs/blob/master/site/en/tutorials/text/word2vec.ipynb`

where $\mathbf{u}_l$ is the word vector for $w_l\,(l = i, j, k)$. If words w_i, w_j, w_k have a high PPMI, then the result of product $(\mathbf{u}_i \circledast \mathbf{u}_j)^T \mathbf{u}_k$ will also be high. This means that product $\mathbf{u}_i \circledast \mathbf{u}_j$ will be close to vector $\mathbf{u}_k$ in the vector space by euclidean distance.

In Table 7.1 we present the nearest neighbors of multiplicative and additive composed vectors for a variety of words. As we can see, the words corresponding to the nearest neighbors of the composed vectors for our tensor method are semantically related to the intended sense both for multiplicative and additive composition. On the other side, for Word2Vec, only additive composition gives vectors whose nearest neighbors are semantically related to the desired sense. We observe that our method is very effective and can be considered as a competitive candidate for NLP tasks.

Table 7.1: Nearest neighbors (in normalized euclidean distance) to elementwise products/additions of word vectors

Composition	Nearest neighbors (Symmetric NTC)	Nearest neighbors (Word2Vec)
boko + haram	mali	bloody
boko * haram	hezbollah	sayyaf
hikers + hiker	climbers	swims
hikers * hiker	climbers	racq
deluge + downpour	drenching	drenching
deluge * downpour	drenching	belgium
covid19 + vaccine	pfizer	coronavirus
covid19 * vaccine	pfizer	pedestrians
israel + palestine	evacuate	settler
israel * palestine	assess	apologise
global + warming	g7	climate
global * warming	seniors	bloke
gun + weapon	loaded	knives
gun * weapon	stun	user

Chapter 8

Conclusion and Future Work

8.1 Conclusions

We considered the NTC problem. First, we developed an accelerated stochastic algorithm for the NMLSME problem. A unique feature of our approach is that each row of the matrix variable is updated using a different step–size, specifically tailored to this row. We experimented with various step–sizes. Then, we used this algorithm and built an AO algorithm for the NTC problem. We tested the data reconstruction effectiveness as well as the convergence speed of our approach using both synthetic and real-world data. We implemented our algorithm using the OpenMP API, and observed significant speedup. Finally, we presented some real–world applications that can be interpreted as NTC problems and can be solved efficiently through our method.

8.2 Future work

Finally, we conclude this thesis by presenting possible future extensions of this work.

First of all, developing algorithms that fully exploit the hessian matrix is of high interest. Also, convergence analysis of the proposed algorithm is an interesting future topic.

An extension to higher-order PPMI tensors could be a possible topic of interest concerning word embeddings.

Another topic of interest could be the studying and comparison of other image restoration algorithms, in terms of effectiveness and elapsed time.

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, no. 3, pp. 455–500, September 2009.

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

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

[6] 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," *IEEE Transactions on Signal Processing*, vol. 66, no. 4, pp. 944–953, Feb. 2018.

[7] P. A. Karakasis, C. Kolomvakis, G. Lourakis, G. Lykoudis, I. M. Papagiannakos, I. Siaminou, C. Tsalidis, and A. P. Liavas, "Partensor," *Tensors for Data Processing: Theory, Methods, and Applications*, pp. 66–90, 2021.

[8] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400 – 407, 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729586

[9] A. B. Juditsky, A. S. Nemirovski, G. Lan, and A. Shapiro, "Stochastic Approximation Approach to Stochastic Programming," in *ISMP 2009 - 20th International Symposium of Mathematical Programming*, Chicago, United States, Aug. 2009. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00981931

[10] A. Beck, *First-Order Methods in Optimization.* Philadelphia, PA: Society for Industrial and Applied Mathematics, 2017. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611974997

[11] Y. Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems," *Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), CORE Discussion Papers*, vol. 22, 01 2010.

[12] L. Armijo, “Minimization of functions having Lipschitz continuous first partial derivatives.” *Pacific Journal of Mathematics*, vol. 16, no. 1, pp. 1 – 3, 1966. [Online]. Available: https://doi.org/

[13] S. Vaswani, A. Mishkin, I. Laradji, M. Schmidt, G. Gidel, and S. Lacoste-Julien, “Painless stochastic gradient: Interpolation, line-search, and convergence rates,” 2021.

[14] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, and H. Asai, “A stochastic quasi-newton method with nesterov’s accelerated gradient,” in *Machine Learning and Knowledge Discovery in Databases*, U. Brefeld, E. Fromont, A. Hotho, A. Knobbe, M. Maathuis, and C. Robardet, Eds. Cham: Springer International Publishing, 2020, pp. 743–760.

[15] I. Siaminou, I. M. Papagiannakos, C. Kolomvakis, and A. P. Liavas, “Accelerated stochastic gradient for nonnegative tensor completion and parallel implementation,” 2021.

[16] OpenMP Architecture Review Board, “Openmp application program interface,” Specification, 2015. [Online]. Available: https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf

[17] T. Papastergiou and V. Megalooikonomou, “A distributed proximal gradient descent method for tensor completion,” in *2017 IEEE International Conference on Big Data (Big Data)*, 2017, pp. 2056–2065.

[18] J. Liu, P. Musialski, P. Wonka, and J. Ye, “Tensor completion for estimating missing values in visual data,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 208–220, 2013.

[19] Z. S. Harris, “Distributional structure,” *¡i¿WORD¡/i¿*, vol. 10, no. 2-3, pp. 146–162, 1954. [Online]. Available: https://doi.org/10.1080/00437956.1954.11659520

[20] O. Levy and Y. Goldberg, “Neural word embedding as implicit matrix factorization,” in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/feab05aa91085b7a8012516bc3533958-Paper.pdf

[21] E. Bailey and S. Aeron, “Word embeddings via tensor factorization,” 2017.

[22] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” 2013.

[23] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts: The MIT Press, 1999. [Online]. Available: http://nlp.stanford.edu/fsnlp/

[24] D. Guthrie, B. Allison, W. Liu, L. Guthrie, and Y. Wilks, “A closer look at skip-gram modelling,” in *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC’06).* Genoa, Italy: European Language Resources Association (ELRA), May 2006. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2006/pdf/357_pdf.pdf

[25] T. Van de Cruys, T. Poibeau, and A. Korhonen, “A tensor-based factorization model of semantic compositionality,” in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* Atlanta, Georgia: Association for Computational Linguistics, Jun. 2013, pp. 1142–1151. [Online]. Available: https://aclanthology.org/N13-1134

[26] 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

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

[28] 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/

[29] K. D. Devine and G. Ballard, “Gentenmpi: Distributed memory sparse tensor decomposition.” [Online]. Available: https://www.osti.gov/biblio/1656940

[30] A. Globerson, G. Chechik, F. Pereira, and N. Tishby, “Euclidean Embedding of Co-occurrence Data,” *The Journal of Machine Learning Research*, vol. 8, pp. 2265–2295, 2007.