ALGORITHMS FOR TENSOR DECOMPOSITIONS AND APPLICATIONS

PHAN ANH HUY

KYUSHU INSTITUTE OF TECHNOLOGY

2011

Algorithms for Tensor Decompositions and Applications

by

PHAN ANH HUY

A Dissertation Submitted in
Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

In the

GRADUATE SCHOOL OF LIFE SCIENCE AND SYSTEMS ENGINEERING

KYUSHU INSTITUTE OF TECHNOLOGY

JAPAN

2011

KYUSHU INSTITUTE OF TECHNOLOGY

GRADUATE SCHOOL OF LIFE SCIENCE AND SYSTEMS ENGINEERING
DEPARTMENT OF BRAIN SCIENCE AND ENGINEERING

P H D T H E S I S

Algorithms for Tensor Decompositions and Applications

PHAN ANH HUY

Thesis Advisor: CICHOCKI ANDRZEJ

Thesis committee :

Chair :	MATSUOKA Kiyotoshi	-	KYUTECH
Reviewers :	HORIO Keiichi	-	KYUTECH
	FURUKAWA Tetsuo	-	KYUTECH
Advisor :	CICHOCKI Andrzej	-	BSI, RIKEN

July, 2011

Acknowledgments

The research reported in this PhD thesis has been carried out under the supervision of Dr. Andrzej Cichocki. A heartfelt thank goes out to Dr. Andrzej Cichocki for his help and scientific collaboration and also for the excellent working environment during the years I spent working in his laboratory.

I am grateful in particular to Professor Kiyotoshi Matsuoka giving me the opportunity to be a PhD student in his department, and to the examining committee members evaluating my work.

I would like to express my appreciation and gratitude to a number of researchers who helped to correct my publications, gave valuable comments and suggestions, especially Professor Danilo Mandic from Imperial College London, Professor Cesar Caiafa from Universidad de Buenos Aires, Argentina, Professor Saied Sanai from University of Surrey, UK, Dr. Petr Tichavský from Institute of Information Theory and Automation, Czech Republic, Dr. Sidney R. Lehky from Salk Institute, USA.

A special thank goes to all the coauthors, colleagues and collaborators: Professor Kiyotoshi Matsuoka, Professor Rafal Zdunek, Dr. Qibin Zhao, Dr. Fengyu Cong, Dr. Hovagim Bakardjian, Dr. Zbynek Koldovský, Professor Jianting Cao.

I would like to thank Dr. Francois Vialatte, Dr. Hovagim Bakardjian, Dr. Qibin Zhao, Dr. Fengyu Cong, Dr. Washisawa who provided and shared valuable EEG data and informative discussions.

Finally, I gratefully acknowledge my parent, my families and friends without whose support and understanding I would not have had the capacity to do this work. Special thanks to my family, especially my lovely son Hao Minh, his smile kept my spirits during the thesis work.

Contents

List of Symbols and Abbreviations

Principal Symbols

$\mathbb{R}_+$	nonnegative real number
$\mathbb{R}^n$	n-dimensional real vector space
$\mathbf{A}$	matrix
$\boldsymbol{a}_r$	vector, component of matrix $\mathbf{A} \in \mathbb{R}^{I \times R}$
$\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}$	tensors
$\boldsymbol{\mathcal{I}}$	identity tensor
$\hat{\boldsymbol{\mathcal{Y}}}$	an approximation of $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$
$\mathbf{Y}_{(n)}$	mode-n matricization of $\boldsymbol{\mathcal{Y}}$
$\boldsymbol{\mathcal{Y}}^{(k)}$	the k-th subtensor of $\boldsymbol{\mathcal{Y}}$ in which the last index $i_N = k$
$\{\mathbf{A}\}^\dagger$	Moore-Penrose pseudo-inverse of a matrix $\mathbf{A}$
$\mathbf{A} \geqslant \mathbf{0}$	nonnegative matrix $\mathbf{A}$ with all entries $a_{ij} \geqslant 0$
$\mathbf{A}^{(n)}$	factor matrix of a tensor decomposition $\mathbf{A}^{(n)} \in \mathbb{C}^{I_n \times R_n}$
$\mathbf{H}$	approximate Hessian
R	number of components in CP
R_n	number of components of factor $\mathbf{A}^{(n)}$ in Tucker decomposition
$\boldsymbol{i}$	index vector $[i_1, i_2, \ldots, i_N]$
$\boldsymbol{r}$	index vector $[r_1, r_2, \ldots, r_N]$
$\boldsymbol{r} \leqslant \mathbf{R}$	$r_n \leqslant R_n, \forall n$
$\odot$	Khatri-Rao product
$\otimes$	Kronecker product
$\circledast, \oslash$	Hadamard product and division
$\circ$	outer product
$\mathbf{S}_w$	within-class scatter matrix
$\mathbf{S}_b$	between-class scatter matrix
$\mathbf{S}_t$	total scatter matrix
K	number of training samples
K_c	number of training samples for class c
$\mathbf{P}_{I \times J}$	permutation matrix defined for an $I \times J$ matrix $\mathbf{X}$: $\text{vec}(\mathbf{X}) = \mathbf{P}_{I \times J}\ \text{vec}\left(\mathbf{X}^T\right)$
$\mathbf{P}_n$	permutation matrix defined for $\boldsymbol{\mathcal{X}}$: $\text{vec}(\boldsymbol{\mathcal{X}}) = \mathbf{P}_n\ \text{vec}\left(\mathbf{X}_{(n)}\right)$
$\{\mathbf{A}\}^{\otimes}$	$\bigotimes_{n=1}^{N} \mathbf{A}^{(n)} = \mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n)} \otimes \cdots \otimes \mathbf{A}^{(1)}$

$\{\mathbf{A}\}^{\otimes -n}$	$\bigotimes_{k\neq n}\mathbf{A}^{(k)} = \mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \cdots \otimes \mathbf{A}^{(1)}$
$\{\mathbf{A}\}^{\circledast}$	$\circledast_{n=1}^{N}\mathbf{A}^{(n)} = \mathbf{A}^{(N)} \circledast \cdots \circledast \mathbf{A}^{(n)} \circledast \cdots \circledast \mathbf{A}^{(1)}$
$\{\mathbf{A}\}^{\circledast -n}$	$\circledast_{k\neq n}\mathbf{A}^{(k)} = \mathbf{A}^{(N)} \circledast \cdots \circledast \mathbf{A}^{(n+1)} \circledast \mathbf{A}^{(n-1)} \circledast \cdots \circledast \mathbf{A}^{(1)}$
$\{\mathbf{A}\}^{\odot}$	$\bigodot_{n=1}^{N}\mathbf{A}^{(n)} = \mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n)} \odot \cdots \odot \mathbf{A}^{(1)}$
$\{\mathbf{A}\}^{\odot -n}$	$\bigodot_{k\neq n}\mathbf{A}^{(k)} = \mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \cdots \odot \mathbf{A}^{(1)}$
$\mathfrak{a} = \text{vec}(\mathcal{A})$	vectorization of $\mathcal{A}$
$\mathcal{G} \times_n \mathbf{A}$	tensor-matrix product
$\mathcal{G} \times \{\mathbf{A}\}$	$\mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)}$
$\mathcal{G} \times_{-n} \{\mathbf{A}\}$	$\mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_{n-1} \mathbf{A}^{(n-1)} \times_{n+1} \mathbf{A}^{(n+1)} \cdots \times_N \mathbf{A}^{(N)}$
$\mathcal{Y} \bar{\times}_n \boldsymbol{a}$	mode-n tensor-vector product
$\mathcal{Y} \bar{\times} \{\boldsymbol{a}\}$	$\mathcal{Y} \bar{\times}_1 \boldsymbol{a}^{(1)} \bar{\times}_2 \boldsymbol{a}^{(2)} \cdots \bar{\times}_N \boldsymbol{a}^{(N)}$
$\langle \mathcal{A}, \mathcal{B} \rangle_{1,\ldots,M;1,\ldots,M}$	contraction between two tensors $\mathcal{A}, \mathcal{B}$
$\langle \mathcal{A}, \mathcal{B} \rangle_{-n}$	contraction between two tensors $\mathcal{A}$ and $\mathcal{B}$ along all modes except mode-n
$\langle \mathcal{A}, \mathcal{B} \rangle$	inner product of two tensors
$\texttt{blkdiag}\left(\mathbf{U}^{(n)}\right)_{n=1}^{N}$	$\bigoplus_{n=1}^{N}\mathbf{U}^{(n)}$ block diagonal matrix

Abbreviations

ALS	Alternating Least Squares
HALS	Hierarchical Alternating Least Squares
QALS	ALS algorithm based on NQP method
rK-QALS	QALS algorithm updates $1 \leqslant K \leqslant R$ components
pQALS	QALS algorithm updates rows of $\mathbf{A}^{(n)}$
ELS	Enhanced or Exact Line Search
dGN	damped Gauss-Newton algorithm
LM	Levenberg-Marquardt iteration
OPT	Optimization algorithm for CP
HOSVD	Higher Order SVD
MPCA	multilinear PCA
HOOI	Higher Order Orthogonality Iteration
HODA	Higher Order Discriminant Analysis
MDA	multilinear discriminant analysis
LDA	linear discriminant analysis
ICA	Independent Component Analysis

KL	Kullback Leibler divergence
CUR	CUR decomposition
GEVD	generalized eigenvalue decomposition
SVD	singular value decomposition
NMF	Nonnegative Matrix Factorization
NTD	Nonnegative Tucker Decomposition
NTF	Nonnegative Tensor Factorization
CP	CANDECOMP/PARAFAC
PARAFAC	Parallel Factor Analysis (equivalent to CP)
DEDICOM	Decomposition into Directional Components
gTF	grid (block) tensor factorization
DTA	dynamic tensor analysis
CMOR	complex Morlet wavelets
NQP	Nonnegative Quadratic Programming
ITPC	inter-trial phase coherence
NMI	normalized mutual information
KNN	k-nearest neighbor classifier
SVM	support vector machines
MIMO	Multiple-Input, Multiple-Output
NFEA	Tensor toolbox for Feature Extraction and Applications
CUDA	NVIDIA's parallel computing architecture (GPU)
BCI	Brain computer interface
DS-CDMA	direct sequence code division multiple access
ONMSE	overall normalized mean-square error
MI	motor imagery
ERD	Event-Related Desynchronization
ERS	Event-Related Synchronization
CSP	common spatial pattern method
DPSK	Differential phase shift keying
BPSK	binary phase-shift keying
QPSK	quadrature phase-shift keying
SNR	signal-to-noise ratio
BER	bit error rate
SIR	signal-to-interference ratio

Chapter 1

Introduction

1.1 Problem Formulation

Data in modern applications such as Brain Computer Interface based on EEG signals often contain multi-modes due to mechanism of data recording, e.g. signals recorded by multiple-sensors (electrodes), in multiple trials, epochs, for multiple subjects and with different tasks, conditions. Moreover, during processing and analysis, dimensionality of the data could be augmented due to expression of the data into sparse domain (time-frequency representation) by different transforms such as Short Time Fourier Transform (STFT), wavelets. That means data itself is naturally a tensor, and has multilinear structures. Standard approaches which analyze such data by considering them as vectors or matrices such as such as PCA/SVD, ICA, NMF, and their variants might be not suitable due to risk of losing the covariance information among various modes. To discover hidden multilinear structures, features within the data, the analysis tools should reflect the multi-dimensional structure of the data[47].

Tensor decompositions and factorizations provide natural representations for multidimensional data by capturing multi-linear and multi-aspect structures in a much lower dimension[25;47;49;106]. Nowadays, tensor have been becoming increasingly important in applications across diverse disciplines, especially signal processing, data mining, feature extraction, classification and multi-way clustering[5;47;106;116]. In chemometrics, Appellof and Davidson[11] studied excitation-emission fluorescence data using the tensor model. Anderson and Bro developed the Nway toolbox[10], and provided excellent examples and data. Vasilescu and Terzopoulos[119] proposed the multilinear projection and Tensorfaces for face recognition in which faces are modeled with multiple factors relating to scene structure (i.e., the location and shapes of visible objects), illumination (i.e., the location and types of light sources), and imaging (i.e., viewpoint, viewing direction and camera characteristics). Sidiropoulos *et. al.*[186] established the model of received signals in direct sequence code division multiple access (DS-CDMA) system as a three-way diversity tensor with modes: antenna, symbol and chip. Later, Sidiropoulos *et. al.*[186] linked parallel factor analysis to multiple-invariance sensor array processing. Yu and Petropulu[214] estimated MIMO system responses using the fourth-order statistics normally generated 5-D tensors. Mørup[131] analyzed multi-channel EEG and MEG data in multiple modes such as frequency, time, space, trials and subjects. Tensor decompositions and multi-way analysis allow naturally to extract hidden (latent) components and to investigate complex relationships among them, for example, in

exploration of social networks[5;15;105;164]. Phan[150] developed the tensor toolbox for feature extraction and applications (NFEA), and provided demonstrations for BCI.

One of the most common tensor decompositions is the Tucker decomposition which was first introduced by Tucker in 1963[206]. Tucker decomposition is an interactive model in which core tensor provides relationship between two components via the Joint Rate index[153]. A particular case of the constrained Tucker decomposition is the DEDICOM (DEcomposition into DIrectional COMponents) model[79;80] which allows to analyze asymmetric relationships between groups (objects). In this direction, Harshman and Lundy[80] analyzed asymmetric measures of yearly trade (import-export) among a set of nations over a period of 10 years. Lundy *et al.*[118] presented an application of three-way DEDICOM to skew-symmetric data for paired preference ratings of treatments for chronic back pain (with additional constraints to obtain meaningful results). Bader *et al.* analyzed email communications of Enron company using the three-way DEDICOM model[13;14].

PARAFAC or canonical polyadic decomposition (CP)[33;34;81;82;83], one of the most popular tensor factorizations, allows approximating multiway data by rank-one tensors in model reduction. CP can also be considered as a particular variation of the Tucker decomposition. Recent years have seen a surge of interest in nonnegative and sparse matrix/tensor factorization and decompositions (NTF and NTD) which provide physically meaningful latent (hidden) components or features with physical or physiological meaning and interpretations[43;47;113]. Nonnegative matrix/tensor factorization or decompositions are emerging techniques for data mining, dimensionality reduction, pattern recognition, object detection, classification, gene clustering, sparse nonnegative representation and coding, and blind source separation (BSS)[19;47;84;86;128;129;157;184;188]. For example, NMF/NTF have already found a wide spectrum of applications in positron emission tomography (PET), spectroscopy, chemometrics and environmental science where the matrices have clear physical meanings and some normalization or constraints are imposed on them[19;177;188].

The main objective of this thesis is to propose algorithms for CP and Tucker decompositions and present applications based on tensor decompositions. The thesis is divided into 7 major chapters. Notation and basic models are introduced in Chapter 1. Proposed algorithms are represented in Chapters 2, 3, 4, 5, and simulations to verify them are in Chapter 6. Chapter 7 is particularly devoted to studying in feature extraction for multiway data.

Chapter 2 introduces the ALS algorithms for tensor decompositions with/without constraints. The "work-horse" Alternating Least Squares (ALS) algorithm has been experimentally proved to work very well on general data[24;187]. However, this algorithm can be relatively slow when data are nearly (multi)collinear or huge. The combination of ALS and line search algorithms[25;50;51] can improve the performance, but they still demand considerable iterations before convergence for some collinear data. For NTF and NTD, the ALS with a rectifier might not work for sparse data. This chapter will present

an efficient variation of the ALS algorithm with low computational cost. The proposed algorithms convert the ALS algorithm for rank-R update to sequential rank-1 updates. That is we establish the ALS update rule for each component of factor. The learning rule does not require matrix inverse, and hence reduces computational cost. Moreover, this would be more stable for ill-conditioned problems and ensure relatively lower complexity even for large -scale problems.

Chapter 3 presents a robust algorithm for nonnegative CP and Tucker decompositions. The "work-horse" ALS algorithm can be adapted for use in nonnegative tensor decompositions including CP and Tucker models by combining with the rectifier $[x]_+ = \max(x, 0)$. However, the modified algorithms can be relatively slow for nearly collinear and sparse data[7;78;109;210]. To improve performance and increase convergence speed, several simple techniques were proposed for the ALS algorithm, such as the weighted ALS incorporating a weighting matrix into the cost function[47], the line-search[140;171], or regularization terms controlling sparsity and orthogonality[41;78;210;215]. Generally, these techniques help ALS cope with collinear and/or sparse factors. However, some other problems arise from controlling regularization parameters. The fact is that the ALS with a simple rectifier might not be an appropriate algorithm for NTF. To this end, we propose a recursive method for solving the nonnegative quadratic programming (NQP). CP and Tucker decompositions can be formulated as NQP problems. Novel ALS algorithms for nonnegative CP and Tucker decompositions are then proposed based on solving NQP problems. Moreover, they can be run on low memory machine or on parallel computing system.

Chapter 4 introduces all-at-once algorithms for tensor decompositions based on the Gauss-Newton (GN) iteration. All-at-once or simultaneous update algorithms are expected to cope with the problem of collinear data[4;142;143;144;167;168;196;202;203]. Nevertheless, those algorithms are computationally demanding due to construction of gradients and Hessians with respect to all the entries of the factors. Moreover, in practical experiments, some algorithms still face problems involving large-scale Jacobians and large-scale inverses of the Hessians. The chapter presents efficient ways to construct the Hessian. Especially, for low-rank approximation, we introduce fast dGN algorithms without building up huge Hessians, and also with elimination of Kronecker products which are often used in CP algorithms. A suite of low-cost algorithms to factorize complex-valued tensors based on damped Gauss-Newton (dGN or LM) iterations with convenient computations for the approximate Hessian and gradients. The proposed algorithms are verified to overwhelmingly outperform "state-of-the-art" algorithms for difficult benchmarks with bottlenecks, swamps for both real and complex-valued tensors.

Chapter 5 presents approaches for tensor factorization suitable for large-scale problems and fast parallel implementation based on grid (or block) tensor decomposition (gTF). The proposed model and algorithms solve till now intractable problem for arbitrary high dimension and large-scale tensors. We factorize all the sub-tensors (block) independently by using efficient algorithms in parallel mode and

next integrate partial results for the whole tensor to estimate desired factors. For practical application such as classification of multiway data, the training data is often augmented with some new samples, hence the basis factors of the feature subspace for the expanded training tensor need to be updated. A simple way is to factorize the new whole training tensor again. However, this approach demands high computational cost. A convenient way is that we update the old bases with factors of the new coming data. A simple low complexity formula for stopping criteria is proposed to the grid tensor factorizations.

Chapter 6 verifies and compares proposed algorithms with well-known algorithms in a variety of benchmarks including real-world data or random tensors with dense or sparse factors, or factors with bias, factors with collinear components. For some experiments, factors are constructed from structured matrices such as Hilbert matrix, Cauchy matrix, Lotkin matrix. Real-world applications are also introduced in this chapter to confirm our algorithms and reveal their abilities in variety of applications such as EEG analysis, clustering, face clustering and classification, BSS in CDMA systems or convolutive MIMO systems.

Chapter 7 proposes a suite of model and algorithms for feature extraction and classification, especially suitable for large scale problems. In our approach, we first decompose multi-way data under the Tucker decomposition with/without constraints to retrieve basis factors and significant features from core tensors. In addition, by revisiting the Tucker decomposition, we have developed family of algorithms based on Higher Order Discriminant Analysis (HODA). The chapter also presents a wide range of applications including object classification, hand-written digit classification, scenes recognition, BCI based on EEG motor imagery signals.

1.2 Tensor Notations and Multilinear Algebra Basics

A tensor is a multi-way array of data; for example a vector is a 1-way tensor and a matrix is 2-way tensor. In general, we shall denote a tensor by bold calligraphic letters, e.g., $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, matrices by bold capital letters, e.g. $\mathbf{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_R] \in \mathbb{R}^{I \times R}$, and vectors by bold italic letters, e.g. $\boldsymbol{a}_j$ or $\boldsymbol{I} = [I_1, I_2, \ldots, I_N]$. For example, for a three-dimensional tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I \times J \times K}$, its frontal slice, lateral slice, and horizontal slice are denoted respectively by $\mathbf{Y}_k = \mathbf{Y}_{::k}, \mathbf{Y}_{:j:}$, and $\mathbf{Y}_{i::}$.

Definition 1.1. Tube *A tube (vector) at a position (i, j) along the mode-3 is denoted by $\boldsymbol{y}_{ij:}$, and the corresponding tubes along the mode-2 and mode-1 are $\boldsymbol{y}_{i:k}$ and $\boldsymbol{y}_{:jk}$[47]. For an N-D tensor $\boldsymbol{\mathcal{Y}}$, a tube at a position $(i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N)$ along mode-n is an I_n vector*

$$\boldsymbol{\mathcal{A}}(i_1, \ldots, i_{n-1}, :, i_{n+1}, \ldots, i_N) = \begin{bmatrix} \boldsymbol{\mathcal{A}}(i_1, \ldots, i_{n-1}, 1, i_{n+1}, \ldots, i_N) \\ \vdots \\ \boldsymbol{\mathcal{A}}(i_1, \ldots, i_{n-1}, I_n, i_{n+1}, \ldots, i_N) \end{bmatrix}. \tag{1.1}$$

Definition 1.2. Outer product *The outer product of the tensors* $\mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ *and* $\mathcal{X} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_M}$ *is given by* $\mathcal{Z} = \mathcal{Y} \circ \mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N \times J_1 \times J_2 \times \cdots \times J_M}$,

$$z_{i_1,i_2,\ldots,i_N,j_1,j_2,\ldots,j_M} = y_{i_1,i_2,\ldots,i_N}\, x_{j_1,j_2,\ldots,j_M}. \tag{1.2}$$

Observe that, the tensor $\mathcal{Z}$ contains all the possible combinations of pair-wise products between the elements of $\mathcal{Y}$ and $\mathcal{X}$. As special cases, the outer product of two vectors $\boldsymbol{a} \in \mathbb{R}^I$ and $\boldsymbol{b} \in \mathbb{R}^J$ yields a rank-one matrix $\mathbf{A} = \boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}^T \in \mathbb{R}^{I \times J}$, and the outer product of three vectors: $\boldsymbol{a} \in \mathbb{R}^I$, $\boldsymbol{b} \in \mathbb{R}^J$ and $\boldsymbol{c} \in \mathbb{R}^Q$ yields a third-order rank-one tensor$\mathcal{Z} = \boldsymbol{a} \circ \boldsymbol{b} \circ \boldsymbol{c} \in \mathbb{R}^{I \times J \times Q}$, $z_{ijq} = a_i\, b_j\, c_q$.

Definition 1.3. (Kronecker product) *The Kronecker product of two matrices* $\mathbf{A} \in \mathbb{R}^{I \times J}$ *and* $\mathbf{B} \in \mathbb{R}^{T \times R}$ *is a matrix denoted as* $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{IT \times JR}$ *and defined as (see the MATLAB function* `kron`*):*

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\,\mathbf{B} & a_{12}\,\mathbf{B} & \cdots & a_{1J}\,\mathbf{B} \\ a_{21}\,\mathbf{B} & a_{22}\,\mathbf{B} & \cdots & a_{2J}\,\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\,\mathbf{B} & a_{I2}\,\mathbf{B} & \cdots & a_{IJ}\,\mathbf{B} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_1 \otimes \boldsymbol{b}_1 & \boldsymbol{a}_1 \otimes \boldsymbol{b}_2 & \cdots & \boldsymbol{a}_J \otimes \boldsymbol{b}_{R-1} & \boldsymbol{a}_J \otimes \boldsymbol{b}_R \end{bmatrix}.$$

It should be mentioned that, in general, the outer product of vectors yields a tensor whereas the Kronecker product gives a vector. For example, for the three vectors $\boldsymbol{a} \in \mathbb{R}^I$, $\boldsymbol{b} \in \mathbb{R}^T$, $\boldsymbol{c} \in \mathbb{R}^Q$ their three-way outer product $\mathcal{Y} = \boldsymbol{a} \circ \boldsymbol{b} \circ \boldsymbol{c} \in \mathbb{R}^{I \times T \times Q}$ is a third-order tensor with the entries $y_{itq} = a_i b_t c_q$, while the three-way Kronecker product of the same vectors is a vector $\text{vec}(\mathcal{Y}) = \boldsymbol{c} \otimes \boldsymbol{b} \otimes \boldsymbol{a} \in \mathbb{R}^{ITQ}$.

Notation 1.1. (Kronecker product of matrices) *Given set of N matrices* $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$*, the following notation denotes Kronecker products among them*

$$\begin{aligned} \{\mathbf{A}\}^{\otimes} &= \bigotimes_{n=1}^{N} \mathbf{A}^{(n)} = \mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n)} \otimes \cdots \otimes \mathbf{A}^{(1)}, \\ \{\mathbf{A}\}^{\otimes_{-n}} &= \bigotimes_{k \neq n} \mathbf{A}^{(k)} = \mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \otimes \cdots \otimes \mathbf{A}^{(1)}. \end{aligned}$$

Definition 1.4. (Hadamard product) *The Hadamard product of two equal-size matrices is the element-wise product denoted by* $\circledast$ *(or* $.*$ *for MATLAB notation) and defined as*

$$\mathbf{A} \circledast \mathbf{B} = \begin{bmatrix} a_{11}\, b_{11} & a_{12}\, b_{12} & \cdots & a_{1J}\, b_{1J} \\ a_{21}\, b_{21} & a_{22}\, b_{22} & \cdots & a_{2J}\, b_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\, b_{I1} & a_{I2}\, b_{I2} & \cdots & a_{IJ}\, b_{IJ} \end{bmatrix}. \tag{1.3}$$

Notation 1.2. (Hadamard product of matrices) *Given set of N matrices* $\mathbf{A}^{(n)} \in \mathbb{R}^{I \times R}$*, the following notation denotes Hadamard products among them*

$$\begin{aligned} \{\mathbf{A}\}^{\circledast} &= \circledast_{n=1}^{N} \mathbf{A}^{(n)} = \mathbf{A}^{(N)} \circledast \cdots \circledast \mathbf{A}^{(n)} \circledast \cdots \circledast \mathbf{A}^{(1)}, \\ \{\mathbf{A}\}^{\circledast_{-n}} &= \circledast_{k \neq n} \mathbf{A}^{(k)} = \mathbf{A}^{(N)} \circledast \cdots \circledast \mathbf{A}^{(n+1)} \circledast \mathbf{A}^{(n-1)} \circledast \cdots \circledast \mathbf{A}^{(1)}. \end{aligned}$$

Definition 1.5. (Khatri-Rao product) *For two matrices* $\mathbf{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_J] \in \mathbb{R}^{I \times J}$ *and* $\mathbf{B} = [\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_J] \in \mathbb{R}^{T \times J}$*, their Khatri-Rao product, denoted by* $\odot$*, performs the following operation:*

$$\mathbf{A} \odot \mathbf{B} = [\boldsymbol{a}_1 \otimes \boldsymbol{b}_1 \;\; \boldsymbol{a}_2 \otimes \boldsymbol{b}_2 \;\; \cdots \;\; \boldsymbol{a}_J \otimes \boldsymbol{b}_J] \tag{1.4}$$

$$= \left[vec(\boldsymbol{b}_1 \boldsymbol{a}_1^T) \;\; vec(\boldsymbol{b}_2 \boldsymbol{a}_2^T) \;\; \cdots \;\; vec(\boldsymbol{b}_J \boldsymbol{a}_J^T) \right] \in \mathbb{R}^{IT \times J}. \tag{1.5}$$

The following properties of the Khatri-Rao product are often employed :

$$(\mathbf{A} \odot \mathbf{B})^T \ (\mathbf{A} \odot \mathbf{B}) = \mathbf{A}^T\mathbf{A} \circledast \mathbf{B}^T\mathbf{B}, \tag{1.6}$$
$$(\mathbf{A} \odot \mathbf{B})^\dagger = [(\mathbf{A}^T\mathbf{A}) \circledast (\mathbf{B}^T\mathbf{B})]^{-1}(\mathbf{A} \odot \mathbf{B})^T . \tag{1.7}$$

These properties also hold true for complex-valued matrices $\mathbf{A} \in \mathbb{C}^{I\times J}$, and $\mathbf{B} \in \mathbb{C}^{T\times J}$. In this case, the transpose operators are replaced by the Hermitian transposes.

Notation 1.3. (Khatri-Rao product of matrices) *Given set of N matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n\times R}$, the following notation denotes Khatri-Rao products among them*

$$\begin{aligned}\{\mathbf{A}\}^{\odot} &= \bigodot_{n=1}^{N}\mathbf{A}^{(n)} = \mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n)} \odot \cdots \odot \mathbf{A}^{(1)},\\ \{\mathbf{A}\}^{\odot_{-n}} &= \bigodot_{k\neq n}\mathbf{A}^{(k)} = \mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)}.\end{aligned}$$

Corollary 1.1.

$$\begin{aligned}\{\mathbf{A}\}^{\odot T}\{\mathbf{A}\}^{\odot} &= \{\mathbf{A}^T\mathbf{A}\}^{\circledast} = \circledast_{n=1}^{N}\mathbf{A}^{(n)T}\mathbf{A}^{(n)},\\ \{\mathbf{A}\}^{\odot_{-n}T}\{\mathbf{A}\}^{\odot_{-n}} &= \{\mathbf{A}^T\mathbf{A}\}^{\circledast_{-n}} = \circledast_{k\neq n}\mathbf{A}^{(k)T}\mathbf{A}^{(k)},\\ \{\mathbf{A}\}^{\otimes T}\{\mathbf{A}\}^{\otimes} &= \{\mathbf{A}^T\mathbf{A}\}^{\otimes} = \bigotimes_{n\neq 1}^{N}\mathbf{A}^{(n)T}\mathbf{A}^{(n)},\\ \{\mathbf{A}\}^{\otimes_{-n}T}\{\mathbf{A}\}^{\otimes_{-n}} &= \{\mathbf{A}^T\mathbf{A}\}^{\otimes_{-n}} = \bigotimes_{k\neq n}\mathbf{A}^{(k)T}\mathbf{A}^{(k)}.\end{aligned}$$

Definition 1.6. (vectorization) *Vectorization of an N-D tensor $\mathcal{A} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ is to map an entry $a_{\boldsymbol{i}} = \mathcal{A}(i_1, i_2, \ldots, i_N)$ to an entry $\mathfrak{a}(\text{ivec}(\boldsymbol{i}, \boldsymbol{I}))$ of vector $\mathfrak{a}$, i.e.,*

$$\text{ivec}(\boldsymbol{i}, \boldsymbol{I}) = i_1 + (i_2 - 1)I_1 + (i_3 - 1)I_1I_2 + \cdots + (i_N - 1)I_1 \cdots I_{N-1}. \tag{1.8}$$

Definition 1.7. (unfolding or matricization of tensor) *Mode-n unfolding of an N-D tensor $\mathcal{A} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ is to horizontally concatenate all tubes of $\mathcal{A}$ along mode-n to establish an $I_n \times (\prod_{m\neq n} I_m)$ matrix. The mode-n unfolding can be expressed via vectorization of tensors as follows*

$$\mathbf{A}_{(n)} = \left[\ \text{vec}\left(\mathcal{A}^{(1,n)}\right) \quad \cdots \quad \text{vec}\left(\mathcal{A}^{(I_n,n)}\right)\ \right]^T, \tag{1.9}$$

where $\mathcal{A}^{(i_n,n)}$ is an $(N-1)$-order subtensor of $\mathcal{A}$ whose the n-th index is fixed to i_n

$$\mathcal{A}^{(i_n,n)}(i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N) = \mathcal{A}(i_1, \ldots, i_n, \ldots, i_N).$$

Notation 1.4. (mode-*n* tensor-matrix product) *The product of a tensor and a matrix along the mode-n is denoted as*

$$\mathcal{Y} = \mathcal{G} \times_n \mathbf{A}, \quad \textit{or} \quad \mathbf{Y}_{(n)} = \mathbf{A}\,\mathbf{G}_{(n)}. \tag{1.10}$$

Multiplication in all possible modes $(n = 1, 2, \ldots, N)$ of a tensor $\mathcal{G}$ and a set of matrices $\mathbf{A}^{(n)}$ is denoted as

$$\mathcal{G} \times \{\mathbf{A}\} = \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)}, \tag{1.11}$$
$$[\mathcal{G} \times \{\mathbf{A}\}]_{(n)} = \mathbf{A}^{(n)}\mathbf{G}_{(n)}\left[\mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \otimes \cdots \otimes \mathbf{A}^{(1)}\right]^T. \tag{1.12}$$

Multiplication of a tensor with all except one or two modes is denoted as

$$\mathcal{G} \times_{-n} \{\mathbf{A}\} = \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_{n-1} \mathbf{A}^{(n-1)} \times_{n+1} \mathbf{A}^{(n+1)} \cdots \times_N \mathbf{A}^{(N)} .$$

$$\mathcal{G} \times_{-(n,m)} \{\mathbf{A}\} = \mathcal{G} \times_1 \mathbf{A}^{(1)} \cdots \times_{n-1} \mathbf{A}^{(n-1)} \times_{n+1} \mathbf{A}^{(n+1)} \cdots \times_{m-1} \mathbf{A}^{(m-1)} \times_{m+1} \mathbf{A}^{(m+1)} \cdots \times_N \mathbf{A}^{(N)} .$$

Notation 1.5. (mode-*n* tensor-vector product) *The mode-n multiplication of a tensor* $\mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ *by a vector* $\boldsymbol{a} \in \mathbb{R}^{I_n}$ *is denoted by*

$$\mathcal{Z} = \mathcal{Y} \,\bar{\times}_n\, \boldsymbol{a} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N}, \tag{1.13}$$

and the tensor-vector product of a tensor $\mathcal{Y}$ *with a set of N column vectors* $\{\boldsymbol{a}\} = \{\boldsymbol{a}^{(1)}, \boldsymbol{a}^{(2)}, \ldots, \boldsymbol{a}^{(N)}\}$ *is given by*

$$\mathcal{Y} \,\bar{\times}\, \{\boldsymbol{a}\} = \mathcal{Y} \,\bar{\times}_1\, \boldsymbol{a}^{(1)} \,\bar{\times}_2\, \boldsymbol{a}^{(2)} \cdots \bar{\times}_N\, \boldsymbol{a}^{(N)} . \tag{1.14}$$

Definition 1.8. (contraction between two tensors) *The contracted product of* $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_M \times J_1 \times \cdots \times J_N}$ *and* $\mathcal{B} \in \mathbb{R}^{I_1 \times \cdots \times I_M \times K_1 \times \cdots \times K_P}$ *along the first M modes is a tensor of size* $J_1 \times \cdots \times J_N \times K_1 \times \cdots \times K_P$*, given by*

$$\langle \mathcal{A}, \mathcal{B} \rangle_{1,\ldots,M;1,\ldots,M}(j_1, \ldots, j_N, k_1, \ldots, k_P) = \sum_{i_1=1}^{I_1} \cdots \sum_{i_M=1}^{I_M} a_{i_1,\ldots,i_M,j_1,\ldots,j_N}\, b_{i_1,\ldots,i_M,k_1,\ldots,k_P}. \tag{1.15}$$

The remaining modes are ordered such that those from $\mathcal{A}$ come before $\mathcal{B}$[106]. The arguments specifying the modes of $\mathcal{A}$ and those of $\mathcal{B}$ for contraction need not be consecutive. However, the sizes of the corresponding dimensions must be equal[106]. For example, the contracted tensor product along the mode-2 of a tensor $\mathcal{A} \in \mathbb{R}^{3\times4\times5}$, and the mode-3 of a tensor $\mathcal{B} \in \mathbb{R}^{7\times8\times4}$ returns a tensor $\mathcal{C} = \langle \mathcal{A}, \mathcal{B} \rangle_{2;3} \in \mathbb{R}^{3\times5\times7\times8}$.

The contracted tensor product of $\mathcal{A}$ and $\mathcal{B}$ along the same M modes simplifies to

$$\langle \mathcal{A}, \mathcal{B} \rangle_{1,\ldots,M;1,\ldots,M} = \langle \mathcal{A}, \mathcal{B} \rangle_{1,\ldots,M}, \tag{1.16}$$

whereas the contracted product of tensors $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\mathcal{B} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$ along all modes except the mode-*n* is denoted as

$$\langle \mathcal{A}, \mathcal{B} \rangle_{-n} = \mathbf{A}_{(n)}\, \mathbf{B}_{(n)}^T \in \mathbb{R}^{I_n \times J_n}, \qquad (I_k = J_k, \;\; \forall k \neq n). \tag{1.17}$$

In a special case of $M = 0$, the contracted product becomes the outer product of two tensors.

The contracted product of two three-way tensors $\mathcal{A} \in \mathbb{R}^{I\times J\times K}$ and $\mathcal{B} \in \mathbb{R}^{P\times Q\times R}$ along the mode-1 returns a four-way tensor defined as

$$\mathcal{C} \;=\; \langle \mathcal{A}, \mathcal{B} \rangle_1 \;\in \mathbb{R}^{J\times K\times Q\times R}, \quad c_{jkqr} = \sum_i a_{ijk} b_{iqr}, \quad (I = P),$$

and the contracted product along the two modes returns a matrix, for example:

$$\mathbf{F} \;=\; \langle \mathcal{A}, \mathcal{B} \rangle_{1,2} = \langle \mathcal{A}, \mathcal{B} \rangle_{-3} \;\in \mathbb{R}^{K\times R}, \quad f_{kr} = \sum_{i,j} a_{ijk} b_{ijr}, \quad (I = P, J = Q)$$

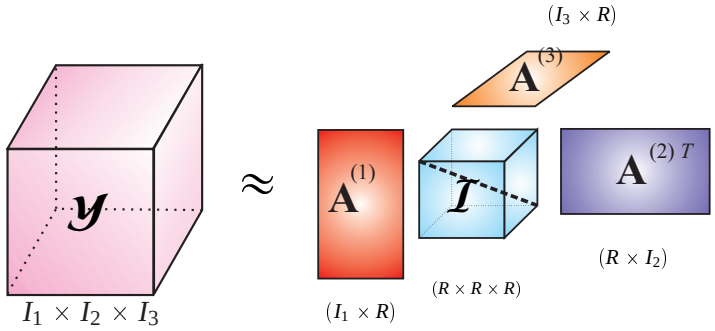


Figure 1.1: Illustration of a 3-D tensor factorization. The $I_1 \times I_2 \times I_3$ dimensional tensor $\mathcal{Y}$ is explained by the three factors $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}$ along the three modes. Factors consist of the same number of components R.

which can be expressed in a matrix multiplication form as $\mathbf{F} = \mathbf{A}_{(3)}\,\mathbf{B}_{(3)}^{T}$. For two tensors of the same dimension, the contracted product along all their modes is their inner product

$$\langle \mathcal{A}, \mathcal{B} \rangle_{1,\ldots,N} \quad = \quad \langle \mathcal{A}, \mathcal{B} \rangle. \tag{1.18}$$

Definition 1.9. (partitioned matrix) *A partitioned matrix* $\mathbf{U}$ *of* N *matrices* $\mathbf{U}^{(n)}$ *along the mode-2 (horizontal) is denoted by*

$$\mathbf{U} = \left[\mathbf{U}^{(1)} \;\cdots\; \mathbf{U}^{(n)} \;\cdots\; \mathbf{U}^{(N)}\right] = \left[\mathbf{U}^{(n)}\right]_{n=1}^{N}, \tag{1.19}$$

and a partitioned matrix $\mathbf{V}$ *of* NM *matrices* $\mathbf{V}^{(n,m)}$ *along two modes is denoted by* $\mathbf{V} = \left[\mathbf{V}^{(n,m)}\right]_{n=1,m=1}^{N,M}$.

Definition 1.10. (block diagonal matrix or direct sum) *A block diagonal matrix* $\mathbf{B}$ *of* N *matrices* $\mathbf{U}^{(n)}$ *is denoted by*

$$\mathbf{B} \quad = \quad \begin{bmatrix} \mathbf{U}^{(1)} & & \\ & \ddots & \\ & & \mathbf{U}^{(N)} \end{bmatrix} = \mathtt{blkdiag}\left(\mathbf{U}^{(n)}\right)_{n=1}^{N} = \bigoplus_{n=1}^{N} \mathbf{U}^{(n)}. \tag{1.20}$$

1.3 Canonical Polyadic Decomposition

Definition 1.11. (Canonical polyadic decomposition or PARAFAC (CP)) *"Factorize a given* N*-th order data tensor* $\mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ *into a set of* N *factors matrices* $\mathbf{A}^{(n)} = [\boldsymbol{a}_1^{(n)}, \boldsymbol{a}_2^{(n)}, \ldots, \boldsymbol{a}_R^{(n)}] \in \mathbb{R}^{I_n \times R}$, $(n = 1, 2, \ldots, N)$ *representing the common (loading) factors"*[33;81;89]*, that is,*

$$\mathcal{Y} \quad \approx \quad \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \circ \boldsymbol{a}_r^{(2)} \circ \ldots \circ \boldsymbol{a}_r^{(N)} = \hat{\mathcal{Y}}, \tag{1.21}$$

where symbol "$\circ$" means outer product, and we assume unit-length components $\|\boldsymbol{a}_r^{(n)}\|_2 = 1$ *for* $n = 1, 2, \ldots, N-1$, $r = 1, 2, \ldots, R$ *(see Figure 1.1). Tensor* $\hat{\mathcal{Y}}$ *is an approximation of the data tensor* $\mathcal{Y}$.

The CP model with nonnegative factors $\mathbf{A}^{(n)}$ is also known as Nonnegative Tensor Factorization (NTF) which is an extension model of Nonnegative Matrix Factorization (NMF). Alternatively we can describe the CP model using tensor notation given by[106]

$$\mathcal{Y} \approx \mathcal{I} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)}, \tag{1.22}$$

where $\mathcal{I}$ is an identity tensor. The mode-n matricization $\mathbf{Y}_{(n)}, n = 1, 2, \ldots, N$ can be represented by set of matrix factorizations:

$$\mathbf{Y}_{(n)} \approx \mathbf{A}^{(n)} \left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \cdots \odot \mathbf{A}^{(1)}\right)^T = \mathbf{A}^{(n)} \{\mathbf{A}\}^{\odot -n T}, \tag{1.23}$$

or as a summation of rank-one approximations

$$\mathbf{Y}_{(n)} \approx \sum_{r=1}^{R} \boldsymbol{a}_r^{(n)} \left(\boldsymbol{a}_r^{(N)} \otimes \cdots \otimes \boldsymbol{a}_r^{(n+1)} \otimes \boldsymbol{a}_r^{(n-1)} \otimes \cdots \otimes \boldsymbol{a}_r^{(1)}\right)^T = \sum_{r=1}^{R} \boldsymbol{a}_r^{(n)} \left(\boldsymbol{a}_r^{\otimes -n}\right)^T. \tag{1.24}$$

We denote concatenation of vectorizations of factors $\mathbf{A}^{(n)}$ by

$$\mathfrak{a} = \left[\text{vec}\left(\mathbf{A}^{(1)}\right)^T \cdots \text{vec}\left(\mathbf{A}^{(N)}\right)^T\right]^T = \left[\boldsymbol{a}_1^{(1)T} \cdots \boldsymbol{a}_R^{(1)T} \cdots \boldsymbol{a}_1^{(N)T} \cdots \boldsymbol{a}_R^{(N)T}\right]^T. \tag{1.25}$$

1.3.1 Basic Statistics for a Synthetic Tensor

This section presents some basic statistics for a synthetic CP tensor $\mathcal{Y}$. In simulation, we often build up dense and large-scale tensors degraded by an additive Gaussian noise. To generate random noise, we need the standard deviation $\text{vec}(\mathcal{Y})$. The following lemma helps us to quickly compute this statistic instead of manipulating on samples $y_{i_1 \ldots i_N}$.

Lemma 1.1. *The following properties hold for an N-dimensional tensor* $\mathcal{Y} = \mathcal{I} \times_1 \mathbf{A}^{(1)} \cdots \times_N \mathbf{A}^{(N)}$.

1. $\sum_{\boldsymbol{i}=\boldsymbol{1}}^{\boldsymbol{I}} y_{\boldsymbol{i}} = \left(\{\mathbf{1}^T \mathbf{A}\}^{\circledast}\right) \mathbf{1}$, *where* $\boldsymbol{i} = [i_1, i_2, \ldots, i_N], \boldsymbol{I} = [I_1, I_2, \ldots, I_N]$.
2. $\sum_{\boldsymbol{i}=\boldsymbol{1}}^{\boldsymbol{I}} y_{\boldsymbol{i}}^2 = \mathbf{1}^T \left(\{\mathbf{A}^T \mathbf{A}\}^{\circledast}\right) \mathbf{1}$.

The multi-index summation is defined for an index vector $\boldsymbol{K} = [K_1, K_2, \ldots, K_N]$ *as*

$$\sum_{\boldsymbol{k}=\boldsymbol{1}}^{\boldsymbol{K}} \equiv \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} \cdots \sum_{k_N=1}^{K_N}. \tag{1.26}$$

Proof. Summation of all the entries of the tensor $\mathcal{Y}$ is given by

$$\begin{aligned} \sum_{\boldsymbol{i}=\boldsymbol{1}}^{\boldsymbol{I}} y_{\boldsymbol{i}} &= \mathbf{1}^T \text{vec}(\mathcal{Y}) = \left(\mathbf{1}_{I_N} \odot \mathbf{1}_{I_{N-1}} \odot \cdots \odot \mathbf{1}_{I_1}\right)^T \left(\mathbf{A}^{(N)} \odot \mathbf{A}^{(N-1)} \odot \cdots \odot \mathbf{A}^{(1)}\right) \mathbf{1} \\ &= \left(\left(\mathbf{1}^T \mathbf{A}^{(N)}\right) \circledast \left(\mathbf{1}^T \mathbf{A}^{(N-1)}\right) \circledast \cdots \circledast \left(\mathbf{1}^T \mathbf{A}^{(1)}\right)\right) \mathbf{1}. \end{aligned}$$

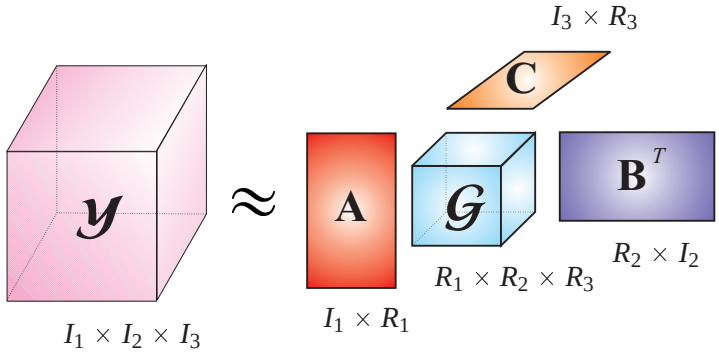


Figure 1.2: Illustration for a 3-way Tucker decomposition; the objective here is to find factors (component matrices) $\mathbf{A}^{(n)} = [\boldsymbol{a}_1^{(n)}, \boldsymbol{a}_2^{(n)}, \ldots, \boldsymbol{a}_{R_n}^{(n)}] \in \mathbb{R}^{I_n \times R_n}$ $(n = 1, 2, 3)$ and a core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$, typically $R_n \ll I_n$.

Frobenius norm of the tensor $\mathcal{Y}$ is given by $\sum_{i=1}^{I} y_i^2 = \text{vec}(\mathcal{Y})^T \text{vec}(\mathcal{Y}) = \mathbf{1}^T \{\mathbf{A}\}^{\odot T} \{\mathbf{A}\}^{\odot} \mathbf{1} = \mathbf{1}^T \left\{\mathbf{A}^T \mathbf{A}\right\}^{\circledast} \mathbf{1}$. □

1.4 Tucker Decomposition

Tucker decomposition[106;206], illustrated in Figure 1.2 for a 3-way case, is a basic model for high dimensional tensors which allows effectively to perform model reduction and feature extraction.

Definition 1.12. (Tucker decomposition) *Decomposition of a given N-th order tensor* $\mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}$ *into a product of an unknown core tensor* $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \cdots \times R_N}$ *and a set of N unknown factors (component matrices),* $\mathbf{A}^{(n)} = [\boldsymbol{a}_1^{(n)}, \boldsymbol{a}_2^{(n)}, \ldots, \boldsymbol{a}_{R_n}^{(n)}] \in \mathbb{R}^{I_n \times R_n}$ $(n = 1, 2, \ldots, N)$[206]

$$\begin{aligned} \mathcal{Y} &= \sum_{j_1=1}^{R_1} \sum_{j_2=1}^{R_2} \cdots \sum_{j_N=1}^{R_N} g_{j_1 j_2 \cdots j_N} \left(\boldsymbol{a}_{j_1}^{(1)} \circ \boldsymbol{a}_{j_2}^{(2)} \circ \cdots \circ \boldsymbol{a}_{j_N}^{(N)} \right) + \mathcal{E} \\ &= \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)} + \mathcal{E} = \mathcal{G} \times \{\mathbf{A}\} + \mathcal{E} = \hat{\mathcal{Y}} + \mathcal{E}, \end{aligned} \tag{1.27}$$

where $\hat{\mathcal{Y}}$ *is an approximation of* $\mathcal{Y}$*, and* $\mathcal{E}$ *denotes the approximation error.*

For three-dimensional data the basic Tucker-3 model of a tensor $\mathcal{Y}$ is represented with three factors $\mathbf{A} = \mathbf{A}^{(1)}, \mathbf{B} = \mathbf{A}^{(2)}, \mathbf{C} = \mathbf{A}^{(3)}$ (see Figure 1.2)

$$\mathcal{Y} = \mathcal{G} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}. \tag{1.28}$$

The above Tucker-3 model is readily reduced to the Tucker-2 model by merging one factor with core tensor, for example factor $\mathbf{A}^{(3)} = \mathbf{C}$ and $\mathcal{F} = \mathcal{G} \times_3 \mathbf{C}$ to give: $\mathcal{Y} = \mathcal{F} \times_1 \mathbf{A} \times_2 \mathbf{B}$.

Note that the Tucker decomposition is in general non unique. However, in the special case where the core tensor has nonzero elements only on the superdiagonal, the Tucker model is reduced uniquely

to the PARAFAC under some mild conditions[81]. To obtain meaningful and unique representation by the Tucker decomposition, orthogonality, sparsity and nonnegativity constraints are often imposed on hidden factors and the core tensor of the Tucker decomposition to obtain meaningful and unique representations[103;131]. For convenience, orthogonal factors are denoted by $\mathbf{U}^{(n)}$, and nonnegative or general factors by $\mathbf{A}^{(n)}$. By imposing nonnegativity constraints for CP, we obtain the NTF (Nonnegative Tensor Factorization) model, while for the Tucker models with the nonnegativity constraints we obtain the so called NTD (Nonnegative Tucker Decomposition) model. NTF and NTD have been found many potential applications in neuroscience, bioinformatics, chemometrics and text mining[47;103;131]. Furthermore, by imposing orthogonality constraints on the factor matrices we obtain a model referred to as the HOSVD (Higher Order Singular Value Decomposition) algorithm or the HOOI (Higher Order Orthogonal Iterations) algorithm, introduced first by Lathauwer *et al.*[58;60].

1.5 Decomposition into Directional Components

DEDICOM (decomposition into directional components) is a family of matrix and tree-way tensor decompositions introduced by Harshman in 1978[81] for matrices and by Kiers in 1993 for 3D tensors[100] and investigated by many researchers, Bader, Kolda, Acar, Takane, Kiers and Sun et al.[106;193]. DEDICOM model is a particular case of the Tucker-2 decomposition, and is described as follows: "Given a three-dimensional data tensor $\mathcal{Y} \in \mathbb{R}^{I \times I \times K}$, the DEDICOM model of this tensor returns a factor matrix $\mathbf{A} \in \mathbb{R}^{I \times J}$ of loadings or weights, a communication pattern matrix $\mathbf{R} \in \mathbb{R}_{+}^{J \times J}$ that captures asymmetric relationships, and a sparse core tensor $\mathcal{D} \in \mathbb{R}^{J \times J \times K}$ with diagonal frontal slices giving the weights of columns of $\mathbf{A}$ for each slice in the third mode (see Figure 1.3)

$$\mathbf{Y}_k = \mathbf{A}\ \mathrm{diag}\{\boldsymbol{d}_k\}\ \mathbf{R}\ \mathrm{diag}\{\boldsymbol{d}_k\}\ \mathbf{A}^T,\ (k = 1, \ldots, K), \tag{1.29}$$

where $\boldsymbol{d}_k$ is the k-th column of matrix $\mathbf{D} \in \mathbb{R}^{J \times K}$. Matrix $\mathbf{D}$ is built up from diagonals of the frontal slices of the core tensor $\mathcal{D}$".

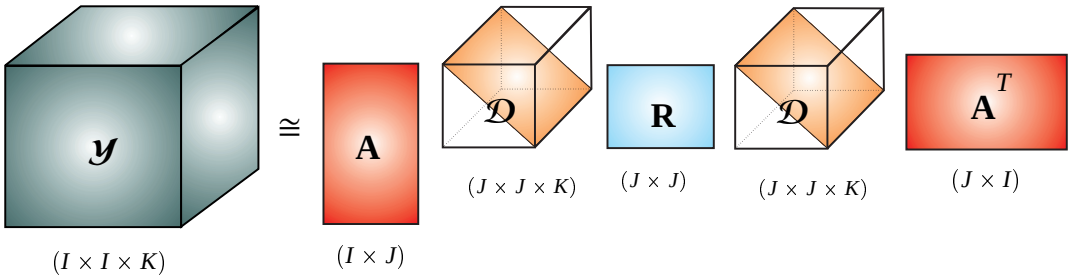
Let $\mathcal{G} \in \mathbb{R}^{J \times J \times K}$ be a core tensor whose frontal slices are represented as

$$\mathbf{G}_k = \mathrm{diag}\{\boldsymbol{d}_k\}\ \mathbf{R}\ \mathrm{diag}\{\boldsymbol{d}_k\}. \tag{1.30}$$

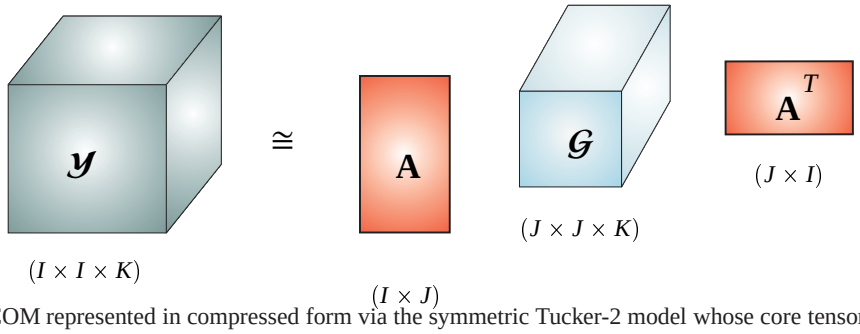
This shows that DEDICOM is related to the particular (constrained) Tucker-2 decomposition

$$\mathcal{Y} \approx \mathcal{G} \times_1 \mathbf{A} \times_2 \mathbf{A}. \tag{1.31}$$

DEDICOM can be employed in extracting some complex relationships in social networks[5;15;105]. Harshman and Lundy[80] analyzed asymmetric measures of yearly trade (import-export) among a set of nations over a period of 10 years. Lundy *et al.*[118] presented an application of three-way DEDICOM to



(a) The three-way DEDICOM model



(b) DEDICOM represented in compressed form via the symmetric Tucker-2 model whose core tensor $\mathcal{G}$ is defined by its frontal slices: $\mathbf{G}_k = \text{diag}\{\boldsymbol{d}_k\}\mathbf{R}\,\text{diag}\{\boldsymbol{d}_k\}$.

Figure 1.3: Illustration of the three-way DEDICOM model of the data tensor $\mathcal{Y} \in \mathbb{R}^{I \times I \times K}$, and its relationship to the symmetric Tucker-2 decomposition.

skew-symmetric data for paired preference ratings of treatments for chronic back pain with additional constraints to get meaningful results. In the context of web link, interesting analysis was performed by Kolda *et al.* in 2005[107] and Kolda and Bader in 2006[105] by combining hyper-links and anchor text information by representing web graph data as a sparse three-way tensor with modes: *web-pages* $\times$ *web-pages* $\times$ *anchor text*. Bader *et al.* analyzed email communications of Enron company using the three-way DEDICOM model[13;14]. Bader, Harshman, and Kolda[15] recently applied their DEDICOM algorithm to email communication graphs over time. In the most social network analysis, the data are nonnegative tensors. For example, the Enron email data can be constructed as a three-way tensor[15;164;170] in which each entry x_{ijk} expresses the number of emails sent from an employee i to an employee j in a month k. Or for the international trade data[80], each entry of the three-way tensor (country $\times$ country $\times$ year) provides the import/export data among nations in a certain year. Nonnegative constrains imposed on matrices **A**, **D** and **R** are necessity due to meaningful interpretation.

CHAPTER 2

Alternating Least Squares Algorithm and Its Variations

2.1 ALS Algorithms for CP and NTF

This chapter presents basic algorithms for CP, which can be straightforwardly applied for NTF. Most algorithms for the (nonnegative) matrix and tensor factorizations are based on minimization of the squared Euclidean distance (Frobenius norm)[7;47;106] used as the global cost function (subject to non-negativity constraints), that is

$$D(\mathcal{Y}||\hat{\mathcal{Y}}) = \frac{1}{2}\|\mathcal{Y} - \hat{\mathcal{Y}}\|_F^2. \tag{2.1}$$

A basic approach to the above formulated optimization problem (2.1) is alternating minimization and projection: the specified cost function is alternately minimized with respect to sets of parameters, each time optimizing one set of arguments while keeping the others fixed. It should be noted that the cost function (2.1) for NTF is convex with respect to entries of $\mathbf{A}^{(n)}$, but not all. Alternating minimization of the cost function (2.1) leads to a nonnegative fixed point ALS algorithm which can be described briefly as follows:

1. Initialize all $\mathbf{A}^{(n)}$ randomly or by using the recursive application of Perron-Frobenius theory to SVD[19;23], or by selected fibers or structures from the data $\mathcal{Y}$[7;212].

2. Estimate $\mathbf{A}^{(n)}$ from the approximation by solving (2.1)

$$\min_{\mathbf{A}^{(n)}} D_F(\mathcal{Y}||\hat{\mathcal{Y}}) = \frac{1}{2}||\mathcal{Y} - \hat{\mathcal{Y}}||_F^2, \text{with fixed}\, \mathbf{A}^{(m)}, m \neq n.$$

3. For nonnegative factors, set all negative elements of $\mathbf{A}^{(n)}$ to zero or a small positive value ε.

4. Estimate other factors until convergence.

The above ALS algorithm can be written in the following explicit form for $\mathbf{A}^{(n)}$ [10;24;33;81;187;188]

$$\mathbf{A}^{(n)} \leftarrow \mathbf{Y}_{(n)} \{\mathbf{A}\}^{\odot -n} \left(\{\mathbf{A}^T\mathbf{A}\}^{\circledast -n}\right)^{\dagger}, \quad (n = 1, 2, \ldots, N), \tag{2.2}$$

where $\mathbf{A}^{\dagger}$ is the Moore-Penrose inverse of $\mathbf{A}$, $\mathbf{Y}_{(n)}$ is the mode-n matricized version of tensor $\mathcal{Y}$, and the Hadamard product $\mathbf{\Gamma}^{(n,n)} = \left\{\mathbf{A}^T\mathbf{A}\right\}^{\circledast -n}$ is given by

$$\mathbf{\Gamma}^{(n,n)} = (\mathbf{A}^{(N)T}\mathbf{A}^{(N)}) \circledast \cdots \circledast (\mathbf{A}^{(n+1)T}\mathbf{A}^{(n+1)}) \circledast (\mathbf{A}^{(n-1)^T}\mathbf{A}^{(n-1)}) \circledast \cdots \circledast (\mathbf{A}^{(1)T}\mathbf{A}^{(1)}). \tag{2.3}$$

A fast implementation of ALS for 3-way tensor[200] reduces the expensive computation of $\mathbf{Y}_{(n)} \bigodot_{k\neq n} \mathbf{A}^{(k)}$. Unfortunately, this algorithm cannot be generalized to higher orders[204]. The ALS algorithm and its variations pointed out in[51] are simple algorithms and can work well for general data[24;187]. For NTF, the ALS algorithm was modified to eliminate all negative entries by a rectifier

$$\mathbf{A}^{(n)} \leftarrow \left[\mathbf{A}^{(n)}_{ALS}\right]_+ = \max\{\varepsilon, \mathbf{A}^{(n)}\}, \tag{2.4}$$

ε is a small constant (typically, 10^{-16}) to enforce positive entries. Note that `max` operator is performed component-wise for entries of matrices. Various additional constraints on $\mathbf{A}^{(n)}$ can be imposed[42].

Computing the Khatri-Rao product $\{\mathbf{A}\}^{\odot -n}$ requires $(N-1)RI^{N-1}$ multiplications (for simplicity, dimensions are assumed to be identical $I_1 = \ldots = I_N = I$). The matrix product $\mathbf{Y}_{(n)}\ \{\mathbf{A}\}^{\odot -n}$ demands RI^N multiplications. Hadamard products $\{\mathbf{A}^T\mathbf{A}\}^{\circledast -n}$ do not consume much multiplications. Hence, the approximate number of multiplications required for the learning rule (2.2) is $RI^N + (N-1)RI^{N-1}$.

This technique was widely applied to employ algorithms for CP as NTF algorithms. However, for practical experiments, ALS (2.4) may not converge to the solution without additional regularization parameters. They may take many iterations to converge. Moreover, they are also not guaranteed to converge to a global minimum or even a stationary point, but only to a solution where the cost functions cease to decrease[19;106]. ALS can face problems when $\mathbf{\Gamma}^{(n,n)}$ is not full column rank, and it is relatively slow for collinear (ill-conditioned) data with swamp, bottle-neck or CP-degeneracies[51]. To improve performance and increase convergence speed, several simple techniques were proposed for the ALS algorithm, such as the weighted ALS incorporating a weighting matrix into the cost function[47], the linesearch extrapolation methods[10;25;81;140;171;171;200], rotation method[145;146], compression[26;101], regularization terms controlling sparsity and orthogonality[41;78;210;215], applying the iterated Tikhonov regularization[133;175], the damped Newton iteration[217], the projected gradient methods[115] or simply adding a small diagonal matrix $\mu\mathbf{I}_R$ into $\mathbf{\Gamma}^{(n,n)}$ [51]

$$\mathbf{A}^{(n)} \leftarrow \mathbf{Y}_{(n)}\ \{\mathbf{A}\}^{\odot -n} \left(\mathbf{\Gamma}^{(n,n)} + \mu\,\mathbf{I}_R\right)^{-1}. \tag{2.5}$$

However, for real-world data, a proper selection of parameter μ influences performance of the final result. Further modifications were discussed in Chapter 4[47].

The fact is that ALS algorithm faces (pseudo-) inverse which can be ill-conditioned for real-world data, and also time consuming to calculate reliably. If $\mathbf{A}^{(n)}$ has only one component $R = 1$, that is the tensor $\mathcal{Y}$ is approximated by a rank-one tensor, the Hadamard product $\left\{\mathbf{A}^T\mathbf{A}\right\}^{\circledast -n}$ in (2.2)

returns a scalar, matrix inverse will become a division by a scalar. Therefore, if we form an ALS update rule for each component $\boldsymbol{a}_r^{(n)}$ of the factor $\mathbf{A}^{(n)}$, this learning rule does not require matrix inverse, and hence reduces computational cost. Moreover, this would be more stable for ill-conditioned problems and ensure relatively lower complexity even for large -scale problems. Based on this idea a new algorithm is proposed in this chapter as follows: instead of estimating the whole factor $\mathbf{A}^{(n)} = [\boldsymbol{a}_1^{(n)}, \boldsymbol{a}_2^{(n)}, \ldots, \boldsymbol{a}_R^{(n)}] \in \mathbb{R}_+^{I_n \times R}$, we sequentially estimate components $\boldsymbol{a}_r^{(n)}$ [37;38;39;44;47].

2.2 Line Search Techniques for ALS Algorithm

Harshman[81] and Bro[10;25] proposed the line search method to predict the factors $\mathbf{A}^{(n)}$ from theirs previous estimations by a simple linear regression

$$\mathbf{A}_{t+1}^{(n)} = \mathbf{A}_t^{(n)} + \eta\, \Delta\mathbf{A}^{(n)}, \quad n = 1, 2, \ldots, N, \tag{2.6}$$

where t denotes the iteration index, $\Delta\mathbf{A}^{(n)} = \mathbf{A}_t^{(n)} - \mathbf{A}_{t-1}^{(n)}$ and stepsize η is iteratively searched so that the LS criterion (2.1) is reduced by substituting the step size $\eta \leftarrow \alpha\,\eta$, where α is a suitable constant.

The LS method can find a suitable step value but it is quite difficult to find an optimal step. To this end, the Exact Line Search (ELS)[70;200] or the Enhanced Line Search (ELS)[171] was proposed to find optimal η at every even iteration by minimizing the modified cost function of (2.1) with fixed $\mathbf{A}^{(n)}$

$$D(\mathcal{Y}||\hat{\mathcal{Y}}) = \frac{1}{2}\left\|\mathcal{Y} - \mathcal{I} \times \left\{\mathbf{A}^{(n)} + \eta\Delta\mathbf{A}^{(n)}\right\}\right\|_F^2.$$

An optimal stepsize is a root of a polynomial $\boldsymbol{p}(\eta)$ of degree $(2N-1)$ which yields the smallest cost function value or the highest fitness. ELS alleviates bottlenecks more efficiently than LS[51]. However, the technique demands high computational cost due to building up and solving $\boldsymbol{p}(\eta)$[200].

2.3 Hierarchical ALS Algorithm Using Squared Euclidean Distances

This section presents a variation of the ALS algorithm which sequentially updates components of the factors. Bro introduced column-wise formulation for the ALS algorithm in his thesis[25]. Heiser and Kroonenberg[208] also suggested a triadic update for 3-way CP in which the first loading vectors in every mode are first estimated, then the second etc. Cichocki *et. al.*[44] introduced the HALS algorithm for NMF in 2007. Ho *et. al.*[90;91] later analyzed convergence of the Rank-one Residue Iteration algorithm exploiting a similar idea to update one component instead of the factor. However, these algorithms are highly computational due to computation of residual tensors during iterations, and limited to matrix and 3-way tensor. Phan and Cichocki[152] proposed the optimized algorithm for CP and NTF in which residue tensors were bypassed. Moreover, the algorithm is straightforwardly derived as a special case

of the ALS algorithm[156;158]. The algorithm was later extended to the alpha- and beta- divergences[37]. For NTF, Phan *et. al.* extended the rank-one update algorithm to rank-K update algorithm[161].

To estimate a component $\boldsymbol{a}_r^{(n)}$, we assume all other components are fixed. The approximation tensor $\hat{\boldsymbol{\mathcal{Y}}}$ is split into two parts as follows

- A rank-one tensor $\hat{\boldsymbol{\mathcal{Y}}}^{(r)}$ is built up from components $\boldsymbol{a}_r^{(n)}$ to be estimated

$$\hat{\boldsymbol{\mathcal{Y}}}^{(r)} = \boldsymbol{a}_r^{(1)} \circ \boldsymbol{a}_r^{(2)} \circ \cdots \circ \boldsymbol{a}_r^{(N)}. \tag{2.7}$$

- A rank-(R-1) CP tensor $\hat{\boldsymbol{\mathcal{Y}}}^{(-r)}$ is composed by N factors $\mathbf{A}_{-r}^{(N)} = \left[\boldsymbol{a}_1^{(n)}, \ldots, \boldsymbol{a}_{r-1}^{(n)}, \boldsymbol{a}_{r+1}^{(n)}, \ldots, \boldsymbol{a}_R^{(n)}\right]$

$$\hat{\boldsymbol{\mathcal{Y}}}^{(-r)} = \sum_{k \neq r} \boldsymbol{a}_k^{(1)} \circ \boldsymbol{a}_k^{(2)} \circ \cdots \circ \boldsymbol{a}_k^{(N)} = \boldsymbol{\mathcal{I}} \times_1 \mathbf{A}_{-r}^{(1)} \times_2 \mathbf{A}_{-r}^{(2)} \cdots \times_N \mathbf{A}_{-r}^{(N)}. \tag{2.8}$$

According to the approximation $\boldsymbol{\mathcal{Y}} = \hat{\boldsymbol{\mathcal{Y}}}^{(-r)} + \hat{\boldsymbol{\mathcal{Y}}}^{(r)} + \boldsymbol{\mathcal{E}}$ by R components $\boldsymbol{a}_r^{(n)}, r = 1, \ldots, R, n = 1, \ldots, N$, we consider the residual tensor $\tilde{\boldsymbol{\mathcal{Y}}}^{(r)}$ which is approximated by the rank-one tensor $\hat{\boldsymbol{\mathcal{Y}}}^{(r)}$

$$\tilde{\boldsymbol{\mathcal{Y}}}^{(r)} = \boldsymbol{\mathcal{Y}} - \hat{\boldsymbol{\mathcal{Y}}}^{(-r)} = \hat{\boldsymbol{\mathcal{Y}}}^{(r)} + \boldsymbol{\mathcal{E}}. \tag{2.9}$$

Hence, the update rule for $\boldsymbol{a}_r^{(n)}$ is deduced from (2.2) as follows

$$\boldsymbol{a}_r^{(n)} \leftarrow \tilde{\mathbf{Y}}_{(n)}^{(r)} \left\{\boldsymbol{a}_r\right\}^{\odot_{-n}} \left(\left\{\boldsymbol{a}_r^T \boldsymbol{a}_r\right\}^{\circledast_{-n}}\right)^{\dagger} = \frac{\tilde{\mathbf{Y}}_{(n)}^{(r)} \left\{\boldsymbol{a}_r\right\}^{\odot_{-n}}}{\left\{\boldsymbol{a}_r^T \boldsymbol{a}_r\right\}^{\circledast_{-n}}} = \frac{\tilde{\boldsymbol{\mathcal{Y}}}^{(r)} \,\bar{\times}_{-n} \left\{\boldsymbol{a}_r\right\}}{\gamma_r^{(n)}}, \tag{2.10}$$

where $\tilde{\mathbf{Y}}_{(n)}^{(r)}$ is the mode-n unfolding tensor of $\tilde{\boldsymbol{\mathcal{Y}}}$, $\{\boldsymbol{a}_r\} \triangleq \{\boldsymbol{a}_r^{(1)}, \boldsymbol{a}_r^{(2)}, \ldots, \boldsymbol{a}_r^{(N)}\}$, and scaling coefficient $\gamma_r^{(n)}$ is computed as follows:

$$\gamma_r^{(n)} = \left\{\boldsymbol{a}_r^T \boldsymbol{a}_r\right\}^{\circledast_{-n}} = \left\{\boldsymbol{a}_r^T \boldsymbol{a}_r\right\}^{\circledast} / \left(\boldsymbol{a}_r^{(n)T} \boldsymbol{a}_r^{(n)}\right) = \begin{cases} \boldsymbol{a}_r^{(N)T} \boldsymbol{a}_r^{(N)}, & n \neq N \\ 1, & n = N. \end{cases} \tag{2.11}$$

Due to normalization $\boldsymbol{a}_r^{(n)} = \boldsymbol{a}_r^{(n)} / \|\boldsymbol{a}_r^{(n)}\|_2$ for $n = 1, 2, \ldots N-1$ after each iteration step, scaling factor $\gamma_r^{(n)}$ can be omitted and the learning rule (2.10) is simplified as

$$\begin{aligned} \boldsymbol{a}_r^{(n)} &\leftarrow \left(\boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{Y}}^{(-r)}\right) \,\bar{\times}_{-n} \left\{\boldsymbol{a}_r\right\} = \boldsymbol{\mathcal{Y}} \,\bar{\times}_{-n} \left\{\boldsymbol{a}_r\right\} - \boldsymbol{\mathcal{I}} \times_n \mathbf{A}_{-r}^{(n)} \,\bar{\times}_{-n} \left\{\mathbf{A}_{-r}^T \boldsymbol{a}_r\right\} \\ &= \boldsymbol{\mathcal{Y}} \,\bar{\times}_{-n} \left\{\boldsymbol{a}_r\right\} - \mathbf{A}_{-r}^{(n)} \left\{\mathbf{A}_{-r}^T \boldsymbol{a}_r\right\}^{\circledast_{-n}}, \end{aligned} \tag{2.12}$$

where $\left\{\mathbf{A}_{-r}^T \boldsymbol{a}_r\right\} \triangleq \left\{\mathbf{A}_{-r}^{(1)T} \boldsymbol{a}_r^{(1)}, \ldots, \mathbf{A}_{-r}^{(N)T} \boldsymbol{a}_r^{(N)}\right\}$. Product $\boldsymbol{\mathcal{Y}} \,\bar{\times}_{-n} \left\{\boldsymbol{a}_r\right\}$ is sequentially calculated as $(N-1)$ tensor-vector multiplications along all modes, but mode-n. We note that a tensor-vector product $\boldsymbol{\mathcal{Y}} \,\bar{\times}_k \boldsymbol{a}_r^{(k)}$ results an $(N-1)$-dimensional tensor and requires I^N multiplications, that means it reduces dimension of the resulted tensor by 1. Therefore, product $\boldsymbol{\mathcal{Y}} \,\bar{\times}_{-n} \left\{\boldsymbol{a}_r\right\}$ needs in total a number

Algorithm 2.1: HALS Algorithm for CP and NTF

Input: $\mathcal{Y}$: input data of size $I_1 \times I_2 \times \cdots \times I_N$,
R: number of basis components
Output: N factors $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$

1 **begin**
2 Initialize all $\mathbf{A}^{(n)}$ and normalize all $\boldsymbol{a}_r^{(n)}$ to unit length
3 **repeat**
4 **for** $n = 1$ *to* N **do** // Update $\mathbf{A}^{(n)}$
5 **for** $r = 1$ *to* R **do**
6 $\boldsymbol{a}_r^{(n)} \leftarrow \mathcal{Y} \bar{\times}_{-n} \{\boldsymbol{a}_r\} - \mathbf{A}_{-r}^{(n)} \left\{\mathbf{A}_{-r}^T \boldsymbol{a}_r\right\}^{\circledast -n}$
7 $\boldsymbol{a}_r^{(n)} \leftarrow \left[\boldsymbol{a}_r^{(n)}\right]_+$ // Retrieve nonnegative entries
8 **if** $n \neq N$ **then** // Normalize and fix scaling
9 $\boldsymbol{a}_r^{(N)} = \frac{\boldsymbol{a}_r^{(N)} \|\boldsymbol{a}_r^{(n)}\|_2}{\|\boldsymbol{a}_r^{(N)}\|_2^2}$, $\boldsymbol{a}_r^{(n)} = \frac{\boldsymbol{a}_r^{(n)}}{\|\boldsymbol{a}_r^{(n)}\|_2}$
10 **end**
11 **end**
12 **end**
13 **until** *a stopping criterion is met*
14 **end**

of multiplications of $I^N + I^{N-1} + \cdots + I^2 = \frac{I}{I-1}\left(I^N - I\right)$. In comparison with that of the ALS rule $RI^N + (N-2)RI^{N-1}$, the update rule (2.12) is at least R times less expensive than ALS (2.2). This is valid because (2.12) updates only one component whereas (2.2) updates R components, Moreover, by employing the cascade calculation, the update rule (2.12) does not demand significant temporal storage. It is necessary to bear in mind that the ALS rule (2.2) computes the Khatri-Rao product $\{\mathbf{A}\}^{\odot -n}$ which may result a large-scale matrix $I^{N-1} \times R$.

Pseudo-code of this algorithm is described in Algorithm 2.1. For NTF, a rectifier is applied to the update rule (2.12) as an additional step to retrieve nonnegative component shown in Step 7. In addition, normalization $\boldsymbol{a}_r^{(n)}$ to unit-length vector is also included. In stead of using the rectifier, this method can be extended for use in compressed sensing by combination with shrinkage rules[47;162;179].

2.4 HALS Algorithm with Constraints for NTF

Natural constraints such as sparseness, smoothness or uncorrelatedness (orthogonality) can be imposed on the factors. In this section, we derive the regularized HALS algorithm with such constraints. Generally, the cost function (2.2) can be incorporated additional penalty terms as

$$D_F^r(\mathcal{Y} \| \mathcal{I} \times \{\mathbf{A}\}) = D_F(\mathcal{Y} \| \mathcal{I} \times \{\mathbf{A}\}) + \sum_n \alpha_n J_{\mathbf{A}^{(n)}}, \tag{2.13}$$

where $J_{\mathbf{A}^{(n)}}$ are suitably designed regularization terms for factors $\mathbf{A}^{(n)}$, and regularization parameters $\alpha_n > 0$ control the amount of regularization. Each factor $\mathbf{A}^{(n)}$ can have independent parameter α_n.

2.4.1 Sparseness Constraints

The following penalty term is often used to impose sparseness

$$J_{\mathbf{A}}^{sp} = \sum_n \alpha_n^{sp} \|\mathbf{A}^{(n)}\|_1, \quad \frac{\partial J^{sp}}{\partial \mathbf{A}^{(n)}} = \alpha_n^{sp}\, \mathbf{1}_{I_n \times R}. \tag{2.14}$$

2.4.2 Orthogonality Constraints

The regularization term which enforces (as much as possible) orthogonality of basis components $\boldsymbol{a}_r^{(n)}$ is introduced in the cost function (2.13) as

$$J_n^{cr} = \alpha_n^{cr} \sum_{p \neq l} \boldsymbol{a}_p^{(n)T}\, \boldsymbol{a}_l^{(n)}, \qquad \frac{\partial J_n^{cr}}{\partial \boldsymbol{a}_r^{(n)}} = \alpha_n^{cr} \sum_{p \neq r} \boldsymbol{a}_p^{(n)} = \alpha_n^{cr}\, \mathbf{A}_{-r}^{(n)} \mathbf{1}_{R-1}\,. \tag{2.15}$$

That also means components $\boldsymbol{a}_r^{(n)}$ should be as sparse as possible. From the learning rule (2.12), we obtain the new learning rule with uncorrelatedness constraints given by

$$\boldsymbol{a}_r^{(n)} \leftarrow \frac{\left[\boldsymbol{\mathcal{Y}} \,\bar{\times}_{-n}\, \{\boldsymbol{a}_r\} - \mathbf{A}_{-r}^{(n)} \left\{\mathbf{A}_{-r}^T\, \boldsymbol{a}_r\right\}^{\circledast -n} + \alpha_n^{cr}\, \mathbf{A}_{-r}^{(n)}\, \mathbf{1}_{R-1}\right]_+}{\gamma_r^{(n)}}. \tag{2.16}$$

2.4.3 Smoothness Constraints

To measure the smoothness of components $\boldsymbol{a}_r^{(n)}$, we employ the penalty term defined as[137],

$$J_n^{sm} = \alpha_n^{sm}\, \|\varphi(\mathbf{L}\, \boldsymbol{a}_r^{(n)})\|_1, \tag{2.17}$$

where $\mathbf{L}$ is a suitably designed matrix (the Laplace operator) which measures the smoothness (by estimating the differences between neighboring samples of $\boldsymbol{a}_r^{(n)}$)[1] and $\varphi : \mathbb{R} \to \mathbb{R}$ is an edge-preserving function applied componentwise. Although this edge-preserving nonlinear function may take various forms[137]:

$$\varphi(t) = |t|^\alpha/\alpha, \quad 1 \leqslant \alpha \leqslant 2, \tag{2.18}$$

$$\varphi(t) = \sqrt{\alpha + t^2}, \tag{2.19}$$

$$\varphi(t) = 1 + |t|/\alpha - \log(1 + |t|/\alpha), \quad \alpha > 0, \tag{2.20}$$

[1] In the special case for $\mathbf{L} = \mathbf{I}_{I_n}$ and $\varphi(t) = |t|$, the smoothness regularization term becomes sparsity term.

we restrict ourself to simple cases, where $\varphi(t) = |t|^{\alpha}/\alpha$ for $\alpha = 1$ or 2, and $\mathbf{L}$ is the derivative operator of the first or second order. For example, the first order derivative operator $\mathbf{L}$ with I_n points can take the form:

$$\mathbf{L} = \begin{pmatrix} 1 & -1 & & & \\ & 1 & -1 & & \\ & & \ddots & \ddots & \\ & & & 1 & -1 \end{pmatrix}, \tag{2.21}$$

and the cost function (2.13) becomes similar to the total-variation (TV) regularization (which is often used in signal and image recovery[6;35]):

$$D_F^r(\boldsymbol{\mathcal{Y}} \| \boldsymbol{\mathcal{I}} \times \{\mathbf{A}\}) \;=\; D_F(\boldsymbol{\mathcal{Y}} \| \boldsymbol{\mathcal{I}} \times \{\mathbf{A}\}) + \alpha_{sm} \sum_{k=1}^{I_n - 1} |a_{kr}^n - a_{(k+1)r}^{(n)}|. \tag{2.22}$$

Another important case assumes that $\varphi(t) = \frac{1}{2}|t|^2$ and $\mathbf{L}$ is the second order derivative operator with I_n points[90]. In such a case, we obtain the Tikhonov-like regularization:

$$D_F^r(\boldsymbol{\mathcal{Y}} \| \boldsymbol{\mathcal{I}} \times \{\mathbf{A}\}) \;=\; D_F(\boldsymbol{\mathcal{Y}} \| \boldsymbol{\mathcal{I}} \times \{\mathbf{A}\}) + \frac{1}{2}\alpha_{sm} \, \|\mathbf{L}\boldsymbol{a}_r^{(n)}\|_2^2. \tag{2.23}$$

In such a case the update rule for $\boldsymbol{a}_r^{(n)}$ is given by:

$$\boldsymbol{a}_r^{(n)} \leftarrow (\gamma_r^{(n)}\mathbf{I} + \alpha_{sm} \, \mathbf{L}^T\mathbf{L})^{-1}(\boldsymbol{\mathcal{Y}} \,\bar{\times}_{-n}\, \{\boldsymbol{a}_r\} - \mathbf{A}_{-r}^{(n)} \, \{\mathbf{A}_{-r}^T \, \boldsymbol{a}_r\}^{\circledast_{-n}}). \tag{2.24}$$

Bro[25] considered a particular case in which $\mathbf{L}$ is the second-order smoothing operator. The learning rule (2.24) is robust to noise. However, it involves a rather high computational cost due to the calculation of an inverse of a large matrix in each iteration. To circumvent this problem and to considerably reduce the complexity of the algorithm we present a second-order smoothing operator $\mathbf{L}$ in the following form:

$$\begin{aligned} \mathbf{L} &= \begin{pmatrix} -2 & 2 & & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 2 & -2 \end{pmatrix} \\ &= \begin{pmatrix} -2 & & & & \\ & -2 & & & \\ & & \ddots & & \\ & & & -2 & \\ & & & & -2 \end{pmatrix} + \begin{pmatrix} 0 & 2 & & & \\ 1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 0 & 1 \\ & & & 2 & 0 \end{pmatrix} \\ &= -2\mathbf{I} + 2\mathbf{S}. \end{aligned} \tag{2.25}$$

However, instead of computing directly $\mathbf{L}\boldsymbol{a}_r^{(n)} = -2\,\boldsymbol{a}_r^{(n)} + 2\mathbf{S}\boldsymbol{a}_r^{(n)}$, in the second term we can approximate $\boldsymbol{a}_r^{(n)}$ by its estimation $\hat{\boldsymbol{a}}_r^{(n)}$ obtained from the previous update. Hence, the smoothing regularization

term with $\varphi(t) = |t|^2/8$ takes a simplified and computationally efficient form:

$$J_n^{sm} = \frac{\alpha_n^{sm}}{2} \| \boldsymbol{a}_r^{(n)} - \mathbf{S}\hat{\boldsymbol{a}}_r^{(n)} \|_2^2 , \tag{2.26}$$

$$\frac{\partial J_n^{sm}}{\partial \boldsymbol{a}_r^{(n)}} = \alpha_n^{sm}\, \boldsymbol{a}_r^{(n)} - \alpha_n^{sm}\, \mathbf{S}\hat{\boldsymbol{a}}_r^{(n)} . \tag{2.27}$$

Finally, the learning rule for the regularized (smoothed) HALS algorithm takes the following form:

$$\boldsymbol{a}_r^{(n)} \leftarrow \frac{\left[\boldsymbol{\mathcal{Y}} \,\bar{\times}_{-n} \{\boldsymbol{a}_r\} - \mathbf{A}_{-r}^{(n)} \left\{ \mathbf{A}_{-r}^T \boldsymbol{a}_r \right\}^{\circledast -n} + \alpha_n^{sm}\, \mathbf{S}\, \boldsymbol{a}^{(n)} \right]_+}{\gamma_r^{(n)} + \alpha_n^{sm}} . \tag{2.28}$$

2.5 Flexible HALS Using Alpha Divergence

The algorithms derived in previous sections can be extended to more robust algorithms by applying a family of generalized Alpha and Beta divergences. We consider a simple approximative NMF model[37;47] to illustrate the basic idea of the proposed algorithm for tensor decomposition

$$\mathbf{Y} \approx \mathbf{A}\mathbf{B}^T = \sum_{r=1}^{R} \boldsymbol{a}_r \boldsymbol{b}_r^T , \tag{2.29}$$

where $\mathbf{Y} \in \mathbb{R}_+^{I \times T}$ is a data matrix and the desired nonnegative matrices are expressed as $\mathbf{A} = [\boldsymbol{a}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_R] \in \mathbb{R}_+^{I \times R}$ and $\mathbf{B} = [\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_R] \in \mathbb{R}_+^{T \times R}$.

We define the Alpha divergence as follows[36;38;45;46]:

$$D_\alpha^{(r)} \left([\mathbf{Y}^{(r)}]_+ \,||\, \boldsymbol{a}_r \boldsymbol{b}_r^T \right) = \begin{cases} \displaystyle\sum_{ik} \left(\frac{z_{ik}^{(r)}}{\alpha(\alpha+1)} \left[\left(\frac{z_{ik}^{(r)}}{y_{ik}^{(r)}} \right)^{\alpha} - 1 \right] - \frac{z_{ik}^{(r)} - y_{ik}^{(r)}}{\alpha+1} \right), & \alpha \neq -1, 0, \quad (2.30a) \\ \displaystyle\sum_{ik} \left((z_{ik}^{(r)})\, \ln \left(\frac{z_{ik}^{(r)}}{y_{ik}^{(r)}} \right) - z_{ik}^{(r)} + y_{ik}^{(r)} \right), & \alpha = 0, \quad (2.30b) \\ \displaystyle\sum_{ik} \left(y_{ik}^{(r)} \ln \left(\frac{y_{ik}^{(r)}}{z_{ik}^{(r)}} \right) + z_{ik}^{(r)} - y_{ik}^{(r)} \right), & \alpha = -1, \quad (2.30c) \end{cases}$$

where $y_{ik}^{(r)} = y_{ik} - \sum_{p \neq r} a_{ip} b_{kp}$ and $z_{ik}^{(r)} = a_{ir} b_{kr}$ for $r = 1, 2, \ldots, R$.

The choice of parameter $\alpha \in \mathbb{R}$ depends on statistical distributions of noise and data. In the special cases of the Alpha divergence for $\alpha = \{1, -0.5, -2\}$, we obtain respectively the Pearson's chi squared, Hellinger's, and Neyman's chi-square distances while for the cases $\alpha = 0$ and $\alpha = -1$, the divergence has to be defined by the limits of (2.30a) as $\alpha \to 0$ and $\alpha \to -1$, respectively. When these limits are evaluated for $\alpha \to 0$ we obtain the generalized Kullback-Leibler divergence defined by Eq. (2.30b) whereas for $\alpha \to -1$ we have the dual generalized Kullback-Leibler divergence given in Eq. (2.30c)[8;42;45;46].

The gradient of the Alpha divergence (2.30) for $\alpha \neq -1$ with respect to a_{ir} and b_{kr} can be expressed in a compact form as:

$$\frac{\partial D_\alpha^{(r)}}{\partial b_{kr}} = \frac{1}{\alpha}\sum_i a_{ij}\left[\left(\frac{z_{ik}^{(r)}}{y_{ik}^{(r)}}\right)^\alpha - 1\right], \tag{2.31}$$

$$\frac{\partial D_\alpha^{(r)}}{\partial a_{ij}} = \frac{1}{\alpha}\sum_k b_{kr}\left[\left(\frac{z_{ik}^{(r)}}{y_{it}^{(r)}}\right)^\alpha - 1\right]. \tag{2.32}$$

By equating the gradients to zero, we obtain a new multiplicative local α-HALS algorithm:

$$\boldsymbol{b}_r \leftarrow \left(\frac{\left[\mathbf{Y}^{(r)\,T}\right]_+^{\bullet[\alpha]}\boldsymbol{a}_r}{\boldsymbol{a}_r^T\boldsymbol{a}_r^{\bullet[\alpha]}}\right)^{\bullet[1/\alpha]}, \quad \boldsymbol{a}_r \leftarrow \left(\frac{\left[\mathbf{Y}^{(r)}\right]_+^{\bullet[\alpha]}\boldsymbol{b}_r}{\boldsymbol{b}_r^T\boldsymbol{b}_r^{\bullet[\alpha]}}\right)^{\bullet[1/\alpha]}, \tag{2.33}$$

where the "rise to the power" operations $\boldsymbol{x}^{\bullet[\alpha]}$ are performed componentwise. The above algorithm can be generalized to the following form

$$\boldsymbol{b}_r \leftarrow \Psi^{-1}\left(\frac{\Psi\left(\left[\mathbf{Y}^{(r)T}\right]_+\right)\boldsymbol{a}_r}{\boldsymbol{a}_r^T\,\Psi(\boldsymbol{a}_r)}\right), \quad \boldsymbol{a}_r \leftarrow \Psi^{-1}\left(\frac{\Psi\left(\left[\mathbf{Y}^{(r)}\right]_+\right)\boldsymbol{b}_r}{\boldsymbol{b}_r^T\Psi(\boldsymbol{b}_r)}\right), \tag{2.34}$$

where $\Psi(\boldsymbol{x})$ is suitable chosen function, for example, $\Psi(\boldsymbol{x}) = \boldsymbol{x}^{\bullet[\alpha]}$, componentwise[2].

In a similar way, the learning rules for the N-order NTF problem can be derived. For this purpose, we consider the n-mode matricized (unfolded) version of the tensor $\mathcal{Y}$

$$\mathbf{Y}_{(n)} \approx \mathbf{A}^{(n)}\,(\mathbf{A}^{\odot -n})^T. \tag{2.35}$$

Actually, this can be considered as an NMF model with $\mathbf{A} \equiv \mathbf{A}^{(n)}$ and $\mathbf{B} \equiv \mathbf{A}^{\odot -n}$, $\boldsymbol{b}_r = [\mathbf{A}^{\odot -n}]_r = \{\boldsymbol{a}_r\}^{\odot -n}$. Applying directly the learning rule (2.34) to the model (2.35) gives

$$\boldsymbol{u}_r^{(n)} \leftarrow \Psi^{-1}\left(\frac{\Psi\left(\left[\tilde{\mathbf{Y}}_{(n)}^{(r)}\right]_+\right)\boldsymbol{b}_r}{\boldsymbol{b}_r^T\Psi(\boldsymbol{b}_r)}\right), \tag{2.36}$$

where $\tilde{\mathbf{Y}}_{(n)}^{(r)}$ is an n-mode matricized version of $\tilde{\mathcal{Y}}^{(r)}$

$$\tilde{\mathbf{Y}}_{(n)}^{(r)} = \mathbf{Y}_{(n)} - \widehat{\mathbf{Y}}_{(n)} + \boldsymbol{a}_r^{(n)}\boldsymbol{b}_r^T = \mathbf{Y}_{(n)} - \widehat{\mathbf{Y}}_{(n)} + \boldsymbol{a}_r^{(n)}\,\{\boldsymbol{a}_r\}^{\odot -nT}. \tag{2.37}$$

For a specific nonlinear function $\Psi(\cdot)$ ($\Psi(x) = x^\alpha$)

$$\begin{aligned}\Psi(\boldsymbol{b}_r) &= \Psi(\{\boldsymbol{a}_r\}^{\odot -n}) = \Psi(\boldsymbol{a}_r^{(N)}) \odot \cdots \odot \Psi(\boldsymbol{a}_r^{(n+1)}) \odot \Psi(\boldsymbol{a}_r^{(n-1)}) \cdots \odot \Psi(\boldsymbol{a}_r^{(1)}) \\ &= \{\Psi(\boldsymbol{a}_r)\}^{\odot -n},\end{aligned} \tag{2.38}$$

[2]For $\alpha = 0$ instead of $\Phi(x) = x^\alpha$ we used $\Phi(x) = \ln(x)$ [36].

and the denominator in (2.36) can be simplified as

$$\boldsymbol{b}_r^T \Psi(\boldsymbol{b}_r) = \{\boldsymbol{a}_r\}^{\odot -n T} \{\Psi(\boldsymbol{a}_r)\}^{\odot -n} = \{\boldsymbol{a}_r^T \Psi(\boldsymbol{a}_r)\}^{\circledast -n}, \tag{2.39}$$

this completes the derivation of a flexible Alpha-HALS NTF update rule, which in the tensor form is given by[37]

$$\boldsymbol{a}_r^{(n)} \leftarrow \Psi^{-1}\left[\frac{\Psi\left([\boldsymbol{\mathcal{Y}}^{(r)}]_+\right) \ \bar{\times}_{-n} \ \{\boldsymbol{a}_r\}}{\left\{\boldsymbol{a}_r^T \Psi(\boldsymbol{a}_r)\right\}^{\circledast -n}}\right]_+, \tag{2.40}$$

where all nonlinear operations are componentwise[3] .

2.6 Flexible HALS Using Beta Divergence

Beta divergence can be considered as a flexible and complementary cost function to the Alpha divergence. In order to obtain local NMF algorithms we introduce the following definition of the Beta divergence[36;45;126]:

$$D_\beta^{(r)}([\mathbf{Y}^{(r)}]_+ \,||\, \boldsymbol{a}_r \boldsymbol{b}_r^T) = \begin{cases} \displaystyle\sum_{ik} \left(([y_{ik}^{(r)}]_+) \, \frac{[y_{ik}^{(r)}]_+^\beta - z_{ik}^{(r)\beta}}{\beta} - \frac{[y_{ik}^{(r)}]_+^{\beta+1} - z_{ik}^{(r)\,\beta+1}}{\beta+1} \right), & \beta > 0, \quad (2.41a) \\ \displaystyle\sum_{ik} \left(([y_{ik}^{(r)}]_+) \, \ln\left(\frac{[y_{ik}^{(r)}]_+}{z_{ik}^{(r)}}\right) - [y_{ik}^{(r)}]_+ + z_{ik}^{(r)} \right), & \beta\text{=}0, \quad (2.41b) \\ \displaystyle\sum_{ik} \left(\ln\left(\frac{z_{ik}^{(r)}}{[y_{ik}^{(r)}]_+}\right) + \frac{[y_{ik}^{(r)}]_+}{z_{ik}^{(r)}} - 1 \right), & \beta\text{=-1}, \quad (2.41c) \end{cases}$$

where $y_{ik}^{(r)} = y_{ik} - \sum_{p \neq r} a_{ip} b_{kp}$ and $z_{ik}^{(r)} = a_{ir} b_{kr}$ for $r = 1, 2, \ldots, R$. The choice of the real-valued parameter $\beta \leqslant -1$ depends on the statistical distribution of data and the Beta divergence corresponds to Tweedie models[42;45;46;126]. For example, if we consider the Maximum Likelihood (ML) approach (with no a priori assumptions) the optimal estimation consists of minimization of the Beta Divergence measure when noise is Gaussian with $\beta = 1$. For the Gamma distribution $\beta = -1$, for the Poisson distribution $\beta = 0$, and for the compound Poisson $\beta \in (-1, 0)$. However, the ML estimation is not optimal in the sense of a Bayesian approach where a priori information of sources and mixing matrix (sparsity, nonnegativity) can be imposed. It is interesting to note that the Beta divergence as special cases includes the standard squared Euclidean distance (for $\beta = 1$), the Itakura-Saito distance ($\beta = -1$), and the generalized Kullback-Leibler divergence ($\beta = 0$).

[3] In practice, instead of half-wave rectifying we often use different transformations, e.g., real part of $\Psi(x)$ or adaptive nonnegative shrinkage function with gradually decreasing threshold till variance of noise σ^2_{noise}.

In order to derive a local learning algorithm, we compute the gradient of (2.41), with respect to elements to $b_{kr},\ a_{ir}$:

$$\frac{\partial D_\beta^{(r)}}{\partial b_{kr}} = \sum_i \left(z_{ik}^{(r)\,\beta} - ([y_{ik}^{(r)}]_+)\, z_{ik}^{(r)\,\beta-1} \right) a_{ir}\ , \tag{2.42}$$

$$\frac{\partial D_\beta^{(r)}}{\partial a_{ir}} = \sum_k \left(z_{ik}^{(r)\,\beta} - ([y_{ik}^{(r)}]_+)\, z_{ik}^{(r)\,\beta-1} \right) b_{kr}. \tag{2.43}$$

By equating the gradient components to zero, we obtain a set of simple HALS updating rules referred to as the Beta-HALS algorithm[37]:

$$b_{kr} \leftarrow \frac{1}{\sum_{i=1}^I a_{ir}^{\beta+1}} \sum_{i=1}^I a_{ir}^\beta\ ([y_{ik}^{(r)}]_+)\ , \tag{2.44}$$

$$a_{ir} \leftarrow \frac{1}{\sum_{k=1}^K b_{kr}^{\beta+1}} \sum_{k=1}^K b_{kr}^\beta\ ([y_{ik}^{(r)}]_+). \tag{2.45}$$

The above update rules can be written in a generalized compact vector form as

$$\boldsymbol{b}_r \leftarrow \frac{([\mathbf{Y}^{(r)\,T}]_+)\Psi(\boldsymbol{a}_r)}{\Psi(\boldsymbol{a}_r^T)\ \boldsymbol{a}_r}\ ,\quad \boldsymbol{a}_r \leftarrow \frac{([\mathbf{Y}^{(r)}]_+)\ \Psi(\boldsymbol{b}_r)}{\Psi(\boldsymbol{b}_r^T)\ \boldsymbol{b}_r}\ , \tag{2.46}$$

where $\Psi(\boldsymbol{b})$ is a suitably chosen convex function (e.g., $\Psi(\boldsymbol{b}) = \boldsymbol{b}^{\bullet[\beta]}$) and the nonlinear operations are performed element-wise.

The above learning rules could be generalized for the N-order NTF problem (using the similar approach as for the Alpha-HALS NTF):

$$\boldsymbol{a}_r^{(n)} \leftarrow \frac{([\tilde{\mathbf{Y}}_{(n)}^{(r)}]_+)\ \Psi(\boldsymbol{b}_r)}{\Psi(\boldsymbol{b}_r^T)\ \boldsymbol{b}_r}\ , \tag{2.47}$$

where $\boldsymbol{b}_r = \{\boldsymbol{a}_r\}^{\odot -n}$, and $\tilde{\mathbf{Y}}_{(n)}^{(r)}$ is defined in (2.37).

By taking into account (2.38), the learning rule (2.47) can be written as follows

$$\boldsymbol{a}_r^{(n)} \leftarrow \frac{([\tilde{\mathbf{Y}}_{(n)}^{(r)}]_+)\ \{\Psi(\boldsymbol{a}_r)\}^{\odot -n}}{\{\Psi(\boldsymbol{a}_r)\}^{\odot -n\,T}\ \{\boldsymbol{a}_r\}^{\odot -n}} = \frac{[\tilde{\boldsymbol{\mathcal{Y}}}^{(r)}]_+\ \bar{\times}_{-n}\ \{\Psi(\boldsymbol{a}_r)\}}{\{\Psi(\boldsymbol{a}_r)^T\ \boldsymbol{a}_r\}^{\circledast -n}}\ . \tag{2.48}$$

Actually, the update rule (2.48) can be simplified to reduce computational cost by normalization of vectors $\boldsymbol{a}_r^{(n)}$ for $n = 1, \ldots, N-1$ to unit length vectors after each iteration step:

$$\boldsymbol{a}_r^{(n)} \leftarrow \left[\tilde{\boldsymbol{\mathcal{Y}}}^{(r)}\ \bar{\times}_{-n}\ \{\Psi(\boldsymbol{a}_r)\} \right]_+,\quad \boldsymbol{a}_r^{(n)} \leftarrow \boldsymbol{a}_r^{(n)} / \|\boldsymbol{a}_r^{(n)}\|_2. \tag{2.49}$$

Once again, this algorithm can be rewritten in the fast form as follows

$$\boldsymbol{a}_r^{(n)} \leftarrow \left[\boldsymbol{\mathcal{Y}} \bar{\times}_{-n}\ \{\Psi(\boldsymbol{a}_r)\} - \mathbf{A}_{-r}^{(n)} \left\{ \Psi(\mathbf{A}_{-r})^T \boldsymbol{a}_r \right\}^{\circledast -n} \right]_+ . \tag{2.50}$$

The HALS NTF algorithm is a special case of (2.50) with $\Psi(x) = x$.

2.7 ALS Algorithm for Tucker Decomposition

The general Tucker model does not impose any constraint on factors and a core tensor. In many applications such as dimensionality reduction and feature extraction, several existing Tucker decomposition algorithms consider orthogonality of factors, such as the Higher-Order Singular Value Decomposition (HOSVD) and Higher Order Orthogonal Iteration (HOOI) algorithms[57;58;60;61]. For Nonnegative Tucker Decomposition (NTD), the multiplicative algorithms[47;103;131;151] are natural extensions of multiplicative Nonnegative Matrix Factorization (NMF) algorithms based on minimization of the squared Euclidean distance (Frobenius norm) and the Kullback-Leibler divergence. Mørup, Hansen, and Arnfred[131] extended multiplicative NMF (Nonnegative Matrix Factorizations) algorithms for sparse Nonnegative Tucker Decompositions (NTD), and also released the ERPWAVELAB toolbox[130] for analysis of multi-channel EEG and MEG data. These cost functions have been recently generalized and extended using the Bregman, Csiszár, and Alpha- and Beta- divergences[45;47;64;103]. The multiplicative algorithms have a relatively low complexity but they are characterized by rather slow convergence and they sometimes converge to spurious local minima.

We consider the squared Euclidean distance (Frobenius norm)[7;103;106;109] subject to nonnegativity constraints, that is

$$D_F(\mathcal{Y} \,||\mathcal{G} \times \{\mathbf{A}\}) = \frac{1}{2}\left\|\mathcal{Y} - \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)}\right\|_F^2 = \frac{1}{2}\left\|\mathcal{Y} - \hat{\mathcal{Y}}\right\|_F^2, \tag{2.51}$$

or its variation in the matricized form given by

$$D_F(\mathcal{Y} \,||\mathcal{G} \times \{\mathbf{A}\}) = \frac{1}{2}\left\|\mathbf{Y}_{(n)} - \mathbf{A}^{(n)}\mathbf{G}_{(n)}\mathbf{A}^{\otimes_{-n}T}\right\|_F^2. \tag{2.52}$$

A simple ALS update rule for $\mathbf{A}^{(n)}$ can be straightforwardly obtained from (2.52)

$$\mathbf{A}^{(n)} \leftarrow \mathbf{Y}_{(n)}\mathbf{A}^{\otimes_{-n}}\mathbf{G}_{(n)}^T\left(\mathbf{G}_{(n)}\left(\mathbf{A}^T\mathbf{A}\right)^{\otimes_{-n}}\mathbf{G}_{(n)}^T\right)^{-1} \tag{2.53}$$

$$= \langle \mathcal{Y} \times_{-n} \{\mathbf{A}^T\}, \mathcal{G}\rangle_{-n} \langle \mathcal{G} \times_{-n} \{\mathbf{A}^T\mathbf{A}\}, \mathcal{G}\rangle_{-n}^{-1}, \tag{2.54}$$

where $\langle \mathcal{X}, \mathcal{Y}\rangle_{-n} = \mathbf{X}_{(n)}\mathbf{Y}_{(n)}^T$ denotes contraction between $\mathcal{X}$ and $\mathcal{Y}$ along all modes except mode-n. A variation of the update rule (2.54) for NTD is combined with a rectifier, i.e., $\mathbf{A}^{(n)} \leftarrow \left[\mathbf{A}^{(n)}\right]_+$.

For an NTD model with nonnegative factors and nonnegative core tensor the ALS algorithms may not converge to a stable solution. Although Bro and Anderson[10] have applied the ALS algorithm to find the nonnegative factors, their algorithm does not allow to impose nonnegativity constraints for a core tensor. In fact, the learning rule (2.54) is not employed to decompose real-world data due to slow convergence and high computational cost. An alternative and well-known ALS algorithm for Tucker decomposition is the HOOI algorithm[58;60]. This section does not aim to introduce the ALS rule (2.54), but to derive an ALS algorithm for NTD with low complexity. The idea of the new algorithm for

Tucker decomposition and NTD is quite similar to that used to derive the HALS algorithm for CP and NTF in previous sections.

If $\mathbf{A}^{(n)}$ has only one component $R_n = 1$, the contracted product under the matrix inverse in (2.54) returns a scalar, matrix inverse will become a division by a scalar. This allows to establish an update rule for each component $\boldsymbol{a}_{r_n}^{(n)}$ of the factor $\mathbf{A}^{(n)}$ which does not require matrix inverse.

2.8 Hierarchical ALS Algorithm for Tucker Decomposition

The proposed algorithm for nonnegative Tucker decomposition consists of two parts:

- Derive update rule for each (nonnegative) component (vector) $\boldsymbol{a}_{r_n}^{(n)}$, for $n = 1, 2, \ldots, N$ and $r_n = 1, 2, \ldots, R_n$,
- Derive update rule for the core tensor $\mathcal{G}$.

With some adjustments of the basic cost function (2.51) and the standard ALS algorithm (2.54), we establish local learning rules for (nonnegative) components and for the core tensor.

2.8.1 Learning Rule for Factors $\mathbf{A}^{(n)}$

In order to estimate $\boldsymbol{a}_{r_n}^{(n)}$ $(\forall n, \forall r_n)$ we split the Tucker model (1.27) into two parts:

- A tensor consists of all rank-one tensors with which the specific component $\boldsymbol{a}_{r_n}^{(n)} \in \mathbb{R}_+^{I_n}$ is not involved

$$\mathcal{Y}^{(-r_n)} = \sum_{\{\boldsymbol{k}|k_n \neq r_n\}} g_{\boldsymbol{k}}\, \boldsymbol{a}_{k_1}^{(1)} \circ \boldsymbol{a}_{k_2}^{(2)} \circ \cdots \circ \boldsymbol{a}_{k_N}^{(N)} = \mathcal{G}_{-r_n} \times_{-n} \{\mathbf{A}\} \times_n \mathbf{A}_{-r_n}^{(n)},$$

  where $\boldsymbol{k} = [k_1, k_2, \ldots, k_N]$ is the index vector $\mathbf{1} \leqslant \boldsymbol{k} \leqslant \boldsymbol{R} = [R_1, R_2, \ldots, R_N]$, $\{\boldsymbol{k}|k_n \neq r_n\}$ denotes a set of vectors $\boldsymbol{k}$ in which $k_n \neq r_n$, $\mathbf{A}_{-r_n}^{(n)}$ is a version of $\mathbf{A}^{(n)}$ without $\boldsymbol{a}_{r_n}^{(n)}$. Subtensors $\mathcal{G}_{-r_n}$ of $\mathcal{G}$ do not consist of any entries $g_{k_1,\ldots,k_{n-1},r_n,k_{n+1},\ldots,k_N}$ with $k_m = 1, 2, \ldots, R_m, m \neq n$.

- Another part consists of all rank-one tensors which contain the component $\boldsymbol{a}_{r_n}^{(n)}$

$$\mathcal{Y}^{(+r_n)} = \sum_{\{\boldsymbol{k}|k_n = r_n\}} g_{\boldsymbol{k}}\, \boldsymbol{a}_{k_1}^{(1)} \circ \cdots \circ \boldsymbol{a}_{k_{n-1}}^{(n-1)} \circ \boldsymbol{a}_{r_n}^{(n)} \circ \boldsymbol{a}_{k_{n+1}}^{(n+1)} \circ \cdots \circ \boldsymbol{a}_{k_N}^{(N)} = \mathcal{G}_{r_n} \times_{-n} \{\mathbf{A}\} \times_n \boldsymbol{a}_{r_n}^{(n)},$$

  where $\mathcal{G}_{r_n} \in \mathbb{R}_+^{R_1 \times \cdots \times R_{n-1} \times 1 \times R_{n+1} \times \cdots \times R_N}$ is a subtensor of the core tensor $\mathcal{G}$ obtained by fixing the n-th index to r_n. Mode-n matricized version of tensor $\mathcal{G}_{r_n}$ is exactly the r_n-th row of mode-n matricized version of the core tensor $\mathcal{G}$, i.e., $[\mathcal{G}_{r_n}]_{(n)} = \left[\mathbf{G}_{(n)}\right]_{r_n:}$.

Using the above notations, we can rewrite the decomposition (1.27) of the data tensor $\mathcal{Y} = \mathcal{Y}^{(+r_n)} + \mathcal{Y}^{(-r_n)} + \mathcal{E}$. We consider a residual tensor $\tilde{\mathcal{Y}}^{(r_n)}$

$$\tilde{\mathcal{Y}}^{(r_n)} = \mathcal{Y} - \mathcal{Y}^{(-r_n)} = \mathcal{Y}^{(+r_n)} + \mathcal{E} == \mathcal{G}_{r_n} \times_{-n} \{\mathbf{A}\} \times_n \mathbf{a}_{r_n}^{(n)} + \mathcal{E}, \tag{2.55}$$

which is approximated by only one component $\mathbf{a}_{r_n}^{(n)}$ along mode-n. Hence, the ALS learning rule (2.54) to estimate the vector $\mathbf{a}_{r_n}^{(n)}$ from the tensor $\tilde{\mathcal{Y}}^{(r_n)}$ can be expressed in a relatively simple form as

$$\mathbf{a}_{r_n}^{(n)} \leftarrow \frac{\langle \tilde{\mathcal{Y}}^{(r_n)} \times_{-n} \{\mathbf{A}^T\}, \mathcal{G}_{r_n} \rangle_{-n}}{\langle \mathcal{G}_{r_n} \times_{-n} \{\mathbf{A}^T\mathbf{A}\}, \mathcal{G}_{r_n} \rangle_{-n}}, \tag{2.56}$$

for $r_n = 1, 2, \ldots, R_n$ and $n = 1, 2, \ldots, N$.

After some algebraic manipulations and replacing the tensor $\tilde{\mathcal{Y}}^{(r_n)}$ by (2.55),

$$\tilde{\mathcal{Y}}^{(r_n)} \times_{-n} \{\mathbf{A}^T\} = \mathcal{Y} \times_{-n} \{\mathbf{A}^T\} - \mathcal{G}_{-r_n} \times_{-n} \{\mathbf{A}^T\mathbf{A}\} \times_n \mathbf{A}_{-r_n}^{(n)}, \tag{2.57}$$

the learning rule (2.56) is simplified as

$$\begin{aligned} \mathbf{a}_{r_n}^{(n)} &\leftarrow \frac{\langle \mathcal{Y} \times_{-n} \{\mathbf{A}^T\}, \mathcal{G}_{r_n} \rangle_{-n} - \langle \mathcal{G}_{-r_n} \times_{-n} \{\mathbf{A}^T\mathbf{A}\} \times_n \mathbf{A}_{-r_n}^{(n)}, \mathcal{G}_{r_n} \rangle_{-n}}{\langle \mathcal{G}_{r_n} \times_{-n} \{\mathbf{A}^T\mathbf{A}\}, \mathcal{G}_{r_n} \rangle_{-n}} \\ &= \frac{\langle \mathcal{Y} \times_{-n} \{\mathbf{A}^T\}, \mathcal{G}_{r_n} \rangle_{-n} - \mathbf{A}_{-r_n}^{(n)} \langle \mathcal{G}_{-r_n}, \mathcal{G}_{r_n} \times_{-n} \{\mathbf{A}^T\mathbf{A}\} \rangle_{-n}}{\langle \mathcal{G}_{r_n}, \mathcal{G}_{r_n} \times_{-n} \{\mathbf{A}^T\mathbf{A}\} \rangle_{-n}}. \end{aligned} \tag{2.58}$$

We note that $\mathbf{A}^{(n)T}\mathbf{A}^{(n)}$ has relatively small size of $R_n \times R_n$ due to $R_n \ll I_n$. Moreover, tensor product $\mathcal{Y} \times_{-n} \{\mathbf{A}^T\}$ returns a tensor of size $R_1 \times \cdots \times R_{n-1} \times I_n \times R_{n+1} \cdots \times R_N$, and tensor product $\mathcal{G} \times_{-n} \{\mathbf{A}^T\mathbf{A}\}$ does not return a large tensor. Hence, they do not demand a significant extra storage.

2.8.2 Update Rules for the Core Tensor

The core tensor can be estimated using two following methods: global update for the whole tensor $\mathcal{G}$, and sequential update for each entry $g_{r_1 r_2 \ldots r_N}$.

2.8.2.1 Multiplicative Update Rule for Core Tensor $\mathcal{G}$

Vectorization of the Tucker model (1.27) gives us a nonnegative matrix factorization of vec($\mathcal{Y}$)

$$\text{vec}(\mathcal{Y}) = \left(\mathbf{A}^{(N)} \otimes \mathbf{A}^{(N-1)} \otimes \cdots \otimes \mathbf{A}^{(1)}\right) \text{vec}(\mathcal{G}). \tag{2.59}$$

The vector vec($\mathbf{G}$) can be estimated by using any existing NMF algorithms. For example, by applying the ISRA (Image Space Reconstruction Algorithm) update rule[55;63;67;110], also often referred to as

Algorithm 2.2: HALS Algorithm for Tucker and NTD

Input: $\mathcal{Y}$: input data of size $I_1 \times I_2 \times \cdots \times I_N$,
$R_1, R_2, \ldots, R_N$: number of basis components for each factor
Output: N factors $\mathbf{A}^{(n)} \in \mathbb{R}_+^{I_n \times R_n}$ and a core tensor $\mathcal{G} \in \mathbb{R}_+^{R_1 \times R_2 \times \cdots \times R_N}$

begin
  Initialization all $\mathbf{A}^{(n)}$ and $\mathcal{G}$
  repeat
    for $n = 1$ *to* N **do** // Update $\mathbf{A}^{(n)}$
      $\mathbf{V} = \langle \mathcal{Y} \times_{-n} \{\mathbf{A}^T\}, \mathcal{G} \rangle_{-n}, \quad \mathbf{W} = \langle \mathcal{G} \times_{-n} \{\mathbf{A}^T\mathbf{A}\}, \mathcal{G} \rangle_{-n}$
      for $r_n = 1$ *to* R_n **do**
        $\boldsymbol{a}_{r_n}^{(n)} \leftarrow \dfrac{\left[\boldsymbol{v}_{r_n} - \mathbf{A}_{-r_n}^{(n)} \boldsymbol{w}_{-r_n,r_n}\right]_+}{w_{r_n,r_n}}$
      end
      $\boldsymbol{\ell} = \left[\|\boldsymbol{a}_1^{(n)}\|_2, \ldots, \|\boldsymbol{a}_{R_n}^{(n)}\|_2\right]$
      $\mathcal{G} \leftarrow \mathcal{G} \times_n \text{diag}\{\boldsymbol{\ell}\}, \quad \mathbf{A}^{(n)} \leftarrow \mathbf{A}^{(n)} \text{ diag}\{\boldsymbol{\ell}^{\bullet[-1]}\}$
    end
    foreach $\mathbf{1} \leqslant \boldsymbol{r} = [r_1, \ldots, r_N] \leqslant [R_1, \ldots, R_N]$ **do** // Update $\mathcal{G}$
      $g_{\boldsymbol{r}} \leftarrow \left[g_{\boldsymbol{r}} + \mathcal{Y} \bar{\times} \{\boldsymbol{a}_{\boldsymbol{r}}\} - \mathcal{G} \bar{\times} \{\mathbf{A}^T \boldsymbol{a}_{\boldsymbol{r}}\}\right]_+$
    end
  until *a stopping criterion is met*
end

Lee-Seung algorithms[113], we obtain the following multiplicative update rule

$$\begin{aligned} \text{vec}(\mathcal{G}) \quad \leftarrow \quad & \text{vec}(\mathcal{G}) \circledast \left(\left(\mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(1)} \right)^T \text{vec}(\mathcal{Y}) \right) \oslash \\ & \left(\left(\mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(1)} \right)^T \left(\mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(1)} \right) \text{vec}(\mathcal{G}) \right) \\ = \quad & \text{vec}(\mathcal{G}) \circledast \text{vec}\left(\mathcal{Y} \times \left\{\mathbf{A}^T\right\}\right) \oslash \text{vec}\left(\mathcal{G} \times \left\{\mathbf{A}^T\mathbf{A}\right\}\right) \end{aligned} \tag{2.60}$$

which can be written in the tensor form as

$$\mathcal{G} \quad \leftarrow \quad \left[\mathcal{G} \circledast \left(\mathcal{Y} \times \{\mathbf{A}^T\}\right) \oslash \left(\mathcal{G} \times \{\mathbf{A}^T\mathbf{A}\}\right)\right]_+ . \tag{2.61}$$

2.8.2.2 Local Update Rule for Core Tensor $\mathcal{G}$

The multiplicative update rules have a relatively low complexity but are characterized by slow convergence and involve the risk of converging to spurious local minima. An alternative approach is that we sequentially update each entry $g_{r_1,r_2,\ldots,r_N}$ of the core tensor $\mathcal{G}$. In this section, we exploit this approach to derive local update rules for entries of the core tensor $\mathcal{G}$.

Intuitively, from the cost function (2.51), the ALS update rule for the core tensor $\mathcal{G}$ can be formulated as follows

$$\mathcal{G} \leftarrow \mathcal{Y} \times_1 \mathbf{A}^{(1)\dagger} \times_2 \mathbf{A}^{(2)\dagger} \cdots \times_N \mathbf{A}^{(N)\dagger}, \tag{2.62}$$

where $\mathbf{A}^{\dagger} = \left(\mathbf{A}^{(n)T}\mathbf{A}^{(n)}\right)^{-1}\mathbf{A}^{(n)T}$ denotes the Moore-Penrose pseudo inversion. However, this update formula requires to compute pseudo inverses of all factors $\mathbf{A}^{(n)}$ for each iteration step.

In order to explain a basic concept, let us assume first that all the factors $\mathbf{A}^{(n)}$ have only one vector, that is $R_n = 1, \forall n$: $\mathbf{A}^{(n)} = \boldsymbol{a}^{(n)}$. In such a simple scenario a data tensor is approximated by a rank-one tensor and the core tensor simplifies to a scalar $\mathcal{G} = g$, and the pseudo-inverses will become transposes of these factors

$$\boldsymbol{a}^{(n)\dagger} = \frac{\boldsymbol{a}^{(n)T}}{\boldsymbol{a}^{(n)T}\boldsymbol{a}^{(n)}}. \tag{2.63}$$

With the assumption that all the components are ℓ_2-norm unit length vectors: $\boldsymbol{a}^{(n)T}\boldsymbol{a}^{(n)} = 1$, the ALS update rule for $\mathcal{G}$ simplifies the tensor-vector products of the data tensor and the components

$$g \leftarrow \mathcal{Y} \times_1 \boldsymbol{a}^{(1)T} \times_2 \boldsymbol{a}^{(2)T} \cdots \times_N \boldsymbol{a}^{(N)T} = \mathcal{Y} \,\bar{\times}_1\, \boldsymbol{a}^{(1)} \,\bar{\times}_2\, \boldsymbol{a}^{(2)} \cdots \bar{\times}_N \boldsymbol{a}^{(N)} = \mathcal{Y}\bar{\times}\{\boldsymbol{a}\}. \tag{2.64}$$

The ALS update rule (2.64) for rank-one approximation is extremely simple and stable since we do not need to compute matrix inverse.

In the next step, we present an algorithm to estimate all the entries of the core tensor for the general Tucker decomposition with factors having more than one component. We assume that an entry $g_{\boldsymbol{r}}$, $\boldsymbol{r} = [r_1, r_2, \ldots, r_N]$ of the core tensor $\mathcal{G}$ needs to be updated. This entry has relation to components $\{\boldsymbol{a}_{\boldsymbol{r}}\} = \{\boldsymbol{a}_{r_1}^{(1)}, \boldsymbol{a}_{r_2}^{(2)}, \ldots, \boldsymbol{a}_{r_N}^{(N)}\}$. We divide the Tucker model (1.27) into two parts

- A tensor $\mathcal{Y}^{(-\boldsymbol{r})}$ consists of all the rank-one tensors which are not built up from the components $\boldsymbol{a}_{r_1}^{(1)}, \boldsymbol{a}_{r_2}^{(2)}, \ldots, \boldsymbol{a}_{r_N}^{(N)}$,

$$\mathcal{Y}^{(-\boldsymbol{r})} = \sum_{\boldsymbol{k}\neq\boldsymbol{r}} g_{\boldsymbol{k}}\, \boldsymbol{a}_{k_1}^{(1)} \circ \boldsymbol{a}_{k_1}^{(1)} \circ \cdots \circ \boldsymbol{a}_{k_N}^{(N)}. \tag{2.65}$$

- And a rank-one tensor built up from $\boldsymbol{a}_{r_1}^{(1)}, \boldsymbol{a}_{r_2}^{(2)}, \ldots, \boldsymbol{a}_{r_N}^{(N)}$: $\mathcal{Y}^{(+\boldsymbol{r})} = g_{\boldsymbol{r}}\, \boldsymbol{a}_{r_1}^{(1)} \circ \boldsymbol{a}_{r_2}^{(2)} \circ \cdots \circ \boldsymbol{a}_{r_N}^{(N)}$.

The decomposition (1.27) can be now rewritten as

$$\mathcal{Y} = \mathcal{Y}^{(-\boldsymbol{r})} + \mathcal{Y}^{(+\boldsymbol{r})} + \mathcal{E}. \tag{2.66}$$

To exploit the learning rule (2.64), we define a new residual tensor which is approximated by the rank-one tensor $\mathcal{Y}^{(+\boldsymbol{r})}$

$$\tilde{\mathcal{Y}}^{(\boldsymbol{r})} = \mathcal{Y} - \mathcal{Y}^{(-\boldsymbol{r})} = \mathcal{Y}^{(+\boldsymbol{r})} + \mathcal{E} = g_{\boldsymbol{r}}\, \boldsymbol{a}_{r_1}^{(1)} \circ \boldsymbol{a}_{r_2}^{(2)} \cdots \circ \boldsymbol{a}_{r_N}^{(N)} + \mathcal{E}, \tag{2.67}$$

Assume that all the components $\boldsymbol{a}_{r_n}^{(n)}$ are ℓ_2-norm unit length vectors: $\boldsymbol{a}_{r_n}^{(n)\,T}\,\boldsymbol{a}_{r_n}^{(n)} = 1$, the entry $g_{\boldsymbol{r}}$ can be updated using the learning rule (2.64)

$$\begin{aligned} g_{\boldsymbol{r}} &\leftarrow \tilde{\boldsymbol{\mathcal{Y}}}^{(\boldsymbol{r})} \,\bar{\times}\, \{\boldsymbol{a}_{\boldsymbol{r}}\} = \left(\boldsymbol{\mathcal{Y}} - \hat{\boldsymbol{\mathcal{Y}}} + \boldsymbol{\mathcal{Y}}^{(+\boldsymbol{r})}\right) \,\bar{\times}\, \{\boldsymbol{a}_{\boldsymbol{r}}\} \\ &= g_{\boldsymbol{r}} + \boldsymbol{\mathcal{Y}} \,\bar{\times}\, \{\boldsymbol{a}_{\boldsymbol{r}}\} - \boldsymbol{\mathcal{G}} \,\bar{\times}\, \{\mathbf{A}^T \boldsymbol{a}_{\boldsymbol{r}}\} \end{aligned} \tag{2.68}$$

where $\{\mathbf{A}^T\,\boldsymbol{a}_{\boldsymbol{r}}\} = \{\mathbf{A}^{(1)T}\,\boldsymbol{a}_{r_1}^{(1)}, \ldots, \mathbf{A}^{(N)T}\boldsymbol{a}_{r_N}^{(N)}\}$

Finally, the learning rules (2.58) and (2.68) are summarized in Algorithm 2.2 (referred here to as the HALS NTD algorithm). Note that $\boldsymbol{w}_{-r_n,r_n} = \left[w_{1,r_n}, \ldots, w_{r_{n-1},r_n}, w_{r_{n+1},r_n}, \ldots, w_{R_n,r_n}\right]^T$ consists of $(R_n - 1)$ entries extracted from vector $\boldsymbol{w}_{r_n}$ except w_{r_n,r_n}. Without rectifiers in Steps 7 and 13, Algorithm 2.2 is for Tucker decomposition.

2.8.3 Regularization for HALS NTD Algorithm

Similar to the HALS algorithms for NTF in Section 2.4, we can impose additional constraints such as sparseness, smoothness or uncorrelatedness (orthogonality) on the factors and the core tensor for NTD. Generally, the cost function (2.51) can be incorporated additional penalty terms as

$$D_F^r(\boldsymbol{\mathcal{Y}} \| \boldsymbol{\mathcal{G}} \times \{\mathbf{A}\}) = D_F(\boldsymbol{\mathcal{Y}} \| \boldsymbol{\mathcal{G}}, \{\mathbf{A}\}) + \sum_n \alpha_n J_{\mathbf{A}^{(n)}} + \alpha_G J_{\boldsymbol{\mathcal{G}}}, \tag{2.69}$$

where $J_{\mathbf{A}^{(n)}}$ and $J_{\boldsymbol{\mathcal{G}}}$ are suitably designed regularization terms for factors $\mathbf{A}^{(n)}$ and core tensor $\boldsymbol{\mathcal{G}}$, and regularization parameters $\alpha_n > 0$ and $\alpha_G > 0$ control the amount of regularization. Each factor $\mathbf{A}^{(n)}$ can have independent parameter α_n.

2.8.3.1 Sparseness Constraints

The following penalty term is often used to impose sparseness

$$J_{\mathbf{A}}^{sp} = \sum_n \alpha_n^{sp} \|\mathbf{A}^{(n)}\|_1, \quad \frac{\partial J^{sp}}{\partial \mathbf{A}^{(n)}} = \alpha_n^{sp}\, \mathbf{1}_{I_n \times R_n}. \tag{2.70}$$

2.8.3.2 Orthogonality Constraints

The regularization term which enforces (as much as possible) orthogonality of basis components $\boldsymbol{a}_{r_n}^{(n)}$ is introduced in the cost function (2.51) as

$$J_n^{cr} = \alpha_n^{cr} \sum_{p \neq l} \boldsymbol{a}_p^{(n)T}\, \boldsymbol{a}_l^{(n)}, \qquad \frac{\partial J_n^{cr}}{\partial \boldsymbol{a}_{r_n}^{(n)}} = \alpha_n^{cr} \sum_{p \neq r_n} \boldsymbol{a}_p^{(n)} = \alpha_n^{cr}\, \mathbf{A}^{(n)} \mathbf{1}_{R_n} - \alpha_n^{cr} \boldsymbol{a}_{r_n}^{(n)}. \tag{2.71}$$

That also means components $\boldsymbol{a}_{r_n}^{(n)}$ should be as sparse as possible. From the learning rule (2.58) and derivation given in A2.1, we obtain a new learning rule with uncorrelatedness constraints given by

$$\begin{aligned} \boldsymbol{a}_{r_n}^{(n)} &\leftarrow \frac{\left[\boldsymbol{v}_{r_n} - \mathbf{A}_{-r_n}^{(n)} \boldsymbol{w}_{-r_n,r_n} - \alpha_n^{cr} \mathbf{A}_{-r_n}^{(n)} \mathbf{1}\right]_+}{w_{r_n,r_n}}, && (2.72) \\ \mathbf{V} &= \langle \boldsymbol{\mathcal{Y}} \times_{-n} \{\mathbf{A}^T\}, \boldsymbol{\mathcal{G}} \rangle_{-n}, && (2.73) \\ \mathbf{W} &= \langle \boldsymbol{\mathcal{G}}, \boldsymbol{\mathcal{G}} \times_{-n} \{\mathbf{A}^T\mathbf{A}\} \rangle_{-n}. && (2.74) \end{aligned}$$

2.8.3.3 Smoothness Constraints

The penalty term is the same as that used in Section 2.4.3

$$J_n^{sm} = \frac{\alpha_n^{sm}}{2} \| \boldsymbol{a}_{r_n}^{(n)} - \mathbf{S}\hat{\boldsymbol{a}}_{r_n}^{(n)} \|_2^2, \qquad \frac{\partial J_n^{sm}}{\partial \boldsymbol{a}_{r_n}^{(n)}} = \alpha_n^{sm} \boldsymbol{a}_{r_n}^{(n)} - \alpha_n^{sm} \mathbf{S}\hat{\boldsymbol{a}}_{r_n}^{(n)}, \tag{2.75}$$

where $\mathbf{S}$ is defined in (2.25). The learning rule for the regularized (smoothed) HALS NTD algorithm takes the following form:

$$\boldsymbol{a}_{r_n}^{(n)} \leftarrow \frac{\left[\boldsymbol{v}_{r_n} - \mathbf{A}_{-r_n}^{(n)} \boldsymbol{w}_{-r_n,r_n} + \alpha_n^{sm} \mathbf{S}\, \boldsymbol{a}_{r_n}^{(n)}\right]_+}{w_{r_n,r_n} + \alpha_n^{sm}}. \tag{2.76}$$

2.8.3.4 Discriminant Constraints

One of important applications of NTD is feature extraction in which the core tensors represent reduced (compressed) features and factors are bases of the feature subspace. We consider the N-way tensor $\boldsymbol{\mathcal{Y}}$ as a training data which is concatenated from I_N samples (subtensors) $\boldsymbol{\mathcal{Y}}^{(k)} = \boldsymbol{\mathcal{Y}}_{i_N=k} \in \mathbb{R}^{I_1 \times I_2 \ldots \times I_{N-1}}$. To extract features from samples $\boldsymbol{\mathcal{Y}}^{(k)}$, we decompose the training tensor $\boldsymbol{\mathcal{Y}}$ into $N-1$ factors except the last mode. Factors $\mathbf{A}^{(n)}$ $(n = 1, 2, \ldots, N-1)$ will be regularized by between-class and within-class scatter matrices. It is interesting to notice that the features for a sample $\boldsymbol{\mathcal{Y}}^{(k)} = \boldsymbol{\mathcal{Y}}_{i_N=k}$ to classify an object can be obtained by a simple projection:

$$\begin{aligned} \boldsymbol{\mathcal{F}}^{(k)} &= \boldsymbol{\mathcal{Y}}^{(k)} \times_1 \mathbf{A}^{(1)T} \times_2 \mathbf{A}^{(2)T} \cdots \times_{N-1} \mathbf{A}^{(N-1)T} \\ &= \boldsymbol{\mathcal{Y}}^{(k)} \times_{-n} \mathbf{A}^T \times_n \mathbf{A}^{(n)T} = \boldsymbol{\mathcal{Z}}^{(k)} \times_n \mathbf{A}^{(n)T}. \end{aligned} \tag{2.77}$$

We denote the average feature tensor for class c by $\bar{\boldsymbol{\mathcal{F}}}^{(c)}$, for $c = 1, 2, \ldots, C$, and the average feature tensor for whole samples by $\bar{\bar{\boldsymbol{\mathcal{F}}}}$. The within-class and between-class scatter matrices $\mathbf{S}_w$ and $\mathbf{S}_b$ are defined as

$$\operatorname{tr}[\mathbf{S}_w] = \sum_{k=1}^{I_N} \| \boldsymbol{\mathcal{F}}^{(k)} - \bar{\boldsymbol{\mathcal{F}}}^{(c_k)} \|_F^2 = \operatorname{tr}\left[\mathbf{A}^{(n)T} \mathbf{S}_w^{(-n)} \mathbf{A}^{(n)}\right], \tag{2.78}$$

$$\operatorname{tr}[\mathbf{S}_b] = \sum_{c=1}^{C} K_c \| \bar{\boldsymbol{\mathcal{F}}}^{(c)} - \bar{\bar{\boldsymbol{\mathcal{F}}}} \|_F^2 = \operatorname{tr}\left[\mathbf{A}^{(n)T} \mathbf{S}_b^{(-n)} \mathbf{A}^{(n)}\right], \tag{2.79}$$

where c_k indicates the category of the sample k, K_c is the number of training samples in the c-th class, and symmetric scatter matrices $\mathbf{S}_w^{(-n)}$ and $\mathbf{S}_b^{(-n)}$ are expressed via tensor contracted products

$$\mathbf{S}_w^{(-n)} = \sum_{k}^{I_N} \langle \mathcal{Z}^{(k)} - \bar{\mathcal{Z}}^{(c_k)}, \mathcal{Z}^{(k)} - \bar{\mathcal{Z}}^{(c_k)} \rangle_{-n}, \tag{2.80}$$

$$\mathbf{S}_b^{(-n)} = \sum_{c}^{C} K_c \langle \mathcal{Z}^{(c)} - \bar{\bar{\mathcal{Z}}}, \mathcal{Z}^{(c)} - \bar{\bar{\mathcal{Z}}} \rangle_{-n}. \tag{2.81}$$

In order to find the discriminant basis factors for nonnegative Tucker decomposition, the overall cost function is designed using penalty terms so that $\mathrm{tr}\,[\mathbf{S}_w]$ is as small as possible while $\mathrm{tr}\,[\mathbf{S}_b]$ is as large as possible

$$J^{dc} = \frac{1}{2}\alpha_w \,\mathrm{tr}\,[\mathbf{S}_w] - \frac{1}{2}\alpha_b \,\mathrm{tr}\,[\mathbf{S}_b]\ . \tag{2.82}$$

The partial derivative with respect to $\mathbf{A}^{(n)}$ is given by

$$\frac{\partial J^{dc}}{\partial \mathbf{A}^{(n)}} = \left(\alpha_w \,\mathbf{S}_w^{(-n)} - \alpha_b \,\mathbf{S}_b^{(-n)}\right) \mathbf{A}^{(n)}. \tag{2.83}$$

From the learning rule (2.58) (assuming that the discriminant terms are in the numerator), we obtain the new learning rule for factors $\mathbf{A}^{(n)}$ as

$$\boldsymbol{a}_{r_n}^{(n)} \leftarrow \frac{\left[\boldsymbol{v}_{r_n} - \mathbf{A}_{-r_n}^{(n)} \boldsymbol{w}_{-r_n,r_n} + \mathbf{S}^{(-n)} \boldsymbol{a}_{r_n}^{(n)}\right]_+}{w_{r_n,r_n}} \tag{2.84}$$

where $\mathbf{S}^{(-n)} = \alpha_w \,\mathbf{S}_w^{(-n)} - \alpha_b \,\mathbf{S}_b^{(-n)}$. When the discriminant terms are in the denominator, an alternative implementation of the learning rule (2.84) is given by

$$\boldsymbol{a}_{r_n}^{(n)} \leftarrow \left[(\mathbf{S}^{(-n)} + w_{r_n,r_n}\mathbf{I})^{-1} \left(\boldsymbol{v}_{r_n} - \mathbf{A}_{-r_n}^{(n)} \boldsymbol{w}_{-r_n,r_n}\right)\right]_+. \tag{2.85}$$

2.8.3.5 General HALS NTD Algorithm with Multiple Constraints

The HALS NTD algorithm can simultaneously impose multiple regularization terms on factors $\mathbf{A}^{(n)}$. From the learning rules (2.72), (2.76) and (2.84), the general learning rule with orthogonality, smoothness and discriminant constraints can be formulated as

$$\boldsymbol{a}_{r_n}^{(n)} \leftarrow \frac{\left[\boldsymbol{v}_{r_n} - \mathbf{A}_{-r_n}^{(n)} \boldsymbol{w}_{-r_n,r_n} - \alpha_n^{cr} \mathbf{A}_{-r_n}^{(n)} \mathbf{1} + (\alpha_n^{sm} \mathbf{S} + \alpha_b \,\mathbf{S}_b^{(-n)} - \alpha_w \,\mathbf{S}_w^{(-n)})\, \boldsymbol{a}_{r_n}^{(n)}\right]_+}{w_{r_n,r_n} + \alpha_n^{sm}}. \tag{2.86}$$

2.9 HALS Algorithm for Large-Scale Data

The HALS algorithms are developed to avoid matrix inverses which are necessary in the ALS algorithms and to reduce the complexity of computation via sequential estimation of the components. To estimate $\mathbf{A}^{(n)}$, we need to calculate product $\mathcal{Y} \times_{-n} \{\mathbf{A}^T\}$ as in Step 5 in Algorithm 2.2. Although this product does not produce significant temporal storage, for large-scale tensor $\mathcal{Y}$, it might be computationally demanding. In this section, we consider a method to deal with this problem. We note that real-world tensors are often sparse or the features replicate along modes. This gives us an ability to manipulate tensors on some tubes which are sampled along modes instead of computing the full tensor.

For each mode-n, we select a set of (random) indices $\mathcal{I}_n$ of J_n, typically $R_n \leqslant J_n \ll I_n$,

$$\mathcal{I}_n = \left\{1 \leqslant \tilde{i}_1 < \tilde{i}_2 < \ldots < \tilde{i}_{J_n} \leqslant I_n \right\}, \tag{2.87}$$

then, build up sub-factors $\tilde{\mathbf{A}}^{(n)} \in \mathbb{R}^{J_n \times R_n}$ from J_n rows of factor $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ with row indices indicated by $\mathcal{I}_n$, which can be written as

$$\tilde{\mathbf{A}}^{(n)} = \mathbf{A}^{(n)}(\mathcal{I}_n, :) \quad \in \mathbb{R}^{J_n \times R_n}. \tag{2.88}$$

Selected indices can be identified by the CUR decomposition or its extension for multiway data[30;75].

The learning rule (2.58) can be reformulated using subfactors $\tilde{\mathbf{A}}^{(n)}$ as follows

$$\boldsymbol{a}_{r_n}^{(n)} \leftarrow \frac{\langle \tilde{\mathcal{Y}}^{(n)} \times_{-n} \{\tilde{\mathbf{A}}^T\}, \mathcal{G}_{-r_n} \rangle_{-n} - \mathbf{A}_{-r_n}^{(n)} \langle \mathcal{G}_{-r_n}, \mathcal{G}_{r_n} \times_{-n} \{\tilde{\mathbf{A}}^T \tilde{\mathbf{A}}\} \rangle_{-n}}{\langle \mathcal{G}_{r_n}, \mathcal{G}_{r_n} \times_{-n} \{\tilde{\mathbf{A}}^T \tilde{\mathbf{A}}\} \rangle_{-n}}, \tag{2.89}$$

where the reduced tensor $\tilde{\mathcal{Y}}^{(n)}$ takes samples from selected tubes along all modes but except mode-n

$$\tilde{\mathcal{Y}}^{(n)} = \mathcal{Y}(\mathcal{I}_1, \ldots, \mathcal{I}_{n-1}, :, \mathcal{I}_{n+1}, \ldots, \mathcal{I}_N). \tag{2.90}$$

The strategy of tube selection reduces the dimension of the error tensor $\tilde{\mathcal{Y}}^{(n)}$ to $R_1 \times R_{n-1} \times I_n \times R_{n+1} \times \ldots \times R_N$ which is much smaller than that of the full error tensor $\mathcal{Y}^{(n)}$. We note that the set of selected tube indices can be changed during the estimation of factors $\mathbf{A}^{(n)}$.

Finally, the learning rule (2.89) dramatically reduces the complexity of the HALS NTD algorithm (2.58) and is suitable for very large-scale low-rank and tensor approximations. In the experimental section, we illustrate the performance of this method for objects classification.

2.10 Speeding up HALS with Inner Loop

This section presents an efficient technique to speed up convergence of HALS algorithms. After some first iterations (for example 20, 30 iterations) for estimation factors $\mathbf{A}^{(n)}$ and core tensor $\mathcal{G}$, we impose

Algorithm 2.3: HALS Algorithm with Inner Loop

Input: $\mathcal{Y}$: input data of size $I_1 \times I_2 \times \cdots \times I_N$,
$R_1, R_2, \ldots, R_N$: number of basis components for each factor
Output: N factors $\mathbf{A}^{(n)} \in \mathbb{R}_+^{I_n \times R_n}$ and a core tensor $\mathcal{G} \in \mathbb{R}_+^{R_1 \times R_2 \times \cdots \times R_N}$

1 **begin**
2 Initialization all $\mathbf{A}^{(n)}$ and $\mathcal{G}$
3 **repeat**
4 **for** $n = 1$ *to* N **do**
5 **for** $k = 1$ *to* L **do** `// Inner loop`
6 Update $\mathbf{A}^{(n)*}$ from the current $\mathbf{A}^{(n)}$
7 **if** $\|\mathbf{A}^{(n)*} - \mathbf{A}^{(n)}\|_F < \varepsilon \|\mathbf{A}^{(n)}\|_F$ **then** Exit inner loop
8 $\mathbf{A}^{(n)} = \mathbf{A}^{(n)*}$
9 **end**
10 **end**
11 Update $\mathcal{G}$
12 **until** *a stopping criterion is met*
13 **end**

an inner loop to update $\mathbf{A}^{(n)}$ illustrated in Step 5 in Algorithm 2.3. The number of iterations in the inner loop L could be 10 or 20, the update can jump out of the loop when difference between two consecutive estimations is lower than a threshold (Step 8).

2.11 Summary

Rank-one update algorithms (or HALS) have been proposed for CP and Tucker with/without nonnegativity constraints. These algorithms avoid matrix inverses. Hence they are more stable than the ALS algorithms, especially for nearly collinear data. These algorithms also avoid Khatri-Rao and Kronecker products often arising in CP and Tucker algorithms. The proposed algorithms compute products of the data tensor and vectors. Because of less computational cost, the rank-one update algorithms are faster than the ALS algorithms. Moreover, multiple constraints such as orthogonality, sparseness, smoothness are straightforwardly imposed on the factors (and the core tensors for Tucker decompositions). These rank-one algorithms can face the same problem of the ALS algorithm such as bottleneck and swamp when data is highly collinear.

A2.1 Appendix: Derivation of Learning Rule for $\mathbf{A}^{(n)}$

This section presents an alternative derivation of the developed HALS NTD algorithm by minimizing a set of local cost functions instead of deriving it from the standard ALS updates rules.

To estimate the component $\boldsymbol{a}_{r_n}^{(n)}$, we assume that all the other components in all factors and the

core tensor are fixed. Instead of minimizing (2.51), we can use a more sophisticated approach by minimizing a set of local cost functions given by:

$$\begin{aligned} D_F^{(r_n)}(\boldsymbol{a}_{r_n}^{(n)}) &= \frac{1}{2}\Big\| \boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{Y}}^{(-r_n)} - \boldsymbol{\mathcal{Y}}^{(+r_n)} \Big\|_F^2 = \frac{1}{2}\Big\| \tilde{\boldsymbol{\mathcal{Y}}}^{(r_n)} - \boldsymbol{\mathcal{G}}_{j_n=r_n} \times_{-n} \{\mathbf{A}\} \times_n \boldsymbol{a}_{r_n}^{(n)} \Big\|_F^2 \\ &= \frac{1}{2}\Big\| \tilde{\mathbf{Y}}_{(n)}^{(r_n)} - \boldsymbol{a}_{r_n}^{(n)} \; [\boldsymbol{\mathcal{G}}_{j_n=r_n}]_{(n)} \; \mathbf{A}^{\otimes_{-n}\,T} \Big\|_F^2, \end{aligned} \tag{2.91}$$

for $r_n = 1, 2, \ldots, R_n$ and $n = 1, 2, \ldots, N$.

We first calculate the gradient of (2.91) with respect to vector $\boldsymbol{a}_{r_n}^{(n)}$

$$\frac{\partial D_F^{(r_n)}}{\partial \boldsymbol{a}_{r_n}^{(n)}} = -\left(\tilde{\mathbf{Y}}_{(n)}^{(r_n)} - \boldsymbol{a}_{r_n}^{(n)} \; [\mathbf{G}_{(n)}]_{r_n:} \, \mathbf{A}^{\otimes_{-n}\,T}\right) \; \mathbf{A}^{\otimes_{-n}} [\mathbf{G}_{(n)}]_{r_n:}^T \tag{2.92}$$

and set it to zero to obtain a learning rule for $\boldsymbol{a}_{r_n}^{(n)}$ ($n = 1, 2, \ldots, N$ and $r_n = 1, 2, \ldots, R_N$) given by

$$\boldsymbol{a}_{r_n}^{(n)} \leftarrow \frac{\tilde{\mathbf{Y}}_{(n)}^{(r_n)} \, \mathbf{A}^{\otimes_{-n}} \, [\mathbf{G}_{(n)}]_{r_n:}^T}{[\mathbf{G}_{(n)}]_{r_n:} \, \mathbf{A}^{\otimes_{-n}\,T} \, \mathbf{A}^{\otimes_{-n}} \, [\mathbf{G}_{(n)}]_{r_n:}^T} = \frac{1}{w_{r_n}} \langle \tilde{\boldsymbol{\mathcal{Y}}}^{(r_n)}, \boldsymbol{\mathcal{G}} \times_{-n} \{\mathbf{A}\} \rangle_{-n}. \tag{2.93}$$

The learning rule (2.93) is equivalent to the update rule given in (2.56).

A2.2 Appendix: Derivation of Learning Rule for Core Tensor $\mathcal{G}$

Entries of the core tensor can be sequentially updated with assumption that all components are fixed. We consider the following cost function

$$\begin{aligned} D_F &= \frac{1}{2}\Big\| \boldsymbol{\mathcal{Y}} - \boldsymbol{\mathcal{Y}}^{(-\boldsymbol{r})} - \boldsymbol{\mathcal{Y}}^{(+\boldsymbol{r})} \Big\|_F^2 = \frac{1}{2}\Big\| \tilde{\boldsymbol{\mathcal{Y}}}^{(\boldsymbol{r})} - \boldsymbol{\mathcal{Y}}^{(+\boldsymbol{r})} \Big\|_F^2 \\ &= \frac{1}{2}\Big\| \mathrm{vec}(\tilde{\mathbf{Y}}^{(\boldsymbol{r})}) - \left(\boldsymbol{a}_{r_N}^{(N)} \otimes \cdots \otimes \boldsymbol{a}_{r_1}^{(1)}\right) g_{\boldsymbol{r}} \Big\|_2^2. \end{aligned} \tag{2.94}$$

To derive the update rule, we calculate the gradient of (2.94) with respect to elements $g_{\bar{\boldsymbol{r}}}$

$$\frac{\partial D_F^{(\boldsymbol{r})}}{\partial g_{\bar{\boldsymbol{r}}}} = -\left(\boldsymbol{a}_{r_N}^{(N)} \otimes \cdots \otimes \boldsymbol{a}_{r_1}^{(1)}\right)^T \left(\mathrm{vec}(\tilde{\mathbf{Y}}^{(\boldsymbol{r})}) - \left(\boldsymbol{a}_{r_N}^{(N)} \otimes \cdots \otimes \boldsymbol{a}_{r_1}^{(1)}\right) g_{\bar{\boldsymbol{r}}}\right) \tag{2.95}$$

and set it to zero to yield a learning rule for entries of the core tensor $\boldsymbol{\mathcal{G}}$, given by

$$g_{\bar{\boldsymbol{r}}} \leftarrow \left[\frac{\left(\boldsymbol{a}_{r_N}^{(N)} \otimes \cdots \otimes \boldsymbol{a}_{r_1}^{(1)}\right)^T \mathrm{vec}(\tilde{\mathbf{Y}}^{(\boldsymbol{r})})}{\left(\boldsymbol{a}_{r_N}^{(N)} \otimes \cdots \otimes \boldsymbol{a}_{r_1}^{(1)}\right)^T \left(\boldsymbol{a}_{r_N}^{(N)} \otimes \cdots \otimes \boldsymbol{a}_{r_1}^{(1)}\right)} \right]_+. \tag{2.96}$$

This update rule can be simplified by taking into account that the Kronecker product of the two unit-length vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, i.e., $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$, is also a unit-length vector, that is, $\| \boldsymbol{c} \|_2^2 = \boldsymbol{c}^T \boldsymbol{c} = (\boldsymbol{a} \otimes \boldsymbol{b})^T (\boldsymbol{a} \otimes \boldsymbol{b}) = (\boldsymbol{a}^T \boldsymbol{a}) \otimes (\boldsymbol{b}^T \boldsymbol{b}) = 1 \otimes 1 = 1$. Hence, if all the components $\boldsymbol{a}_{r_n}^{(n)}$ are normalized to the ℓ_2-norm unit length vectors, we obtain a simplified version of the learning rule (2.96)

$$g_{\boldsymbol{r}} \leftarrow \left[\tilde{\boldsymbol{\mathcal{Y}}}^{(\boldsymbol{r})} \; \bar{\times}_1 \; \boldsymbol{a}_{r_1}^{(1)} \; \bar{\times}_2 \; \boldsymbol{a}_{r_2}^{(2)} \cdots \bar{\times}_N \; \boldsymbol{a}_{r_N}^{(N)} \right]_+. \tag{2.97}$$

CHAPTER 3

Appropriate ALS Algorithms for Nonnegative CP and Tucker Decompositions

Chapter 2 introduced the "work-horse" Alternating Least Squares (ALS) algorithms for CP and Tucker decompositions and adapted it for use in nonnegative tensor decompositions including CP and Tucker models by combining with the rectifier $[x]_+ = \max(x, 0)$. However, the modified algorithms can be relatively slow for nearly collinear data[7;78;109;210]. To improve performance and increase convergence speed, several simple techniques were proposed for the ALS algorithm, such as the weighted ALS incorporating a weighting matrix into the cost function[47], the line-search[140;171;200], or regularization terms controlling sparsity and orthogonality[41;78;210;215], the projected gradient methods[115], active set method[102]. Further modifications were discussed in Chapter 4[47].

Generally, these techniques help ALS cope with collinear and/or sparse factors. However, some other problems arise from controlling regularization parameters. The fact is that the ALS with a simple rectifier might not be an appropriate algorithm for NTF. Simple experiments with sparse and nonnegative factors can straightforwardly prove this confirmation. An alternative version of the ALS algorithm is the HALS algorithm[37;152] which estimates only one component for each factor, and avoids inverses of matrices. This algorithm was proved to outperform the multiplicative algorithms[73]. However, for tensor composed by highly collinear factors, both multiplicative and HALS algorithms often fail to factorize such data. Moreover, the HALS algorithm may not factorize sparse tensors without using additional regularization terms to control sparsity.

To this end, an appropriate ALS algorithm for NTF is proposed by recursively solving nonnegative quadratic programming problems in this chapter as follows

- Propose a recursive method for solving the nonnegative quadratic programming.
- Formulate ALS algorithms for nonnegative CP and Tucker decompositions based on solving nonnegative quadratic programming problems.
- Proposed ALS algorithm for low memory machine or for parallel computing system.

3.1 Recursive Update Rules for Nonnegative Quadratic Programming

In this section, we reinvestigate the nonnegative quadratic programming which involves applications in signal processing and machine learning such as nonnegative matrix/tensor factorizations and decompositions (NMF, NTF, NTD)[47;103;113;130;132;142;143;183], the classification by support vector machines[181;182].

Problem 3.1 (Nonnegative Quadratic Programming)
Consider the quadratic programming problem with nonnegative constraints

$$\begin{aligned} &\textit{minimize} && f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T \mathbf{Q}\, \boldsymbol{x} - \boldsymbol{b}^T \boldsymbol{x}, \\ &\textit{subject to} && \boldsymbol{x} \geqslant \mathbf{0}, \end{aligned} \tag{3.1}$$

where $\mathbf{Q} \in \mathbb{R}^{R\times R}$ *is a nonnegative symmetric positive-semidefinite matrix,* $\boldsymbol{b} = [b_1\, b_2\, \cdots b_R]^T$, $\boldsymbol{x} = [x_1\, x_2\, \cdots x_R]^T$. *This problem has an unique global optimal solution* $\boldsymbol{x}^* = [x_1^*\, x_2^*\, \cdots x_R^*]^T, x_r^* \geqslant 0, \forall r$.

Although the optimization problem (3.1) is similar to the NQP mentioned in[181;182] for SVM or for NMF[216], the matrix $\mathbf{Q}$ in our NQP is different, and is nonnegative. Due to the nonnegativity constraints, Problem 3.1 does not have an analytical solution. Sha *et. al*[182] proposed the multiplicative iterative learning rules employed SVM. Zdunek and Cichocki[216] proposed a method using second-order Taylor expansion with using second-order Taylor expansion. Some other methods such as the projected gradient[115], the exponentiated gradient[104] could be employed in order to solve (3.1). Practical results revealed that the multiplicative update rules converge slowly[115], while convergences of other iterative are quite sensitive to the choice of learning rate.

By exploiting the structure of the nonnegative matrix $\mathbf{Q}$ and the objective function (3.1), this section will derive a robust algorithm for NQP based on the recursive technique. The proposed algorithm can be applied to any applications involving nonnegative least squares approximation. We note that the nonnegative CP and Tucker decompositions can be converted to the nonnegative quadratic programming after a few mathematical manipulations.

Problem 3.1 can be solved by consideration of two Lemmas 3.1 and 3.2.

Lemma 3.1. *A stationary point of function (3.1) is denoted by*

$$\tilde{\boldsymbol{x}} = [\tilde{x}_1\, \tilde{x}_2\, \cdots\, \tilde{x}_R]^T = \mathbf{Q}^{-1}\, \boldsymbol{b}. \tag{3.2}$$

If $\exists \tilde{x}_r < 0$, *then* $x_r^* = 0$.

Proof. Without loss of generality, we assume that $\tilde{x}_R < 0$. The gradient of (3.1) with respect to x_R at point $x^*{}_1, \cdots, x^*{}_{R-1}$ given by

$$g(x_R) = \nabla_{x_R} f\,(x^*{}_1, \cdots, x^*{}_{R-1}, x_R) = q_{R,R}\, x_R + \boldsymbol{q}_{R,1:R-1}\, \boldsymbol{x}^*_{1:R-1} - b_R \tag{3.3}$$

Algorithm 3.1: Recursive Algorithm for NQP

Input: $\mathbf{Q}$: nonnegative symmetric matrix $(R \times R)$, $\boldsymbol{b}$: vectors of R entries
Output: $\boldsymbol{x}$: nonnegative vector minimizes (3.1)

```
begin
    I = {1, 2, ..., R}
    repeat
        x̃_I = Q_I^{-1} b_I
        I_- = {r ∈ I : x̃_r < 0}; I ← I \ I_-
    until I_- = ∅
    x = max{0, x̃}
end
```

has a solution given by

$$\breve{x}_R = \frac{b_R - \boldsymbol{q}_{R,1:R-1}\, \boldsymbol{x}^*_{1:R-1}}{q_{R,R}}. \tag{3.4}$$

Assume that $x^*_R > 0$, by taking into the assumption of $f\left(x^*{}_1, \cdots, x^*{}_{R-1}, x^*_R\right) \leqslant f\left(x^*{}_1, \cdots, x^*{}_{R-1}, x_R\right), \forall x_R \in [0, +\infty)$, we straightforwardly obtain $x^*_R \equiv \breve{x}_R$. That means $\boldsymbol{x}^*$ is the solution of equation (3.2): $\mathbf{Q}\,\boldsymbol{x}^* = \boldsymbol{b}$. Consequently, $\boldsymbol{x}^* \equiv \tilde{\boldsymbol{x}}$ or $x^*_R = \tilde{x}_R > 0$. That conflicts to the above assumption $\tilde{x}_R < 0$. Therefore, we have $x^*_R = 0$. □

Lemma 3.2 (Solution of Problem 3.1)**.** *Solution $\boldsymbol{x}^*$ of Problem 3.1 can be found under a recursive formulation.*

Proof. The gradient of the function $f(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ given by

$$\nabla f(\boldsymbol{x}) = \mathbf{Q}\,\boldsymbol{x} - \boldsymbol{b} \tag{3.5}$$

has an unique solution

$$\tilde{\boldsymbol{x}} = \left[\tilde{x}_1\ \tilde{x}_2\ \cdots\ \tilde{x}_R\right]^T = \mathbf{Q}^{-1}\,\boldsymbol{b}. \tag{3.6}$$

It is straightforward that if $\tilde{\boldsymbol{x}} \geqslant 0$, then $\boldsymbol{x}^* \equiv \tilde{\boldsymbol{x}}$ is solution of (3.1).

For another case, we assume that there are $K \leqslant R$ nonnegative entries $\bar{I} = \{r : \tilde{x}_r < 0, 1 \leqslant r \leqslant R\}, \mathtt{card}\{\bar{I}\} = K$. Based on Lemma 3.1, all the corresponding variables are zeros $x^*_{r_k} = 0$. The rest $(R - K)$ variables are solutions of a similar Problem 3.1 but of an lower order $(R - K)$

$$\begin{aligned} &\text{minimize} && f\left(\boldsymbol{x}_I\right) = \frac{1}{2}\boldsymbol{x}_I^T\,\mathbf{Q}_I\,\boldsymbol{x}_I - \boldsymbol{b}_I^T\,\boldsymbol{x}_I \\ &\text{subject to} && \boldsymbol{x}_I \geqslant \mathbf{0}, \end{aligned} \tag{3.7}$$

where $\mathbf{Q}_I$ is a part of the matrix $\mathbf{Q}$ deleted K rows and K columns with indices $\bar{I}$, and $\boldsymbol{b}_I$ is a part of the vector $\boldsymbol{b}$ removed K entries $\bar{I}$. This establishes a recursive formulation to find nonnegative solution of (3.1). This procedure described in Algorithm 3.1 iterates until there are not any zero entry $\tilde{x}$, that means $K = 0$.

Algorithm 3.1 has never fallen into infinite-loop and ensures to find the proper solution after a finite number of iterations which does not exceed I_n.

□

3.2 Novel Alternative Least Square Algorithm for NTF

3.2.1 QALS Algorithm for NTF

The proposed algorithm will be derived based on the ALS minimization of the squared Euclidean distance (Frobenius norm) to estimate the factor $\mathbf{A}^{(n)}$ while the others fixed. We convert the cost function for NTF to a quadratic function given by

$$\begin{aligned} D &= \frac{1}{2}\|\mathcal{Y}-\hat{\mathcal{Y}}\|_F^2 = \frac{1}{2}\|\mathcal{Y}\|_F^2 + \frac{1}{2}\|\hat{\mathcal{Y}}\|_F^2 - \langle \mathcal{Y}, \hat{\mathcal{Y}}\rangle \\ &= \frac{1}{2}\|\mathcal{Y}\|_F^2 + \frac{1}{2}\|\mathbf{A}^{(n)}\,\{\mathbf{A}\}^{\odot_{-n}\,T}\|_F^2 - \langle \mathbf{Y}_{(n)}, \mathbf{A}^{(n)}\{\mathbf{A}\}^{\odot_{-n}\,T}\rangle \\ &= \frac{1}{2}\|\mathcal{Y}\|_F^2 + \frac{1}{2}\mathrm{vec}\Big(\mathbf{A}^{(n)T}\Big)^T \Big(\mathbf{I}_{I_n}\otimes\mathbf{\Gamma}^{(n)}\Big)\mathrm{vec}\Big(\mathbf{A}^{(n)T}\Big) - \mathrm{vec}\Big(\mathbf{\Phi}^{(n)T}\Big)^T \mathrm{vec}\Big(\mathbf{A}^{(n)T}\Big)\,, \end{aligned} \tag{3.8}$$

where $\mathbf{\Gamma}^{(n)} = \{\mathbf{A}^T\mathbf{A}\}^{\circledast_{-n}} \in \mathbb{R}^{R\times R}$, and $\mathbf{\Phi}^{(n)} = \mathbf{Y}_{(n)}\{\mathbf{A}\}^{\odot_{-n}} \in \mathbb{R}^{I_n\times R}$. The first term in (3.8) $\|\mathcal{Y}\|_F^2$ is constant, hence it can be ignored. The last two terms in (3.8) can be formulated as a minimization problem

$$\begin{aligned} \min \quad & f\left(\mathfrak{a}^{(n)}\right) = \frac{1}{2}\,\mathfrak{a}^{(n)\,T}\,\mathbf{\Psi}^{(n)}\,\mathfrak{a}^{(n)} - \mathrm{vec}\Big(\mathbf{\Phi}^{(n)T}\Big)^T\,\mathfrak{a}^{(n)}\,, \\ \text{s.t.} \quad & \mathfrak{a}^{(n)} \geqslant \mathbf{0}, \end{aligned} \tag{3.9}$$

where $\mathbf{\Psi}^{(n)} = \mathbf{I}_{I_n}\otimes\mathbf{\Gamma}^{(n)} \in \mathbb{R}^{I_nR\times I_nR}$, and $\mathfrak{a}^{(n)}$ is a vectorized version of the factor $\mathbf{A}^{(n)\,T}$, and expressed by concatenation of all rows $\boldsymbol{a}_{i_n}^{(n)}$, $(i_n = 1,2,\ldots,I_n)$ of the factor $\mathbf{A}^{(n)}$

$$\mathfrak{a}^{(n)} = \mathrm{vec}\Big(\mathbf{A}^{(n)\,T}\Big) = \begin{bmatrix} \boldsymbol{a}_{1:}^{(n)} & \boldsymbol{a}_{2:}^{(n)} & \cdots & \boldsymbol{a}_{I_n:}^{(n)} \end{bmatrix}^T. \tag{3.10}$$

We have $\mathbf{A}^{(n)\,T}\mathbf{A}^{(n)}$ $(\forall n)$ are symmetric positive-semidefinite matrices. Moreover, for full column rank matrices $\mathbf{A}^{(n)}$, $\mathbf{A}^{(n)\,T}\mathbf{A}^{(n)}$ $(\forall n)$ are symmetric positive definite matrices. Hence, the Hadamard product $\mathbf{\Gamma}^{(n)}$ is again positive-(semi)definite according to the Schur product theorem[94]. This leads to the Kronecker product $\mathbf{\Psi}^{(n)}$ also a positive-(semi)definite[1]. As a consequence, the problem (3.9) is in the form of Problem 3.1 in which $\boldsymbol{x} \triangleq \mathfrak{a}^{(n)} \geqslant 0$ and $\mathbf{Q} \triangleq \mathbf{\Psi}^{(n)}$. The factor $\mathbf{A}^{(n)}$ can be updated using Algorithm 3.1.

It is worth noting that the stationary point of the function (3.9) without nonnegative constraints given by

$$\tilde{\mathfrak{a}}^{(n)} = \mathbf{\Psi}^{(n)^{-1}}\,\mathrm{vec}\Big(\mathbf{\Phi}^{(n)T}\Big) = \mathrm{vec}\Big(\Big(\mathbf{\Phi}^{(n)}\mathbf{\Gamma}^{(n)^{-1}}\Big)^T\Big) \tag{3.11}$$

is indeed a vectorized version of the ALS algorithm (2.2). If there is not any negative entry, the factor $\mathbf{A}^{(n)}$ is exactly the ALS update. Otherwise, we solve the reduced system as in (3.7). Pseudo-code of the proposed algorithm is listed in Algorithm 3.2. The function `nqp` refers to as Algorithm 3.1. If we

ignore the loop Step 3 in the function nqp, the ALS algorithm (2.2) is obtained. Normalization of the factors $\mathbf{A}^{(n)}$ is always necessary but is not explicitly described in Algorithm 3.2

$$\boldsymbol{a}_r^{(n)} = \frac{\boldsymbol{a}_r^{(n)}}{\|\boldsymbol{a}_r^{(n)}\|_2}, \quad n = 1, 2, \ldots, N-1, \forall r. \tag{3.12}$$

We note that the Kronecker product $\boldsymbol{\Psi}^{(n)} = \mathbf{I}_{I_n} \otimes \boldsymbol{\Gamma}^{(n)}$ is indeed the direct sum of I_n matrice $\boldsymbol{\Gamma}^{(n)}$ given by

$$\boldsymbol{\Psi}^{(n)} = \mathbf{I}_{I_n} \otimes \boldsymbol{\Gamma}^{(n)} = \begin{bmatrix} \boldsymbol{\Gamma}^{(n)} & & & \\ & \boldsymbol{\Gamma}^{(n)} & & \\ & & \ddots & \\ & & & \boldsymbol{\Gamma}^{(n)} \end{bmatrix}, \tag{3.13}$$

and produces a sparse matrix whose number of nonzero elements is only $I_n R^2$, and does not consume significant temporary extra-storage. Moreover, the inverse of the blockdiagonal matrix $\boldsymbol{\Psi}^{(n)}$ is quickly calculated based on the inverses of block matrices

$$\boldsymbol{\Psi}^{(n)^{-1}} = \begin{bmatrix} \boldsymbol{\Gamma}^{(n)^{-1}} & & & \\ & \boldsymbol{\Gamma}^{(n)^{-1}} & & \\ & & \ddots & \\ & & & \boldsymbol{\Gamma}^{(n)^{-1}} \end{bmatrix}. \tag{3.14}$$

The reduced version $\tilde{\boldsymbol{\Psi}}^{(n)}$ of the diagonal block matrix $\boldsymbol{\Psi}^{(n)}$ after deleting columns and rows with the same indices is also a diagonal block matrix. Hence, inverses of $\tilde{\boldsymbol{\Psi}}^{(n)}$ also can be expressed by inverses of its diagonal blocks as in (3.14). This helps to reduce computational cost in Step 4 of the NQP function nqp during the estimation Step 5 of Algorithm 3.2. Therefore, for low rank approximations $R \ll I_n$, Algorithm 3.1 has relatively low computational cost.

3.2.2 Algorithm for Low Memory Machine and Parallel Computing

The NQP problem (3.9) demands to solve a system of $I_n R$ variables. Although $\boldsymbol{\Psi}^{(n)}$ in (3.8) is a sparse matrix, for large-scale data and high-rank approximation, the complexity of Algorithm 3.2 increases rapidly with increasing the number of samples I_n and the number of components R. Hence, it could demand high computational cost, and also large space cost. To this end, an alternative algorithm will be presented in this section to run on low memory machine, or in a parallel system with multiple nodes.

From definition of vectors $\mathfrak{a}^{(n)}$ (3.10) and the structure of matrices $\boldsymbol{\Psi}^{(n)}$ (3.13), the cost function (3.8) is rewritten as I_n simultaneous nonnegative quadratic programming problems for all I_n rows $\boldsymbol{a}_{i_n:}^{(n)}$ $i_n = 1, 2, \ldots, I_n$ of the factor $\mathbf{A}^{(n)}$

$$\begin{aligned} \min \quad & f_{i_n}(\boldsymbol{a}_{i_n:}^{(n)}) = \frac{1}{2}\, \boldsymbol{a}_{i_n:}^{(n)}\, \boldsymbol{\Gamma}^{(n)}\, \boldsymbol{a}_{i_n:}^{(n)T} - \boldsymbol{\phi}_{i_n:}^{(n)}\, \boldsymbol{a}_{i_n:}^{(n)T}, \\ \text{s.t.} \quad & \boldsymbol{a}_{i_n:}^{(n)} \geqslant \mathbf{0} \end{aligned} \tag{3.15}$$

Algorithm 3.2: NQP for NTF - QALS

Input: $\mathcal{Y}$: tensor $I_1 \times I_2 \times \cdots \times I_N$,
R : number of approximation components
Output: N nonnegative factors $\mathbf{A}^{(n)}$ of size $(I_n \times R)$ minimize the problem (3.8)

```
1 begin
2     Random or leading singular vectors to initialize for A^(n)
3     repeat
4         for n = 1 to N do
5             vec(A^(n)T) = nqp(I_In ⊗ ({A^T A}^⊛−n), vec((Y_(n){A}^⊙−n)^T))
6         end
7     until a stopping criterion is met
8 end
```

where $\boldsymbol{\phi}_{i_n:}^{(n)}$ is the i_n-th row of the matrix $\boldsymbol{\Phi}^{(n)}$. For each problem (3.15), Algorithm 3.1 is straightforwardly applied to estimate the rows $\boldsymbol{a}_{i_n:}^{(n)}$ with $\mathbf{Q} \triangleq \boldsymbol{\Gamma}^{(n)}$, $\boldsymbol{x} \triangleq \boldsymbol{a}_{i_n:}^{(n)T}$ and $\boldsymbol{b} \triangleq \boldsymbol{\phi}_{i_n:}^{(n)}$. It is clear that the NQP problem (3.15) solves only R variables. Hence this demands much lower computational cost than solving Problem (3.9). The proposed method is suitable for large-scale problem on a limited-memory system.

Moreover, these simultaneous problems can be independently solved. They can be optimized to run for multi-core CPU PCs or GPU devices such as CUDA. The code also can be run on a remote cluster of computers using multiple workers to take advantage of parallel processing. The pseudo-code of this algorithm is given in Algorithm 3.3 in which `parfor` denotes the parallel loop. Matlab implementation of this algorithm allows each iteration of the `parfor` loop (Step 8) is executed in parallel on MATLAB workers. Because several workers can be solving concurrently the NQP (3.15) on the same loop, this algorithm can provide significantly better performance than Algorithm 3.2.

3.2.3 Complexity of QALS Algorithms

Computational complexity of Algorithms 3.2 and 3.3 depends mostly on inverse of matrices $\mathbf{Q} \equiv \boldsymbol{\Psi}^{(n)}$ or $\mathbf{Q} \equiv \boldsymbol{\Gamma}^{(n)}$ and the number of recursive iterations of Algorithm 3.1. For Algorithm 3.2, $\mathbf{Q}$ is a block diagonal matrix of I_n $(R \times R)$ matrices. For Algorithm 3.3, $\mathbf{Q}$ is only an $R \times R$ matrix. This means algorithms have space cost of order $O(IR^2)$ and $O(R^2)$, respectively. We note that Algorithm 3.2 updates the whole factor $\mathbf{A}^{(n)}$, and can convert to Algorithm 3.3 to update rows of $\mathbf{A}^{(n)}$. Hence, we only need to analyze complexity of Algorithm 3.3. In the worst case, the NQP algorithm repeats R times to update one row. Hence, the worst case complexity of Algorithm 3.1 is of order $O(R^3 + (R-1)^3 + \cdots + 2^3 + 1) = O(R^4)$.

Practical analysis on random tensors (discussed further in Section 6.2.3) shows that the number of recursive iterations did not exceed $(\log_2(R) + 1)$, was high on some first iterations of estimation,

Algorithm 3.3: Parallel Algorithm for NTF - pQALS

Input: $\mathcal{Y}$: tensor $I_1 \times I_2 \times \cdots \times I_N$,
R : number of approximation components
Output: N nonnegative factors $\mathbf{A}^{(n)}$ of size $(I_n \times R)$ minimize the problem (3.8)

begin
  Random or leading singular vectors to initialize for $\mathbf{A}^{(n)}$
  repeat
    for $n = 1$ **to** N **do**
      $\mathbf{\Gamma}^{(n)} = \{\mathbf{A}^T\mathbf{A}\}^{\circledast -n}$
      $\mathbf{\Phi}^{(n)} = \mathbf{Y}_{(n)}\{\mathbf{A}\}^{\odot -n}$
      parfor $i_n = 1$ **to** I_n **do** // parallel loop for update $\mathbf{A}^{(n)}$
        $\boldsymbol{a}_{i_n:}^{(n)\,T} = \mathrm{nqp}\left(\mathbf{\Gamma}^{(n)}, \boldsymbol{\phi}_{i_n:}^{(n)\,T}\right)$
      end
    end
  until *a stopping criterion is met*
end

and tended to be lower for later iterations. This result is valid because the number of variables to be tuned is reduced during the estimation until convergence is achieved. Illustration of occurrence rates for the number of recursive iterations in the NQP function is shown in Figure 6.13 for various tensor factorizations. As seen in Figure 6.13, even for $R = 600$, $I_n = 1000$, the NQP function in Algorithm 3.3 iterated almost 4 times. Let K be the smallest integer $2^K \geqslant R$: $K = \lceil \log_2(R) \rceil$. The worst case complexity of Algorithm 3.3 for update $\boldsymbol{a}_{i_n:}^{(n)}$ is $O(R^3 + (R-1)^3 + \cdots + (R-K+1)^3) = O(KR^3) = O(\log_2(R)R^3)$.

3.2.4 Simplified Algorithm for NMF

A simplified version of Algorithms 3.2 and 3.3 for 2 dimensional data (matrix) formulates the QALS algorithms for NMF described by the following model

$$\mathbf{Y} \approx \mathbf{A}\,\mathbf{X}^T, \tag{3.16}$$

where $\mathbf{Y} \in \mathbb{R}_+^{I \times J}$ is the observed data, $\mathbf{A} \in \mathbb{R}_+^{I \times R}$, and $\mathbf{X} \in \mathbb{R}^{J \times R}$ are two nonnegative factors. The algorithm sequentially updates factors $\mathbf{A}$ and $\mathbf{X}$ by solving two nonnegative quadratic programming problems

$$\begin{aligned}
&\min \quad f_{\mathbf{A}}(\mathbf{A}) = \frac{1}{2}\,\mathrm{vec}\big(\mathbf{A}^T\big)^T\,\big(\mathbf{I}_I \otimes \big(\mathbf{X}^T\mathbf{X}\big)\big)\,\mathrm{vec}\big(\mathbf{A}^T\big) - \mathrm{vec}\big(\mathbf{X}^T\mathbf{Y}^T\big)^T\,\mathrm{vec}\big(\mathbf{A}^T\big)\,,\\
&\text{s.t.} \quad \mathbf{A} \geqslant \mathbf{0},\\
&\min \quad f_{\mathbf{X}}(\mathbf{X}) = \frac{1}{2}\,\mathrm{vec}\big(\mathbf{X}^T\big)^T\,\big(\mathbf{I}_J \otimes \big(\mathbf{A}^T\mathbf{A}\big)\big)\,\mathrm{vec}\big(\mathbf{X}^T\big) - \mathrm{vec}\big(\mathbf{A}^T\mathbf{Y}\big)^T\,\mathrm{vec}\big(\mathbf{X}^T\big)\,,\\
&\text{s.t.} \quad \mathbf{X} \geqslant \mathbf{0},
\end{aligned}$$

Algorithm 3.4: QALS Algorithm for NMF

Input: $\mathbf{Y}$: nonnegative matrix $I \times J$
R : number of approximation components
Output: $\mathbf{A} \in \mathbb{R}^{I \times R}$ and $\mathbf{X} \in \mathbb{R}^{J \times R}$ nonnegative factors

```
begin
    Random or leading singular vectors to initialize A and X
    repeat
        vec(A^T) = nqp(I_I ⊗ (X^T X), vec(X^T Y^T))
        vec(X^T) = nqp(I_J ⊗ (A^T A), vec(A^T Y))
    until a stopping criterion is met
end
```

Pseudo code of this algorithm is given in Algorithm 3.4.

3.3 QALS Algorithm for NTD

In a similar way to derive the QALS algorithm for NTF, we derive an ALS algorithm for NTD by consideration of the cost function for NTD as an NQP problem as follows

$$\begin{aligned} D &= \frac{1}{2}\|\mathcal{Y} - \hat{\mathcal{Y}}\|_F^2 = \frac{1}{2}\|\mathcal{Y}\|_F^2 + \frac{1}{2}\|\hat{\mathcal{Y}}\|_F^2 - \langle \mathcal{Y}, \hat{\mathcal{Y}} \rangle \\ &= \frac{1}{2}\|\mathcal{Y}\|_F^2 + \frac{1}{2}\|\mathbf{A}^{(n)}\,\mathbf{G}_{(n)}\,\{\mathbf{A}\}^{\otimes_{-n}T}\|_F^2 - \langle \mathbf{Y}_{(n)}, \mathbf{A}^{(n)}\mathbf{G}_{(n)}\,\{\mathbf{A}\}^{\otimes_{-n}T} \rangle \\ &= \frac{1}{2}\|\mathcal{Y}\|_F^2 + \frac{1}{2}\mathrm{vec}\Big(\mathbf{A}^{(n)T}\Big)^T \Big(\mathbf{I}_n \otimes \mathbf{\Lambda}^{(n)}\Big)\mathrm{vec}\Big(\mathbf{A}^{(n)T}\Big) - \mathrm{vec}\Big(\mathbf{\Upsilon}^{(n)T}\Big)^T \mathrm{vec}\Big(\mathbf{A}^{(n)T}\Big), \end{aligned} \tag{3.17}$$

where $\mathbf{\Lambda}^{(n)} = \langle \mathcal{G} \times_{-n} \{\mathbf{A}^T\mathbf{A}\}, \mathcal{G} \rangle_{-n} \in \mathbb{R}^{R_n \times R_n}$, and $\mathbf{\Upsilon}^{(n)} = \langle \mathcal{Y} \times_{-n} \{\mathbf{A}^T\}, \mathcal{G} \rangle_{-n} \in \mathbb{R}^{I_n \times R_n}$. The first term in (3.17) $\|\mathcal{Y}\|_F^2$ is constant, hence it can be ignored. The last two terms in (3.8) can be formulated as a minimization problem to find $\mathbf{A}^{(n)}$

$$\begin{aligned} \min \quad & f\left(\mathfrak{a}^{(n)}\right) = \frac{1}{2}\,\mathfrak{a}^{(n)\,T}\,\mathbf{I}_n \otimes \mathbf{\Lambda}^{(n)}\,\mathfrak{a}^{(n)} - \mathrm{vec}\Big(\mathbf{\Upsilon}^{(n)T}\Big)^T\,\mathfrak{a}^{(n)}, \\ \text{s.t.} \quad & \mathfrak{a}^{(n)} \geqslant \mathbf{0}, \end{aligned} \tag{3.18}$$

where $\mathfrak{a}^{(n)}$ is a vectorized version of the factor $\mathbf{A}^{(n)\,T}$. In order to find $\mathcal{G}$, the cost function (3.17) is also rewritten

$$D = \frac{1}{2}\|\mathcal{Y}\|_F^2 + \frac{1}{2}\mathrm{vec}(\mathcal{G})^T \left(\{\mathbf{A}^T\mathbf{A}\}^{\otimes}\right)\mathrm{vec}(\mathcal{G}) - \mathrm{vec}\big(\mathcal{Y} \times \{\mathbf{A}^T\}\big)^T\,\mathrm{vec}(\mathcal{G}), \tag{3.19}$$

as

$$\begin{aligned} \min \quad & f(\boldsymbol{g}) = \frac{1}{2}\,\boldsymbol{g}^T\,\{\mathbf{A}^T\mathbf{A}\}^{\otimes}\,\boldsymbol{g} - \mathrm{vec}\big(\mathcal{Y} \times \{\mathbf{A}^T\}\big)^T\,\boldsymbol{g}, \\ \text{s.t.} \quad & \boldsymbol{g} = \mathrm{vec}(\mathcal{G}) \geqslant \mathbf{0}. \end{aligned} \tag{3.20}$$

Algorithm 3.5: NQP for NTD - QALS

Input: $\mathcal{Y}$: tensor $I_1 \times I_2 \times \cdots \times I_N$,
$R_1, R_2, \ldots, R_N$: number of approximation components
Output: N factors $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ and a core tensor $\mathcal{G}$ minimize the problem (3.17)

1 **begin**
2 Random or leading singular vectors to initialize for $\mathbf{A}^{(n)}$
3 **repeat**
4 **for** $n = 1$ *to* N **do**
5 $\mathbf{\Lambda}^{(n)} = \langle \mathcal{G} \times_{-n} \{\mathbf{A}^T\mathbf{A}\}, \mathcal{G} \rangle_{-n}$
6 $\mathbf{\Upsilon}^{(n)} = \langle \mathcal{Y} \times_{-n} \{\mathbf{A}^T\}, \mathcal{G} \rangle_{-n}$
7 $\text{vec}\big(\mathbf{A}^{(n)\,T}\big) = \texttt{nqp}\left(\mathbf{I}_n \otimes \mathbf{\Lambda}^{(n)}, \text{vec}\left(\mathbf{\Upsilon}^{(n)T}\right)\right)$
8 **end**
9 $\text{vec}(\mathcal{G}) = \texttt{nqp}\big(\{\mathbf{A}^T\mathbf{A}\}^{\otimes}, \text{vec}\big(\mathcal{Y} \times \{\mathbf{A}^T\}\big)\big)$
10 **until** *a stopping criterion is met*
11 **end**

According to Algorithm 3.2, factors $\mathbf{A}^{(n)}$ and core tensor $\mathcal{G}$ can be updated as follows

$$\text{vec}\left(\mathbf{A}^{(n)T}\right) \leftarrow \texttt{nqp}\left(\mathbf{I}_n \otimes \mathbf{\Lambda}^{(n)}, \text{vec}\left(\mathbf{\Upsilon}^{(n)T}\right)\right), \tag{3.21}$$

$$\text{vec}(\mathcal{G}) \leftarrow \texttt{nqp}\big(\{\mathbf{A}^T\mathbf{A}\}^{\otimes}, \text{vec}\big(\mathcal{Y} \times \{\mathbf{A}^T\}\big)\big). \tag{3.22}$$

Of course, we also have an alternative update rule for rows of factors $\mathbf{A}^{(n)}$ which is much low complexity than (3.21)

$$\boldsymbol{a}_{i_n:}^{(n)T} \leftarrow \texttt{nqp}\left(\mathbf{\Lambda}^{(n)}, \boldsymbol{\nu}_{i_n:}^{(n)T}\right). \tag{3.23}$$

Algorithm 3.5 illustrates pseudo-code of the above learning rules for NTD. In practice, in order to improve stability and convergence of the algorithm, we utilize the regularized NQP algorithm in section 3.4 instead of the nqp function in Steps 7 and 9.

3.4 Regularization for QALS Algorithms

An important point for QALS algorithms is that matrices $\mathbf{\Gamma}^{(n)}$, $\mathbf{\Lambda}^{(n)}$ are well-conditioned. For factorization of collinear factors or in some first iterations, an additional regularization parameter μ is added to matrices to ensure the stability of the algorithms. The parameter plays a role similar to that of the damping parameter in the Levenberg-Marquardt iteration. The parameter μ should be initialized by a large enough value, then slowly descends down to near-zero after each 5-10 iterations. Finally, the solution of the NQP problem is slightly modified as follows

$$\boldsymbol{x} = \texttt{nqp}(\mathbf{Q} + \mu\mathbf{I}, \boldsymbol{b}). \tag{3.24}$$

Algorithm 3.6: Rank-K Update Algorithm - rK-QALS

Input: $\mathcal{Y}$: tensor $I_1 \times I_2 \times \cdots \times I_N$,
R : number of approximation components
Output: N nonnegative factors $\mathbf{A}^{(n)}$ of size $(I_n \times R)$

begin
  Random or leading singular vectors to initialize for $\mathbf{A}^{(n)}$
  repeat
    for $n = 1$ *to* N **do**
      repeat
        $\mathcal{R} = \texttt{activeset}$
        $\mathbf{\Gamma}_{\mathcal{R}}^{(n)} = \{\mathbf{A}_{\mathcal{R}}^T \mathbf{A}_{\mathcal{R}}\}^{\circledast -n}$
        $\mathbf{\Phi}_{\mathcal{R}}^{(n)} = \mathbf{Y}_{(n)} \mathbf{A}_{\mathcal{R}}^{\odot -n} - \mathbf{A}_{\mathcal{R}}^{(n)} \left\{\mathbf{A}_{\mathcal{R}}^T \mathbf{A}_{\mathcal{R}}\right\}^{\circledast -n}$
        $\text{vec}\left(\mathbf{A}_{\mathcal{R}}^{(n)\,T}\right) = \texttt{nqp}\left(\mathbf{I}_{I_n} \otimes \mathbf{\Gamma}_{\mathcal{R}}^{(n)}, \text{vec}\left(\mathbf{\Phi}_{\mathcal{R}}^{(n)T}\right)\right)$
      until *all components* $\boldsymbol{a}_r^{(n)}$ *are updated*
    end
  until *a stopping criterion is met*
end

3.5 Rank-K Update QALS Algorithms

This section aims to derive learning rule to update a subset of $\tilde{R}$ components ($1 \leqslant \tilde{R} \leqslant R$). Assume that $\mathcal{R} = \{r_k : 1 \leqslant r_k \leqslant R, k = 1, 2, \ldots, \tilde{R}\}$ is a set of $\tilde{R}$ indices of components $\mathbf{A}_{\mathcal{R}}^{(n)} = \left[\boldsymbol{a}_{r_1}^{(n)}\, \boldsymbol{a}_{r_2}^{(n)} \cdots \boldsymbol{a}_{r_{\tilde{R}}}^{(n)}\right] \in \mathbb{R}_+^{I_n \times \tilde{R}}$ to be estimated. In order to derive the update rule for $\mathbf{A}_{\mathcal{R}}^{(n)}$, we split the CP model (1.11) into two parts:

- A tensor consists of all rank-one tensors in which the components $\mathbf{A}_{\mathcal{R}}^{(n)}$, $(n = 1, 2, \ldots, N)$ do not involve $\mathcal{Y}^{(-\mathcal{R})} = \sum_{r \notin \mathcal{R}} \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(N)}$.
- Another part consists of all rank-one tensors which are constructed from the components $\mathbf{A}_{\mathcal{R}}^{(n)}$ $(\forall n)$: $\mathcal{Y}^{(+\mathcal{R})} = \sum_{r \in \mathcal{R}} \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(N)} = \mathcal{I} \times \{\mathbf{A}_{\mathcal{R}}\}$.

We consider a residual tensor $\tilde{\mathcal{Y}}^{(+\mathcal{R})} = \mathcal{Y} - \mathcal{Y}^{(-\mathcal{R})} = \mathcal{Y}^{(+\mathcal{R})} + \mathcal{E} = \mathcal{I} \times \{\mathbf{A}_{\mathcal{R}}\} + \mathcal{E}$, which is approximated by $\tilde{R}$ components $\mathbf{A}_{\mathcal{R}}^{(n)}$ along the mode-n. The cost function (2.1) is rewritten as a cost function of the factors $\mathbf{A}_{\mathcal{R}}^n$ as follows

$$\begin{aligned} D^{(+\mathcal{R})} &= \frac{1}{2}\|\tilde{\mathcal{Y}}^{(+\mathcal{R})} - \mathcal{Y}^{(+\mathcal{R})}\|_F^2 \\ &= \frac{1}{2}\|\tilde{\mathcal{Y}}^{(+\mathcal{R})}\|_F^2 + \frac{1}{2}\|\mathbf{A}_{\mathcal{R}}^{(n)}\, \mathbf{A}_{\mathcal{R}}^{\odot -n\,T}\|_F^2 - \langle \tilde{\mathbf{Y}}_{(n)}^{(+\mathcal{R})}, \mathbf{A}_{\mathcal{R}}^{(n)} \mathbf{A}_{\mathcal{R}}^{\odot -n\,T} \rangle \\ &= \frac{1}{2}\|\tilde{\mathcal{Y}}^{(+\mathcal{R})}\|_F^2 + \frac{1}{2}\text{vec}\left(\mathbf{A}_{\mathcal{R}}^{(n)T}\right)^T \left(\mathbf{I}_{I_n} \otimes \mathbf{\Gamma}_{\mathcal{R}}^{(n)}\right) \text{vec}\left(\mathbf{A}_{\mathcal{R}}^{(n)T}\right) - \text{vec}\left(\mathbf{\Phi}_{\mathcal{R}}^{(n)T}\right)^T \text{vec}\left(\mathbf{A}_{\mathcal{R}}^{(n)T}\right), \end{aligned} \tag{3.25}$$

where $\mathbf{\Gamma}_{\mathcal{R}}^{(n)} = \{\mathbf{A}_{\mathcal{R}}^T \mathbf{A}_{\mathcal{R}}\}^{\circledast -n} \in \mathbb{R}^{R\times R}$, and $\mathbf{\Phi}_{\mathcal{R}}^{(n)} = \tilde{\mathbf{Y}}_{(n)}^{(+\mathcal{R})} \{\mathbf{A}\}_{\mathcal{R}}^{\odot -n} \in \mathbb{R}^{I_n \times R}$. While fixing the components $\boldsymbol{a}_r^{(n)} (r \notin \mathcal{R}, n = 1,2,\ldots,N)$ during the estimation process of factors $\mathbf{A}_{\mathcal{R}}^{(n)}$, the first term in (3.25) $\|\tilde{\boldsymbol{\mathcal{Y}}}\|_F^2$ is considered as constant, hence it can be ignored. The last two terms in (3.25) formulate a minimization problem in the form of Problem 3.1:

$$\begin{aligned} \min \quad & f\left(\mathfrak{a}^{(n)}\right) = \frac{1}{2}\mathfrak{a}^{(n)\,T}\mathbf{\Gamma}\,\mathfrak{a}^{(n)} - \mathrm{vec}\left(\mathbf{\Phi}_{\mathcal{R}}^{(n)T}\right)^T \mathfrak{a}^{(n)}, \\ \text{s.t.} \quad & \mathfrak{a}^{(n)} \geqslant \mathbf{0}. \end{aligned} \tag{3.26}$$

where $\mathbf{\Gamma} = \mathbf{I}_{I_n} \otimes \mathbf{\Gamma}_{\mathcal{R}}^{(n)} \in \mathbb{R}^{I_n R \times I_n R}$, and $\mathfrak{a}^{(n)} = \mathrm{vec}\left(\mathbf{A}_{\mathcal{R}}^{(n)\,T}\right)$. Construction of the residual tensor $\tilde{\boldsymbol{\mathcal{Y}}}^{(\mathcal{R})}$ could demand large-scale temporary extra-storage for the tensor $\boldsymbol{\mathcal{Y}}^{(-\mathcal{R})}$, and also high computational cost due to Khatri-Rao products. To significantly reduce the complexity, the matrix $\mathbf{\Phi}^{(n)}$ is constructed without building up the tensors $\tilde{\boldsymbol{\mathcal{Y}}}^{(+\mathcal{R})}$ as well as $\boldsymbol{\mathcal{Y}}^{(-\mathcal{R})}$ as follows

$$\mathbf{\Phi}_{\mathcal{R}}^{(n)} = \mathbf{Y}_{(n)}\mathbf{A}_{\mathcal{R}}^{\odot -n} - \mathbf{A}_{\bar{\mathcal{R}}}^{(n)} \left\{\mathbf{A}_{\bar{\mathcal{R}}}^T \mathbf{A}_{\mathcal{R}}\right\}^{\circledast -n}, \tag{3.27}$$

where $\bar{\mathcal{R}} = \{1:R\}\backslash\mathcal{R}$, $\mathbf{A}_{\bar{\mathcal{R}}}^{(n)}$ is a subset of the factor $\mathbf{A}^{(n)}$ ignored the components $\mathbf{A}_{\mathcal{R}}^{(n)}$.

All the components $\mathbf{A}_{\mathcal{R}}^{(n)}$ will be updated using Algorithm 3.1. Such procedure will be replicated for different subsets $\mathcal{R}$ until each component $\boldsymbol{a}_r^{(n)}\ \forall n, \forall r$ should be updated at least once. Pseudo-code of the proposed algorithm is listed in Algorithm 3.6. The function `nqp` refers to as Algorithm 3.1. We note that when $\tilde{R} = 1$, the proposed algorithm becomes the HALS algorithm presented in Chapter 1 which updates one component $\boldsymbol{a}_r^{(n)}$.

In each loop to estimate the factor $\mathbf{A}^{(n)}$ (Steps 5-10), a set of column indices $\mathcal{R}$ will be selected in Step 6. Normally, this selection step will be replicated until all the components should be updated at least one time. A subset $\mathcal{R}$ is an $\tilde{R}$-subset on R elements. Therefore, there are in total $C_R^{\tilde{R}}$ possible combinations of selection $\mathcal{R}$. A simple approach is to proceed the estimation over all the subsets $\mathcal{R}$. This procedure demands high computational cost, but could improve the performance. Dividing the set of R indices $\{1,2,\ldots,R\}$ into subsets $\mathcal{R}$ with empty intersections or as least as possible is a suggested strategy which has low computational cost. we can select $\tilde{R}$ highly correlated components. An alternative strategy is that for each component index r, we sequentially select $K = \min(\tilde{R}, R-\tilde{R})$ consecutive indices $\mathcal{R} = \{r, r+1, \ldots, r+K-1\}$.

3.6 Summary

Novel algorithms for NTF and NTD have been proposed in this chapter based on recursively solving the nonnegative quadratic programming problems. A variation of the proposed algorithm which is suitable for low memory or parallel computing is also presented. Moreover, we derived an flexible algorithm (rK-QALS) which sequentially updates a subset of $1 \leqslant \tilde{R} \leqslant R$ components of factors. For

$\tilde{R} = 1$, the rK-QALS algorithm simplifies to the HALS algorithm[37]. Adaptive choice behavior of the number of components being updated in the rK-QALS algorithm is a possible future work. The performances of the proposed algorithm are verified for difficult synthetic benchmarks and also for clustering and classification problems in Chapter 6. Our algorithm not only works well for dense data, but also for sparse data without any additional regularization parameter as other algorithms. Especially, the proposed algorithm copes with highly collinear factors. The proposed algorithm for NQP can be applied to any applications involving nonnegative least squares approximation.

CHAPTER 4

All-at-Once Algorithms for Tensor Decompositions

4.1 All-at-Once Algorithms for CP

Chapter 2 presented the ALS algorithm and proposed a variant of this algorithm which updates components sequentially. We call it the HALS algorithm which can work for CP with and without non-negative and other constraints. However, alternating least squares algorithms often fail for data with different magnitudes of factors[143], or collinearity of factors, such as bottleneck when two or more components are collinear[51;77;171], or swamps in which a bottleneck exists in all modes[25;51;127;172]. Alternative least squares (ALS) algorithms with line searches, regularization, rotation can improve performance, but they do not completely solve the problems. All-at-once algorithms which simultaneously update all the factors $\mathbf{A}^{(n)}$ instead of the alternating least square methods are expected to cope with this problem[4;142;143;176;196;202;203]. Nevertheless, those algorithms are computationally demanding due to construction of gradients and Hessians with respect to all the entries of the factors. Acar *et al.*[4] presented the OPT algorithm in the Matlab Tensor toolbox[17] which fits a CP model to a tensor via optimization. The OPT algorithm[4;17] employs the nonlinear conjugate gradient (NCG) method with Fletcher-Reeves, Polak-Ribière, and Hestenes-Stiefel updates[185],

$$\mathfrak{a} \leftarrow \mathfrak{a} - \eta \frac{\partial D}{\partial \mathfrak{a}}, \tag{4.1}$$

where $\mathfrak{a}$ is vectorization of all factors $\mathbf{A}^{(n)}$ defined in (1.25), η is the step size.

An alternative approach to minimize the cost function is the damped Gauss-Newton (dGN) method with update rule given by

$$\boxed{\mathfrak{a} \leftarrow \mathfrak{a} + \left(\mathbf{J}^T \mathbf{J} + \mu \mathbf{I}_{RT}\right)^{-1} \mathbf{J}^T \left(\underset{\sim}{y} - \underset{\sim}{\hat{y}}\right),} \tag{4.2}$$

where $\underset{\sim}{y} = \text{vec}(\boldsymbol{\mathcal{Y}})$, $\underset{\sim}{\hat{y}} = \text{vec}\left(\hat{\boldsymbol{\mathcal{Y}}}\right)$, $\mathbf{J} \in \mathbb{R}^{(\prod_n I_n) \times (RT)}$ $(T = \sum_n I_n)$ is the Jacobian of $\text{vec}\left(\hat{\boldsymbol{\mathcal{Y}}}\right)$ with respect to $\mathfrak{a}$, and the damping parameter $\mu > 0$. Paatero[143] emphasized advantage of dGN compared with ALS when dealing with problems regarding swamps, different magnitudes of factors. We note that the approximate Hessian $\mathbf{H} = \mathbf{J}^T \mathbf{J}$ is rank-deficient[143;144;200;203]. To deal with this problem and to improve convergence and stability of the algorithm, the Jacobian can be forced to be full rank by

adding additional rows as proposed by Bini and Boito[22]. An alternative approach is to employ the dGN iteration with a damping parameter μ.

The Gauss-Newton algorithm can be derived from Newton's method. Hence, the rate of convergence of the update rule (4.2) is at most quadratic. However, these methods face problems involving the large-scale Jacobian and large-scale inverse of the approximate Hessian $\mathbf{H} = \mathbf{J}^T\mathbf{J} \in \mathbb{R}^{RT\times RT}, T = \sum_n I_n$. In order to eliminate the Jacobian, Paatero[143] established explicit expressions for submatrices of $\mathbf{H}$. We note that inverse of $\mathbf{H}$ is the largest workload of the GN algorithm with a complexity of order $O(N^3R^3I^3)$ for factorization of a tensor with $I_n = I, \forall n$. Paatero[143] solved the inverse problem $\mathbf{H}^{-1}$ by Cholesky decomposition of the approximate Hessian and back substitution. However, the algorithm is of order $O(10/3R^3I^3)$ for 3-way symmetric tensor factorization $I_n = I, \forall n$, and still computationally demanding. Tomasi[200] extended Paatero's results[143], and derived a convenient method to construct $\mathbf{H}$ and the gradient for N-way tensor without using the Jacobian. In order to cope with inverse of $\mathbf{H}$, Tomasi[201] used QR decomposition. However, existing dGN algorithms are still not sufficiently efficient for the large-scale inverse problem $\mathbf{H}^{-1}$.

Recently, Tichavský and Koldovský[196] have proposed a novel method to compute inverse of Hessian based on $3R^2 \times 3R^2$ dimensional matrices. For low-rank approximation $R \ll I_n, \forall n$, this method dramatically improves the running time. However, the algorithms still demand significant temporary extra-storage, and high computational cost due to employment of Kronecker products, and it is restricted for third-order tensors.

Another approach is to consider the CP decomposition as a joint diagonalization problem[56;111;117;173]. However, the method will not be discused in this chapter.

This chapter will derive a suite of low cost algorithms to factorize real and complex-valued tensors with/without nonnegative constraints based on damped Gauss-Newton (dGN or LM) iterations with convenient computations for the approximate Hessian and gradients. Especially, for low-rank tensor approximation, we introduce fast dGN algorithms without building up huge approximate Hessians, and also with elimination of Kronecker products which are often used in CP algorithms. The proposed algorithms are verified to overwhelmingly outperform "state-of-the-art" algorithms for difficult benchmarks with bottlenecks, swamps for both real and complex-valued tensors.

4.2 Damped Gauss-Newton Algorithm

In this section, we will derive a fast dGN algorithm for low-rank approximation of tensors with arbitrary dimensions. The most important challenge of the update rule (4.2) is to reduce the computational cost for evaluation of the approximate Hessian $\mathbf{H}$ and its inverse. We derive a low rank adjustment for the approximate Hessian and employ the binomial inverse theorem[94] to inverse an $NR^2 \times NR^2$ ma-

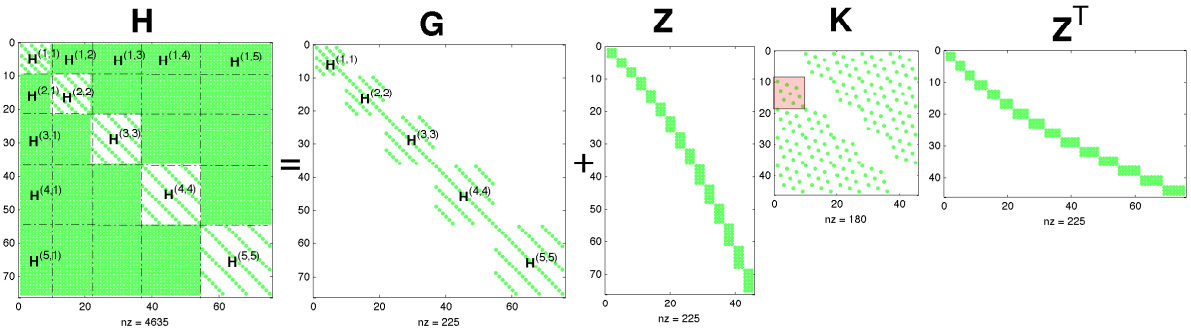


Figure 4.1: Illustration of the approximate Hessian for a 5-D tensor which can be expressed as a low rank adjustment $\mathbf{H} = \mathbf{G} + \mathbf{Z}\,\mathbf{K}\,\mathbf{Z}^T$ as in Theorem 4.1. Green dots indicate nonzero elements.

trix which is much smaller than $\mathbf{H} \in \mathbb{R}^{R\sum_n I_n \times R\sum_n I_n}$. This result allows formulating a low complexity update rule equivalent to the dGN rule (4.2).

4.2.1 Low-Rank Adjustment for Approximate Hessian

Theorem 4.1 (Low rank Adjustment for approximate Hessian **H**). *The approximate Hessian* **H** *can be decomposed as*

$$\mathbf{H} = \mathbf{G} + \mathbf{Z}\,\mathbf{K}\,\mathbf{Z}^T, \tag{4.3}$$

where **G** *is an invertible block diagonal matrix,* **Z** *is a block diagonal matrix consisting of* $N\,R^2$ *columns, and kernel matrix* **K** *is an* $N \times N$ *block of matrices* $\mathbf{K}^{(n,m)}$

$$\mathbf{G} = \mathtt{blkdiag}\left(\mathbf{\Gamma}^{(n,n)} \otimes \mathbf{I}_{I_n}\right)_{n=1}^{N}, \tag{4.4}$$

$$\mathbf{Z} = \mathtt{blkdiag}\left(\mathbf{I}_R \otimes \mathbf{A}^{(n)}\right)_{n=1}^{N}, \tag{4.5}$$

$$\mathbf{K}^{(n,m)} = (1-\delta_{n,m})\,\mathbf{P}_{R,R}\;\mathrm{diag}\left(\mathrm{vec}\left(\mathbf{\Gamma}^{(n,m)}\right)\right), \tag{4.6}$$

where $\delta_{n,m}$ *is the Kronecker delta. The permutation matrix* $\mathbf{P}_{R,R}$ *is defined for an* $R \times R$ *matrix (see Appendix A4.1), and* $\mathbf{\Gamma}^{(n,m)}$ *are symmetric matrices defined as*

$$\mathbf{\Gamma}^{(n,m)} = \left[\mathbf{\Gamma}^{(n,m)}\right]^T = \left[\mathbf{\Gamma}^{(m,n)}\right]^T = \circledast_{k\neq n,m}\,\mathbf{C}^{(k)}, \qquad \mathbf{C}^{(n)} = \mathbf{A}^{(n)T}\mathbf{A}^{(n)}. \tag{4.7}$$

Proof of Theorem 4.1 follows from Theorems presented in the subsequent sections. In Figure 4.1, we illustrate an example of the approximate Hessian for a $3\times4\times5\times6\times7$ dimensional tensor composed by 5 factors of 3 components. The approximate Hessian in the left hand side of Figure 4.1 consists of $(N(N-1))R^2$ rank-one matrices and NR^2 diagonal matrices located along the main diagonal of **H**.

In order to prove Theorem 4.1, we seek for explicit expressions for the Jacobian and the approximate Hessian. Partial or similar results have been shown by Paatero[144] and Tomasi[200]. However, our purpose and those of Paatero[144] and Tomasi[200] are quite different. Both Paatero[144] and Tomasi[200] introduced explicit expressions for submatrices of size $I_n \times I_m$ or $I_n \times I_n$ of the approximate Hessian in

order to bypass the large-scale Jacobian $\mathbf{J}$ and to establish fast computation of $\mathbf{H}$, but they have not successfully solved the large-scale inverse problem $\mathbf{H}^{-1}$. In this section, our aim is to deal with this hard problem. Moreover, due to different vectorization of factors and to the chapter being self-contained, the results for the approximate Hessian and the Jacobian need to be reinvestigated.

4.2.1.1 Explicit Expression for Approximate Hessian

In order to compute the Jacobian $\mathbf{J}$, and $\mathbf{H}$, we consider the commutation matrix $\mathbf{P}_n$ of size $\prod_{n=1}^{N} I_n \times \prod_{n=1}^{N} I_n$ mapping $\text{vec}(\mathcal{Y}) = \mathbf{P}_n \text{ vec}\big(\mathbf{Y}_{(n)}\big)$ and defined in Lemma 4.1. This definition is related to the commutation matrix defined by Magnus and Neudecker[120], and considered by Tomasi in[200]. However, they are not the same. Moreover, explicit expression of $\mathbf{P}_n$ is presented in Appendix A4.2.

From (1.24) and Lemma 4.1, the gradient of the vector $\hat{\mathcal{Y}}$ with respect to each component $\boldsymbol{a}_r^{(n)}$, for $n = 1, 2, \ldots, N$ and $r = 1, 2, \ldots, R$ is computed as

$$\frac{\partial \hat{\mathcal{Y}}}{\partial \boldsymbol{a}_r^{(n)}} = \frac{\partial \text{vec}\big(\hat{\mathcal{Y}}\big)}{\partial \boldsymbol{a}_r^{(n)}} = \mathbf{P}_n \frac{\partial \text{vec}\big(\hat{\mathbf{Y}}_{(n)}\big)}{\partial \boldsymbol{a}_r^{(n)}} = \mathbf{P}_n \frac{\partial \sum_{r=1}^{R} \boldsymbol{a}_r^{(n)} \left(\bigodot_{k\neq n} \boldsymbol{a}_r^{(k)}\right)^T}{\partial \boldsymbol{a}_r^{(n)}} = \mathbf{P}_n \left(\bigodot_{k\neq n} \boldsymbol{a}_r^{(k)} \otimes \mathbf{I}_{I_n}\right), \quad (4.8)$$

where $\mathbf{I}_{I_n}$ is an $I_n \times I_n$ identity matrix. As a consequence, the Jacobian matrix $\mathbf{J}$ has a form of[143;202]

$$\begin{aligned} \mathbf{J} = \frac{\partial \hat{\mathcal{Y}}}{\partial \mathfrak{a}} &= \left[\frac{\partial \hat{\mathcal{Y}}}{\partial \boldsymbol{a}_1^{(1)}} \cdots \frac{\partial \hat{\mathcal{Y}}}{\partial \boldsymbol{a}_R^{(1)}} \cdots \frac{\partial \hat{\mathcal{Y}}}{\partial \boldsymbol{a}_1^{(n)}} \cdots \frac{\partial \hat{\mathcal{Y}}}{\partial \boldsymbol{a}_R^{(n)}} \cdots \frac{\partial \hat{\mathcal{Y}}}{\partial \boldsymbol{a}_1^{(N)}} \cdots \frac{\partial \hat{\mathcal{Y}}}{\partial \boldsymbol{a}_R^{(N)}} \right] \\ &= \left[\mathbf{P}_1 \left(\left(\bigodot_{k\neq 1} \mathbf{A}^{(k)} \right) \otimes \mathbf{I}_{I_1} \right) \cdots \mathbf{P}_n \left(\left(\bigodot_{k\neq n} \mathbf{A}^{(k)} \right) \otimes \mathbf{I}_{I_n} \right) \cdots \mathbf{P}_N \left(\left(\bigodot_{k\neq N} \mathbf{A}^{(k)} \right) \otimes \mathbf{I}_{I_N} \right) \right]. \end{aligned} \quad (4.9)$$

We express the approximate Hessian $\mathbf{H}$ as an $N \times N$ block matrix, and establish the explicit expression for $\mathbf{H}$

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}^{(1,1)} & \cdots & \mathbf{H}^{(1,m)} & \cdots & \mathbf{H}^{(1,N)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{H}^{(n,1)} & \cdots & \mathbf{H}^{(n,m)} & \cdots & \mathbf{H}^{(n,N)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{H}^{(N,1)} & \cdots & \mathbf{H}^{(N,m)} & \cdots & \mathbf{H}^{(N,N)} \end{bmatrix}, \text{with } \mathbf{H}^{(n,m)} = \begin{bmatrix} \mathbf{H}_{1,1}^{(n,m)} & \cdots & \mathbf{H}_{r,1}^{(n,m)} & \cdots & \mathbf{H}_{R,1}^{(n,m)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{H}_{1,s}^{(n,m)} & \cdots & \mathbf{H}_{r,s}^{(n,m)} & \cdots & \mathbf{H}_{r,s}^{(n,m)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{H}_{1,R}^{(n,m)} & \cdots & \mathbf{H}_{r,R}^{(n,m)} & \cdots & \mathbf{H}_{R,R}^{(n,m)} \end{bmatrix}, \quad (4.10)$$

where blocks $\mathbf{H}_{r,s}^{(n,m)} \in \mathbb{R}^{RI_n \times RI_m}$ is defined as

$$\mathbf{H}_{r,s}^{(n,m)} = \left(\frac{\partial \hat{\mathcal{Y}}}{\partial \boldsymbol{a}_r^{(n)}}\right)^T \left(\frac{\partial \hat{\mathcal{Y}}}{\partial \boldsymbol{a}_s^{(m)}}\right), \qquad (n, m = 1, 2, \ldots, N, \quad r, s = 1, 2, \ldots, R). \quad (4.11)$$

The purpose herein is to establish the explicit expressions for block matrices $\mathbf{H}_{r,s}^{(n,m)}$, for $\forall n, \forall m$, and $\forall r, \forall s$.

Theorem 4.2. *A submatrix* $\mathbf{H}_{r,s}^{(n,m)}$ *is a rank-one or diagonal matrix given by*

$$\mathbf{H}_{r,s}^{(n,m)} = \delta_{n,m}\,\gamma_{rs}^{(n,n)}\,\mathbf{I}_{I_n} + (1-\delta_{n,m})\,\gamma_{r\,s}^{(n,m)}\,\boldsymbol{a}_r^{(n)}\,\boldsymbol{a}_s^{(m)\,T}, \tag{4.12}$$

where $\gamma_{rs}^{(n,m)}$ *is the* (r,s) *entry of the symmetric matrix* $\boldsymbol{\Gamma}^{(n,m)}$ *defined in (4.7).*

Proof. From (4.8) and (4.11), submatrices $\mathbf{H}_{r,s}^{(n,n)}$ can be quickly expressed as

$$\mathbf{H}_{r,s}^{(n,n)} = \left(\left(\bigodot_{k\neq n}\boldsymbol{a}_r^{(k)}\right)^T \otimes \mathbf{I}_{I_n}\right)\mathbf{P}_n^T\,\mathbf{P}_n\left(\bigodot_{k\neq n}\boldsymbol{a}_s^{(k)}\otimes\mathbf{I}_{I_n}\right) = \left(\prod_{k\neq n}\boldsymbol{a}_r^{(k)\,T}\,\boldsymbol{a}_s^{(k)}\right)\mathbf{I}_{I_n} = \gamma_{rs}^{(n,n)}\,\mathbf{I}_{I_n}.$$

The gradient in (4.8) can be rewritten as concatenation of Kronecker products given by

$$\frac{\partial\hat{\boldsymbol{y}}}{\partial\boldsymbol{a}_r^{(n)}} = \mathbf{P}_n\left[\bigodot_{k\neq n}\boldsymbol{a}_r^{(k)}\otimes\boldsymbol{e}_1^{(n)}\;\bigodot_{k\neq n}\boldsymbol{a}_r^{(k)}\otimes\boldsymbol{e}_2^{(n)}\;\cdots\;\bigodot_{k\neq n}\boldsymbol{a}_r^{(k)}\otimes\boldsymbol{e}_{I_n}^{(n)}\right] = \left[\mathbf{P}_n\left(\bigodot_{k\neq n}\boldsymbol{a}_r^{(k)}\otimes\boldsymbol{e}_{i_n}^{(n)}\right)\right]_{i_n=1}^{I_n},$$

where unit vector $\boldsymbol{e}_{i_n}^{(n)}$ for $i_n = 1,2,\ldots,I_n$ is the i_n-th column of the identity matrix $\mathbf{I}_{I_n}\in\mathbb{R}^{I_n\times I_n}$.

Based on Theorem 4.13, an (i_n,i_m) entry of a sub matrix $\mathbf{H}_{r,s}^{(n,m)}$ for $n\neq m$ and $i_n = 1,2,\ldots,I_n$, and $i_m = 1,2,\ldots,I_m$ is calculated by

$$\begin{aligned}
\left[\mathbf{H}_{r,s}^{(n,m)}\right]_{i_n,i_m} &= \left[\frac{\partial\hat{\boldsymbol{y}}}{\partial\boldsymbol{a}_r^{(n)}}\right]_{i_n}^T\left[\frac{\partial\hat{\boldsymbol{y}}}{\partial\boldsymbol{a}_s^{(m)}}\right]_{i_m} = \left(\left(\bigodot_{k\neq n}\boldsymbol{a}_r^{(k)}\right)^T\otimes\boldsymbol{e}_{i_n}^{(n)\,T}\right)\mathbf{P}_n^T\,\mathbf{P}_m\left(\bigodot_{k\neq m}\boldsymbol{a}_s^{(k)}\otimes\boldsymbol{e}_{i_m}^{(m)}\right)\\
&= \left(\boldsymbol{a}_r^{(N)\,T}\otimes\cdots\otimes\boldsymbol{a}_r^{(n+1)\,T}\otimes\boldsymbol{e}_{i_n}^{(n)\,T}\otimes\boldsymbol{a}_r^{(n-1)\,T}\otimes\cdots\otimes\boldsymbol{a}_r^{(1)\,T}\right)\\
&\quad\left(\boldsymbol{a}_s^{(N)}\otimes\cdots\otimes\boldsymbol{a}_s^{(m+1)}\otimes\boldsymbol{e}_{i_m}^{(m)}\otimes\boldsymbol{a}_s^{(m-1)}\otimes\cdots\otimes\boldsymbol{a}_s^{(1)}\right)\\
&= \left(\prod_{k\neq n,m}\left(\boldsymbol{a}_r^{(k)\,T}\,\boldsymbol{a}_s^{(k)}\right)\right)\left(\left(\boldsymbol{a}_r^{(m)\,T}\,\boldsymbol{e}_{i_m}^{(m)}\right)\otimes\left(\boldsymbol{e}_{i_n}^{(n)\,T}\,\boldsymbol{a}_s^{(n)}\right)\right) = \gamma_{r\,s}^{(n,m)}\,a_{i_m\,r}^{(m)}\,a_{i_n\,s}^{(n)}.
\end{aligned} \tag{4.13}$$

This leads to a compact formula of a block matrix $\mathbf{H}_{r,s}^{(n,m)}$ for $n\neq m$ given in (4.12). □

The result in Theorem 4.2 is somewhat similar to those introduced by Paatero[143] for 3-way tensors, and by Tomasi[200] for N-way tensors. Both Paatero[143] and Tomasi[200] introduced the results for the fast computation of the approximate Hessian and to bypass the large-scale Jacobian $\mathbf{J}$, but they have not employed these structures of the approximate Hessian to overcome the problem of large-scale inverse problem. We reinvestigated the results in Theorem 4.2 in order to establish an explicit expression for the whole approximate Hessian as a low rank adjustment given in Theorem 4.1.

Theorem 4.3. *A submatrix* $\mathbf{H}^{(n,m)}$ $(\forall n,\forall m)$ *has an explicit expression given by*

$$\mathbf{H}^{(n,m)} = \delta_{n,m}\left(\boldsymbol{\Gamma}^{(n,n)}\otimes\mathbf{I}_{I_n}\right) + \left(\mathbf{I}_R\otimes\mathbf{A}^{(n)}\right)\mathbf{K}^{(n,m)}\left(\mathbf{I}_R\otimes\mathbf{A}^{(m)\,T}\right). \tag{4.14}$$

Proof. From (4.12), a diagonal block matrix $\mathbf{H}^{(n,n)}$ is expressed as

$$\mathbf{H}^{(n,n)} = \begin{bmatrix} \gamma_{1,1}^{(n,n)}\mathbf{I}_{I_n} & \cdots & \gamma_{r,1}^{(n,n)}\mathbf{I}_{I_n} & \cdots & \gamma_{R,1}^{(n,n)}\mathbf{I}_{I_n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \gamma_{1,s}^{(n,n)}\mathbf{I}_{I_n} & \cdots & \gamma_{r,s}^{(n,n)}\mathbf{I}_{I_n} & \cdots & \gamma_{r,s}^{(n,n)}\mathbf{I}_{I_n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \gamma_{1,R}^{(n,n)}\mathbf{I}_{I_n} & \cdots & \gamma_{r,R}^{(n,n)}\mathbf{I}_{I_n} & \cdots & \gamma_{R,R}^{(n,n)}\mathbf{I}_{I_n} \end{bmatrix} = \boldsymbol{\Gamma}^{(n,n)} \otimes \mathbf{I}_{I_n}. \tag{4.15}$$

From (4.12), by employing (4.95), we vertically concatenate all $\mathbf{H}_{r,s}^{(n,m)}$ $(n \neq m)$ which have the same index r in the same column, then horizontally concatenate the resulting blocks by using (4.98). The final expression for the (n,m) block matrix $\mathbf{H}^{(n,m)}$ can be written as

$$\begin{aligned} \mathbf{H}^{(n,m)} &= \begin{bmatrix} \gamma_{1,1}^{(n,m)}\boldsymbol{a}_1^{(n)}\boldsymbol{a}_1^{(m)\,T} & \cdots & \gamma_{r,1}^{(n,m)}\boldsymbol{a}_r^{(n)}\boldsymbol{a}_1^{(m)\,T} & \cdots & \gamma_{R,s}^{(n,m)}\boldsymbol{a}_R^{(n)}\boldsymbol{a}_1^{(m)\,T} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \gamma_{1,s}^{(n,m)}\boldsymbol{a}_1^{(n)}\boldsymbol{a}_s^{(m)\,T} & \cdots & \gamma_{r,s}^{(n,m)}\boldsymbol{a}_r^{(n)}\boldsymbol{a}_s^{(m)\,T} & \cdots & \gamma_{R,s}^{(n,m)}\boldsymbol{a}_R^{(n)}\boldsymbol{a}_s^{(m)\,T} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \gamma_{1,R}^{(n,m)}\boldsymbol{a}_1^{(n)}\boldsymbol{a}_R^{(m)\,T} & \cdots & \gamma_{r,R}^{(n,m)}\boldsymbol{a}_r^{(n)}\boldsymbol{a}_R^{(m)\,T} & \cdots & \gamma_{R,R}^{(n,m)}\boldsymbol{a}_R^{(n)}\boldsymbol{a}_R^{(m)\,T} \end{bmatrix} \\ &= \begin{bmatrix} \left(\mathbf{I}_R \otimes \boldsymbol{a}_1^{(n)}\right)\mathbf{D}_1^{(n,m)}\mathbf{A}^{(n)\,T} & \cdots & \left(\mathbf{I}_R \otimes \boldsymbol{a}_r^{(n)}\right)\mathbf{D}_r^{(n,m)}\mathbf{A}^{(n)\,T} & \cdots & \left(\mathbf{I}_R \otimes \boldsymbol{a}_R^{(n)}\right)\mathbf{D}_R^{(n,m)}\mathbf{A}^{(n)\,T} \end{bmatrix} \\ &= \begin{bmatrix} \left(\mathbf{I}_R \otimes \boldsymbol{a}_1^{(n)}\right) & \cdots & \left(\mathbf{I}_R \otimes \boldsymbol{a}_r^{(n)}\right) & \cdots & \left(\mathbf{I}_R \otimes \boldsymbol{a}_R^{(n)}\right) \end{bmatrix} \mathbf{D}^{(n,m)} \left(\mathbf{I}_R \otimes \mathbf{A}^{(m)\,T}\right) \\ &= \left(\mathbf{I}_R \otimes \mathbf{A}^{(n)}\right)\mathbf{P}_{R,R}\,\mathbf{D}^{(n,m)}\left(\mathbf{I}_R \otimes \mathbf{A}^{(m)\,T}\right), \end{aligned} \tag{4.16}$$

where $\mathbf{D}_r^{(n,m)} = \operatorname{diag}\left(\boldsymbol{\gamma}_r^{(n,m)}\right)$ is a diagonal matrix whose diagonal entries are the r-th row of $\boldsymbol{\Gamma}^{(n,m)}$, and $\mathbf{D}^{(n,m)} = \operatorname{diag}\left(\operatorname{vec}\left(\boldsymbol{\Gamma}^{(n,m)}\right)\right)$. □

By establishing expressions for submatrices $\mathbf{H}^{(n,m)}$, we can straightforwardly prove Theorem 4.1.

Proof. (Theorem 4.1) From (4.14), we construct a sparse matrix $\mathbf{G}$ consisting all block matrices $\mathbf{H}^{(n,n)}$ locating along the diagonal of the approximate Hessian (4.10), that is

$$\mathbf{G} = \texttt{blkdiag}\left(\mathbf{H}^{(n,n)}\right)_{n=1}^{N} = \texttt{blkdiag}\left(\boldsymbol{\Gamma}^{(n,n)} \otimes \mathbf{I}_{I_n}\right)_{n=1}^{N}. \tag{4.17}$$

From Theorem 4.3, and by using the product of block matrices, the difference $\mathbf{H} - \mathbf{G}$ can be straightforwardly decomposed into three matrices defined in Theorem 4.1 as

$$\mathbf{H} - \mathbf{G} = \mathbf{Z}\,\mathbf{K}\,\mathbf{Z}^T. \tag{4.18}$$

This completes the proof of Theorem 4.1. □

4.2.2 Fast Inverse of the Approximate Hessian H

Theorem 4.1 allows inversion of the large approximate Hessian via a much smaller matrix[94]

$$\mathbf{H}_\mu^{-1} = (\mathbf{H} + \mu\,\mathbf{I}_T)^{-1} = \left(\mathbf{G}_\mu + \mathbf{Z}\,\mathbf{K}\,\mathbf{Z}^T\right)^{-1} = \mathbf{G}_\mu^{-1} - \mathbf{G}_\mu^{-1}\,\mathbf{Z}\mathbf{B}_\mu\,\mathbf{Z}^T\,\mathbf{G}_\mu^{-1}, \tag{4.19}$$

with

$$\mathbf{B}_\mu = \left(\mathbf{K}^{-1} + \mathbf{Z}^T\,\mathbf{G}_\mu{}^{-1}\,\mathbf{Z}\right)^{-1}, \tag{4.20}$$

if $\mathbf{K}$ is invertible, or in an alternative form

$$\mathbf{B}_\mu = \mathbf{K}\,\left(\mathbf{I}_{NR^2} + \mathbf{Z}^T\,\mathbf{G}_\mu^{-1}\,\mathbf{Z}\,\mathbf{K}\right)^{-1}, \tag{4.21}$$

where $\mathbf{G}_\mu = \mathbf{G} + \mu\,\mathbf{I}_T = \texttt{blkdiag}\left(\left(\mathbf{\Gamma}^{(n,n)} + \mu\,\mathbf{I}_R\right) \otimes \mathbf{I}_{I_n}\right)_{n=1}^{N}$, and its inverse can be efficiently computed from $\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)} = \mathbf{\Gamma}_\mu^{(n,n)\,-1} = \left(\mathbf{\Gamma}^{(n,n)} + \mu\,\mathbf{I}_R\right)^{-1}$

$$\widetilde{\mathbf{G}}_\mu = \mathbf{G}_\mu^{-1} = \texttt{blkdiag}\left(\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)} \otimes \mathbf{I}_{I_n}\right)_{n=1}^{N}. \tag{4.22}$$

The following results are used to inverse the approximate Hessian in (4.19)

$$\mathbf{L}_\mu = \mathbf{G}_\mu^{-1}\,\mathbf{Z} = \texttt{blkdiag}\left(\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)} \otimes \mathbf{A}^{(n)}\right)_{n=1}^{N}, \tag{4.23}$$

$$\mathbf{\Psi} = \mathbf{Z}^T\,\mathbf{G}_\mu^{-1}\,\mathbf{Z} = \texttt{blkdiag}\left(\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)} \otimes \mathbf{C}^{(n)}\right)_{n=1}^{N}. \tag{4.24}$$

The matrix $\mathbf{K}$ can also be expressed as a product of the permutation matrix $\mathbf{P}_{R,R}$ and a partitioned matrix of matrices $\mathbf{D}^{(n,m)} = (1-\delta_{n,m})\,\text{diag}\left(\text{vec}\left(\mathbf{\Gamma}^{(n,m)}\right)\right)$

$$\mathbf{K} = (\mathbf{I}_N \otimes \mathbf{P}_{R,R})\,\left[\mathbf{D}^{(n,m)}\right]_{n,m}. \tag{4.25}$$

If all the entries $\gamma_{r,s}^{(n,m)}$ are non-zeros, the matrix $\mathbf{D}$ is invertible, and its inverse is also a partitioned matrix comprising diagonal matrices. Proof is briefly described in Appendix A4.3.

4.2.3 Fast dGN Algorithm

From dGN rule (4.2), we can replace $\mathbf{H}_\mu^{-1}$ by those in (4.19) to formulate the fast dGN algorithm

$$\mathfrak{a} \leftarrow \mathfrak{a} + \widetilde{\mathbf{G}}_\mu\mathbf{J}^T\,\left(\mathcal{y} - \hat{\mathcal{y}}\right) - \mathbf{L}_\mu\,\mathbf{B}_\mu\,\mathbf{L}_\mu^T\,\mathbf{J}^T\,\left(\mathcal{y} - \hat{\mathcal{y}}\right). \tag{4.26}$$

The Jacobian still exists in the update rule (4.26), and it could demand high computational cost. We also note that $\mathbf{B}_\mu$ is inverse of an $NR^2 \times NR^2$ matrix given in (4.20) or (4.21), and $\mathbf{L}_\mu$ is a block diagonal matrix of N Kronecker products $\left(\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)} \otimes \mathbf{A}^{(n)}\right) \in \mathbb{R}^{RI_n \times R^2}$ given in (4.23). Construction of $\mathbf{L}_\mu$ might have a computational complexity of order $O\left(T\,R^3\right)$, and requires an extra-storage of $O\left(TR^3\right)$. In order to completely bypass the Jacobian $\mathbf{J}$ in (4.26), avoid building up the matrix $\mathbf{L}_\mu$, we seek for convenient methods for computing $\widetilde{\mathbf{G}}_\mu\mathbf{J}^T\,\left(\mathcal{y} - \hat{\mathcal{y}}\right)$, $\boldsymbol{w} = \mathbf{L}_\mu^T\,\mathbf{J}^T\,\left(\mathcal{y} - \hat{\mathcal{y}}\right)$, and product $\mathbf{L}_\mu\,\mathbf{B}_\mu\,\boldsymbol{w}$.

4.2.3.1 Elimination of Jacobian

From (4.9), (4.22), projection of $\mathcal{E} = \mathcal{Y} - \hat{\mathcal{Y}}$ onto the matrix $\mathbf{G}_\mu{}^{-1}\,\mathbf{J}^T$ can be expressed as

$$\begin{aligned}
\left(\widetilde{\mathbf{G}}_\mu\,\mathbf{J}^T\,\mathrm{vec}(\mathcal{E})\right)^T &= \left[\mathrm{vec}(\mathcal{E})^T\,\mathbf{P}_n\left(\left(\left(\bigodot_{k\neq n}\mathbf{A}^{(k)}\right)\,\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)}\right)\otimes\mathbf{I}_{I_n}\right)\right]_{n=1}^{N}\\
&= \left[\mathrm{vec}\left(\mathbf{E}_{(n)}\left(\bigodot_{k\neq n}\mathbf{A}^{(k)}\right)\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)}\right)^T\right]_{n=1}^{N}\\
&= \left[\mathrm{vec}\left(\mathbf{Y}_{(n)}\left(\bigodot_{k\neq n}\mathbf{A}^{(k)}\right)\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)} - \mathbf{A}^{(n)}\left(\bigodot_{k\neq n}\mathbf{A}^{(k)}\right)^T\left(\bigodot_{k\neq n}\mathbf{A}^{(k)}\right)\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)}\right)^T\right]_{n=1}^{N}\\
&= \left[\mathrm{vec}\left(\mathbf{Y}_{(n)}\left(\bigodot_{k\neq n}\mathbf{A}^{(k)}\right)\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)} - \mathbf{A}^{(n)}\,\mathbf{\Gamma}^{(n,n)}\,\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)}\right)^T\right]_{n=1}^{N}\\
&= \left[\mathrm{vec}\left(\mathbf{A}_{ALS}^{(n)} - \hat{\mathbf{A}}_{ALS}^{(n)}\right)^T\right]_{n=1}^{N}, \qquad (4.27)
\end{aligned}$$

where factors $\mathbf{A}_{ALS}^{(n)} = \mathbf{Y}_{(n)}\left(\bigodot_{k\neq n}\mathbf{A}^{(k)}\right)\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)}$, and $\hat{\mathbf{A}}_{ALS}^{(n)} = \mathbf{A}^{(n)}\,\mathbf{\Gamma}^{(n,n)}\,\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)}$ are results of the damped rule (2.5)[51] for the tensors $\mathcal{Y}$ and $\hat{\mathcal{Y}}$ obtained from factors $\mathbf{A}^{(n)}$. Similarly, we have derived a convenient formula to compute the following projection

$$\begin{aligned}
\boldsymbol{w} = \mathbf{Z}^T\,\mathbf{G}_\mu{}^{-1}\,\mathbf{J}^T\,\mathrm{vec}(\mathcal{E}) &= \left[\mathrm{vec}\left(\mathbf{A}^{(n)\,T}\left(\mathbf{A}_{ALS}^{(n)} - \hat{\mathbf{A}}_{ALS}^{(n)}\right)\right)^T\right]_{n=1}^{N\quad T}\\
&= \mathrm{vec}\left(\left[\mathbf{A}^{(n)\,T}\mathbf{A}_{ALS}^{(n)} - \mathbf{\Gamma}\,\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)}\right]_{n=1}^{N}\right), \qquad (4.28)
\end{aligned}$$

where $\mathbf{\Gamma} = \circledast_{n=1}^{N}\mathbf{C}^{(n)}$.

4.2.3.2 Elimination of Kronecker Products and Large-scale Extra-storage for $\mathbf{L}_\mu$

This section copes with computational issues regarding to the Kronecker products in (4.23), and the matrix product $\mathbf{L}_\mu\,\boldsymbol{f}$ in which $\boldsymbol{f} = \mathbf{B}_\mu\boldsymbol{w}$. We denote a 3-D tensor $\mathcal{F} \in \mathbb{R}^{R\times R\times N}$ whose vectorization is the vector $\boldsymbol{f} \in \mathbb{R}^{NR^2}$, that is

$$\mathrm{vec}(\mathcal{F}) = \boldsymbol{f} = \mathbf{B}_\mu\boldsymbol{w}, \qquad (4.29)$$

or each $R \times R$ frontal slice $\mathbf{F}_n \in \mathbb{R}^{R\times R}$ is given by: $\mathrm{vec}(\mathbf{F}_n) = \left[f_{(n-1)R+1}\ \cdots\ f_{nR+1}\ \cdots\ f_{nR^2}\right]^T$, $n = 1, 2, \ldots, N$. Noting that $\left(\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)}\otimes\mathbf{A}^{(n)}\right)\,\mathrm{vec}(\mathbf{F}_n) = \mathrm{vec}\left(\mathbf{A}^{(n)}\,\mathbf{F}_n\,\widetilde{\mathbf{\Gamma}}_\mu^{(n,n)}\right)$, we obtain

$$\mathbf{L}_\mu \boldsymbol{f} = \begin{bmatrix} \operatorname{vec}\left(\mathbf{A}^{(1)} \mathbf{F}_1 \widetilde{\boldsymbol{\Gamma}}_\mu^{(1,1)}\right) \\ \vdots \\ \operatorname{vec}\left(\mathbf{A}^{(n)} \mathbf{F}_n \widetilde{\boldsymbol{\Gamma}}_\mu^{(n,n)}\right) \\ \vdots \\ \operatorname{vec}\left(\mathbf{A}^{(N)} \mathbf{F}_N \widetilde{\boldsymbol{\Gamma}}_\mu^{(N,N)}\right) \end{bmatrix}. \tag{4.30}$$

Each product inside (4.30) has a complexity of $O\left(I_n R^2 + R^3\right)$. As a consequence, $\mathbf{L}_\mu \boldsymbol{f}$ in (4.30) has a complexity of $O\left(TR^2 + NR^3\right) \approx O\left(TR^2\right)$ which is significantly lower than $O\left(2TR^3\right)$ for building up $\mathbf{L}_\mu$ and direct computation $\mathbf{L}_\mu \boldsymbol{f}$. Furthermore, this fast computation does not use any significant temporary extra-storage.

4.2.3.3 Fast dGN Algorithm

Replacing $\widetilde{\mathbf{G}}_\mu \mathbf{J}^T \left(\underline{y} - \underline{\hat{y}}\right)$, $\mathbf{L}_\mu^T \mathbf{J}^T \left(\underline{y} - \underline{\hat{y}}\right)$, and $\mathbf{L}_\mu \mathbf{B}_\mu \boldsymbol{w}$ in (4.26) by those in (4.27), (4.28) and (4.30) reformulates a compact update rule expressed for each factor $\mathbf{A}^{(n)}, n = 1, 2, \ldots, N$

$$\boxed{\mathbf{A}^{(n)} \leftarrow \mathbf{A}_{ALS}^{(n)} + \mathbf{A}^{(n)} \left(\mathbf{I}_R - \left(\mathbf{F}_n + \boldsymbol{\Gamma}^{(n,n)}\right) \widetilde{\boldsymbol{\Gamma}}_\mu^{(n,n)}\right),} \tag{4.31}$$

with $\mathbf{F}_n$ are frontal slices of $\mathcal{F}$ and $\boldsymbol{f} = \operatorname{vec}(\mathcal{F})$ is solution of the inverse problem

$$\boldsymbol{f} = \mathbf{K} \left(\mathbf{I}_{NR^2} + \boldsymbol{\Psi}\mathbf{K}\right)^{-1} \boldsymbol{w}, \tag{4.32}$$

or

$$\boldsymbol{f} = \left(\mathbf{K}^{-1} + \boldsymbol{\Psi}\right)^{-1} \boldsymbol{w}, \tag{4.33}$$

for invertible $\mathbf{K}$, where $\boldsymbol{w}$ is given in (4.28).

We note that both linear systems (4.32) and (4.33) have computational complexity of order $O(N^3 R^6)$ if we discard the sparsity and symmetric structures of matrices $\boldsymbol{\Phi}_1 = \mathbf{I}_{NR^2} + \boldsymbol{\Psi}\mathbf{K}$ and $\boldsymbol{\Phi}_2 = \mathbf{K}^{-1} + \boldsymbol{\Psi}$, while $(\mathbf{H} + \mu\,\mathbf{I})^{-1}$ has a computational complexity of order $O(R^3(\sum_n I_n)^3)$ or $O(N^3 R^3 I^3)$ for a tensor with $I_1 = I_2 = \cdots = I_N = I$. As a consequence, solving an inverse problem $\boldsymbol{f} = \mathbf{K}\,\boldsymbol{\Phi}_1^{-1}\,\boldsymbol{w}$ in (4.32) or $\boldsymbol{f} = \boldsymbol{\Phi}_2^{-1}\,\boldsymbol{w}$ in (4.33) is much less expensive than solving the inverse problem $(\mathbf{H} + \mu\mathbf{I})^{-1}\ \mathbf{J}^T\,\boldsymbol{e}$. Moreover, the update rule (4.31) does not employ the approximate Hessian $\mathbf{H}$ and the Jacobian $\mathbf{J}$. Hence, they are significantly faster than the standard dGN algorithms[143;200].

Pseudo code of the proposed algorithm based the update rule (4.31) is given in Algorithm 4.1. If components of $\mathbf{A}^{(n)}$ are mutually non-orthogonal, $\mathbf{K}$ is invertible, and its inverse can be efficiently computed as in Appendix A4.3. In this case, Step 8 is replaced by (4.104). Although normalization for factors $\mathbf{A}^{(n)}\,(n \neq N)$ is not explicitly shown in Algorithm 4.1, it is always needed after Step 23 in

Algorithm 4.1: Fast Algorithm for Low-Rank Approximation

Input: $\mathcal{Y}$: input data of size $I_1 \times I_2 \times \cdots \times I_N$,
R: number of basis components
Output: N factors $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$ such that the cost function (2.1) is minimized.

begin
  Random or SVD initialization for $\mathbf{A}^{(n)}, \forall n$
  repeat
    for $n = 1$ *to* N **do**
      for $m = n+1$ *to* N **do** // $\mathbf{K}$ in Eq. (4.6)
        $\mathbf{K}^{(n,m)} = \mathbf{K}^{(m,n)} = \mathbf{P}_{R,R}\,\mathrm{vec}\left(\mathbf{\Gamma}^{(n,m)}\right)$ // $\mathbf{\Gamma}^{(n,m)} = \circledast_{k\neq n,m}\, \mathbf{C}^{(k)}, \mathbf{C}^{(n)} = \mathbf{A}^{(n)T}\mathbf{A}^{(n)}$
      end
      $\widetilde{\mathbf{\Gamma}}_{\mu}^{(n,n)} = \left(\mathbf{\Gamma}^{(n,n)} + \mu\,\mathbf{I}_R\right)^{-1}$
      $\mathbf{A}_{ALS}^{(n)} \leftarrow \mathbf{Y}_{(n)} \left(\bigodot_{k\neq n} \mathbf{A}^{(k)}\right) \widetilde{\mathbf{\Gamma}}_{\mu}^{(n,n)}$ // damped ALS factor in (2.5)
      $\boldsymbol{w}_n = \mathrm{vec}\left(\mathbf{A}^{(n)\,T}\,\mathbf{A}_{ALS}^{(n)} - \mathbf{\Gamma}\,\widetilde{\mathbf{\Gamma}}_{\mu}^{(n,n)}\right)$ // Eq. (4.28), $\mathbf{W} = [\boldsymbol{w}_n]$
      $\mathbf{\Psi}^{(n,n)} = \widetilde{\mathbf{\Gamma}}_{\mu}^{(n,n)} \otimes \mathbf{C}^{(n)}$ // $\mathbf{\Psi} = \mathrm{blkdiag}\left(\mathbf{\Psi}^{(n,n)}\right)$ in Eq. (4.24)
    end
    $\boldsymbol{f} = \left(\mathbf{K}^{-1} + \mathbf{\Psi}\right)^{-1}\mathrm{vec}(\mathbf{W})$ // or $\boldsymbol{f} = \mathbf{K}\left(\mathbf{I} + \mathbf{\Psi}\mathbf{K}\right)^{-1}\mathrm{vec}(\mathbf{W})$ in Eq. (4.32)
    for $n = 1$ *to* N **do** // Update $\mathbf{A}^{(n)}$ using Eq. (4.31)
      $\mathbf{A}^{(n)} \leftarrow \mathbf{A}_{ALS}^{(n)} + \mathbf{A}^{(n)}\left(\mathbf{I}_R - \left(\mathbf{F}_n + \mathbf{\Gamma}^{(n,n)}\right)\widetilde{\mathbf{\Gamma}}_{\mu}^{(n,n)}\right)$ // $\mathrm{vec}(\mathcal{F}) = \boldsymbol{f}$
    end
    Update μ
  until *a stopping criterion is met*
end

Algorithm 4.1. A practical normalization is that the energy of the components is equally distributed in all modes. The method often enhances the convergence speed of the LM iteration[203].

4.2.3.4 Two Variants of the Fast dGN Algorithm

From (4.32), (4.33), we present two variants of the fast dGN algorithm which solve the corresponding inverse problem $\mathbf{\Phi}^{-1}\boldsymbol{w}$.

(a) **fLM$_\mathbf{a}$**. For problem (4.32), $\mathbf{\Phi} \triangleq \mathbf{\Phi}_1$ comprises N diagonal matrices $\mathbf{I}_{R^2}$, and $N\,(N-1)$ block matrices $\left(\mathbf{\Gamma}^{(n,n)^{-1}} \otimes \mathbf{C}^{(n)}\right)\mathbf{P}_{R,R}\,\mathbf{D}^{(n,m)}$, for $n \neq m$ given by

$$\mathbf{\Phi}_1 = \mathbf{I}_{NR^2} + \mathbf{\Psi}\,\mathbf{K}. \tag{4.34}$$

Note that $\mathbf{\Phi}_1$ is not symmetric, and its density is given by

$$d_{\Phi_1} = \frac{N\,(N-1)\,R^4 + N\,R^2}{N^2\,R^4} = \frac{(N-1)R^2 + 1}{NR^2}\,. \tag{4.35}$$

For 3-D tensor factorizations, the fast dGN algorithm in which Step 20 solves (4.33) simplifies into the LM-1 algorithm in [196].

(b) **fLM$_\mathbf{b}$**. For problem (4.32), we have a sparse and symmetric matrix $\mathbf{\Phi}_2$ of size $NR^2 \times NR^2$ derived from (4.6) and (4.24)

$$\mathbf{\Phi}_2 = \mathbf{K}^{-1} + \mathbf{\Psi}. \tag{4.36}$$

Appendix A4.3 presents an explicit form of $\mathbf{K}^{-1}$ which is a partitioned matrix of $(R^2 \times R^2)$ diagonal matrices (see Theorem 4.14). Hence, it has only $N^2\,R^2$ non-zero entries. The block diagonal matrix $\mathbf{\Psi}$ (4.24) is diagonally constructed from N $(R^2 \times R^2)$ sub-matrices. As a consequence, the density of the sparse matrix $\mathbf{\Phi}_2 \in \mathbb{R}^{NR^2 \times NR^2}$ is

$$d_{\Phi_2} = \frac{N^2\,R^2 + N\,R^4 - N\,R^2}{N^2\,R^4} = \frac{R^2 + N - 1}{N\,R^2}\,. \tag{4.37}$$

Because $\mathbf{\Phi}_1$ in (4.34) is not symmetric and less sparse than $\mathbf{\Phi}_2$, solving the linear system in (4.32) involving $\mathbf{\Phi}_1$ could be more time consuming than that in (4.33) with $\mathbf{\Phi}_2$. Inverse of $\mathbf{K}$ is not expensive and has explicit expression given in Theorem 4.14. However, when factor matrices have mutually orthogonal columns, $\mathbf{K}$ has collinear columns and rows, and is singular. We can switch between two variants by verifying whether $\mathbf{\Gamma}^{(n,m)}$ consists of any zero or close-to-zero entries. In Figure 4.2, we illustrate structures and properties of the two matrices $\mathbf{\Phi}_1$ and $\mathbf{\Phi}_2$ for a $3 \times 4 \times 5 \times 6 \times 7$ dimensional tensor composed by $R = 3$ rank-one tensors. We can reduce computational cost of inverse problems (4.32), (4.33) by employing sparsity and symmetric structures of $\mathbf{\Phi}_1$ and $\mathbf{\Phi}_2$.

4.2.3.5 Complexity of the Fast dGN Algorithm

For simplicity, complexity of Algorithm 4.1 is evaluated for an N-D tensor with $I_n = I, \forall n$.

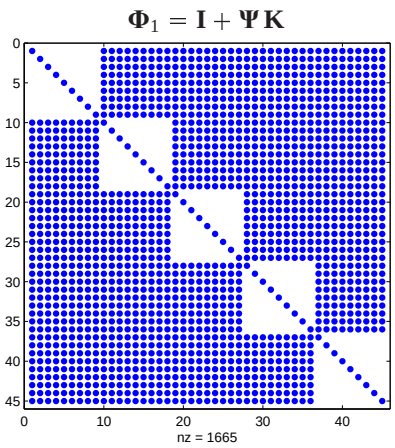
Step 8 computes $\mathbf{\Gamma}^{(n,n)}$ with complexity $O\left((N-2)R^2\right)$. Hence, building up $\mathbf{K}$ demands a complexity $O\left(\frac{1}{2}N(N-1)(N-2)R^2\right) = O\left(\frac{1}{2}N^3R^2\right)$.

Step 11 has a cost of $O\left(NR^3\right)$.

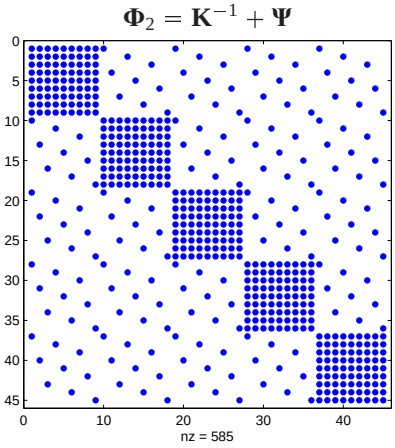
Step 13 computes the damped factors $\mathbf{A}^{(n)}$ at a cost of $O\left(NI^{N-1}R\right)$, and is one of the most expensive step in the fast dGN algorithm. We note that the large workload $\mathbf{Y}_{(n)} \bigodot\limits_{k\neq n} \mathbf{A}^{(k)}$ is used for evaluation of gradient, and exists in all CP algorithms such as ALS, OPT.

Step 17 builds up the block diagonal matrix $\mathbf{\Psi}$ with a complexity $O\left(NR^4\right)$.

Step 20 solves the inverse problem (4.32) or (4.33) with a cost of $O\left(N^3R^6\right)$. This step is much faster than inverse of the approximate Hessian $O\left(N^3R^3I^3\right)$ due to $R \ll I_n = I$ or $NR < \sum_n I_n$.



(a) $d_{\Phi_1} = \frac{(N-1)R^2+1}{NR^2}$.



(b) $d_{\Phi_2} = \frac{R^2+N-1}{NR^2}$.

Figure 4.2: Illustration of structure of $NR^2 \times NR^2$ sparse matrices $\mathbf{\Phi}_1$ and $\mathbf{\Phi}_2$ for a $3 \times 4 \times 5 \times 6 \times 7$ dimensional tensor composed by $R = 3$ rank-one tensors. The matrix $\mathbf{\Phi}_1$ is less sparse than the matrix $\mathbf{\Phi}_2$. Blue dots denote nonzero entries.

The total expense of the fast dGN algorithm per one iteration to update all the factors $\mathbf{A}^{(n)}$ is approximately $O\left(NI^{N-1}R + N^3R^6\right)$. For $N > 7$, the proposed algorithm has the same order of complexity as that of ALS. However, fast dGN requires less iterations than ALS. Hence, it converges faster than ALS.

4.2.4 Damping Parameter in LM Algorithm

The choice of damping parameter μ in the fast dGN algorithms (4.31) affects the direction and the step size $\Delta\mathfrak{a} = \mathbf{H}_\mu^{-1}\, \boldsymbol{g}$ in the update rule (4.2): $\mathfrak{a} \leftarrow \mathfrak{a} + \Delta\mathfrak{a}$[136]. In this chapter, the damping parameter μ is updated using the efficient strategy proposed by Nielsen[136]:

$$\mu \leftarrow \begin{cases} 2\max\left\{\frac{1}{3}, 1-(2\rho-1)^3\right\}, & \rho > 0, \\ 2\mu, & \textit{otherwise}, \end{cases}, \tag{4.38}$$

$$\rho = \frac{\|\boldsymbol{e}_{t-1}\|_2^2 - \|\boldsymbol{e}_t\|_2^2}{\Delta\mathfrak{a}^T\left(\boldsymbol{g} + \mu\,\Delta\mathfrak{a}\right)}, \tag{4.39}$$

$$\boldsymbol{g} = \mathbf{J}^T\left(\boldsymbol{y} - \hat{\boldsymbol{y}}\right) = \begin{bmatrix} \operatorname{vec}\left(\mathbf{Y}_{(1)}\left(\bigodot_{k\neq 1}\mathbf{A}^{(k)}\right) - \mathbf{A}^{(1)}\,\mathbf{\Gamma}^{(1,1)}\right) \\ \vdots \\ \operatorname{vec}\left(\mathbf{Y}_{(N)}\left(\bigodot_{k\neq N}\mathbf{A}^{(k)}\right) - \mathbf{A}^{(N)}\,\mathbf{\Gamma}^{(N,N)}\right) \end{bmatrix} \in \mathbb{R}^{R\sum_{n=1}^{N} I_n}. \tag{4.40}$$

where $\boldsymbol{e}_t = \text{vec}\left(\mathcal{Y} - \hat{\mathcal{Y}}_t\right)$, the gradient $\boldsymbol{g}$ can be straightforwardly derived as in (4.27) or in[200;202]. The factors $\mathbf{A}^{(n)}$ will be updated unless the new approximate error (2.1) is lower than the previous one: $\|\boldsymbol{e}_t\|_2 < \|\boldsymbol{e}_{t-1}\|_2$. The algorithm can stop when μ increases to a sufficiently large value (e.g. 10^{30}). In practice, factors $\mathbf{A}^{(n)}$ are often initialized using R leading left singular vectors of the tensor unfoldings along the corresponding mode[47;60;106], then run over ALS (2.2) after few iterations. According to the CP model (1.21), all the components $\boldsymbol{a}_r^{(n)} (n \neq N)$ except ones of the last factor are unit-length vectors. The initial value of the damping parameter μ is chosen as the maximum diagonal entry of $\mathbf{H}$ as

$$\begin{aligned} \mu_0 &= \tau \max\{\text{diag}(\mathbf{H})\} = \tau \max\left\{\text{diag}\left(\mathbf{\Gamma}^{(1,1)}\right) \cdots \text{diag}\left(\mathbf{\Gamma}^{(n,n)}\right) \cdots \text{diag}\left(\mathbf{\Gamma}^{(N,N)}\right)\right\} \\ &= \tau \max\left\{1, \text{diag}\left(\mathbf{C}^{(N)}\right)\right\}, \end{aligned} \tag{4.41}$$

where τ is typically in the range $[10^{-8}, 1]$.

4.3 Damped Gauss-Newton Algorithm for NTF

This section extends the dGN algorithm for CP to NTF. Paatero[143] proposed the PMF3 algorithm for NTF which minimizes the cost function (2.1) with a logarithmic penalty function to prevent factors $\mathbf{A}^{(n)}$ reaching zeros. In order to derive the fast dGN algorithm for NTF, we consider a similar cost function (2.1) with additional penalty terms P_l and P_s to enforce nonnegativity and sparsity constraints on factors $\mathbf{A}^{(n)}$, respectively,

$$D_+ = D(\mathcal{Y}, \{\mathbf{A}^{(n)}\}) + P_l(\{\mathbf{A}^{(n)}\}) + P_s(\{\mathbf{A}^{(n)}\}), \tag{4.42}$$

$$P_l = -\sum_{n=1}^{N} \alpha_n \sum_{i_n=1}^{I_n} \sum_{r=1}^{R} \log(a_{i_n r}^{(n)}), \quad P_s = \sum_{n=1}^{N} \beta_n \left\|\mathbf{A}^{(n)}\right\|_1, \tag{4.43}$$

where parameters $\alpha_n > 0$ and $\beta_n > 0$ for $n = 1, 2, \ldots, N$. The update rule derived from the cost function (4.42) simultaneously updates all the factors $\mathbf{A}^{(n)}$ based on the damped GN iteration (4.2)[143], and can be expressed in a common formula as

$$\boxed{\mathfrak{a} \leftarrow \mathfrak{a} - (\mathbf{H} + \mu \mathbf{I})^{-1}\, \boldsymbol{g},} \tag{4.44}$$

where $\mathfrak{a}^T = \left[\text{vec}\left(\mathbf{A}^{(n)}\right)^T\right]_{n=1}^{N}$, $\mu > 0$ is the damping parameter. The gradient $\boldsymbol{g}$ and the approximate Hessian $\mathbf{H}$ are given by

$$\boldsymbol{g} = \mathbf{J}^T(\hat{\mathcal{Y}} - \mathcal{Y}) + \frac{\partial P_l}{\partial \mathfrak{a}} + \frac{\partial P_s}{\partial \mathfrak{a}} = \mathbf{J}^T(\hat{\mathcal{Y}} - \mathcal{Y}) - \left[\left(\alpha_n \text{vec}\left(\mathbf{A}^{(n)}\right)^{\bullet[-1]} - \beta_n\right)^T\right]_{n=1}^{N \quad T}, \tag{4.45}$$

$$\mathbf{H} = \mathbf{J}^T \mathbf{J} + \frac{\partial^2 P_l}{\partial \mathfrak{a}^2} + \frac{\partial^2 P_s}{\partial \mathfrak{a}^2} = \mathbf{J}^T \mathbf{J} + \bigoplus_{n=1}^{N} \text{diag}\left\{\alpha_n \text{vec}\left(\mathbf{A}^{(n)}\right)^{\bullet[-2]}\right\}, \tag{4.46}$$

$$\mathbf{J} \;=\; \Big[\; \mathbf{P}_1\,(\{\mathbf{A}\}^{\odot -1}\otimes \mathbf{I}_1) \;\;\cdots\;\; \mathbf{P}_N\,(\{\mathbf{A}\}^{\odot -N}\otimes \mathbf{I}_N)\;\Big]\,, \tag{4.47}$$

where "$\bigoplus$" denotes a direct sum: $\mathbf{A}\oplus\mathbf{B} = \mathrm{diag}\{\mathbf{A},\mathbf{B}\}$, and $[\boldsymbol{x}]^{\bullet[p]}$ denotes element-wise powers. In the sequence, we present more efficient computation methods for the learning rule (4.44) for NTF.

4.3.1 Fast Computation of the Gradient g

From (4.40), by denoting $\mathbf{F}^{(n)} = \mathbf{Y}_{(n)}\{\mathbf{A}\}^{\odot -n} - \mathbf{A}^{(n)}\boldsymbol{\Gamma}^{(n,n)}$, the gradient (4.45) can be expressed as a concatenation vector without using the Jacobian $\mathbf{J}$

$$\boldsymbol{g}^T = \left[\mathrm{vec}\Big(-\mathbf{F}^{(n)} - \alpha_n\,\Big(\mathbf{A}^{(n)}\Big)^{\bullet[-1]} + \beta_n\Big)^T\right]_{n=1}^{N}. \tag{4.48}$$

4.3.2 Construction and Inverse of Approximate Hessian H

The approximate Hessian $\mathbf{H}_\mu = \mathbf{H}+\mu\mathbf{I}$ can be expressed as the concatenation of $R\,I_n \times R\,I_m$ dimensional block matrices $\mathbf{H}_\mu^{(n,m)}$ as $\mathbf{H}+\mu\mathbf{I} = \Big[\mathbf{H}_\mu^{(n,m)}\Big]$.

Theorem 4.4 (Expression of block matrix $\mathbf{H}_\mu^{(n,m)}$)**.** *A diagonal block matrix $\mathbf{H}_\mu^{(n,n)}$ can be expressed by*

$$\mathbf{H}_\mu^{(n,n)} = \boldsymbol{\Gamma}^{(n,n)}\otimes\mathbf{I}_{I_n} + \mathrm{diag}\left\{\alpha_n\,\mathrm{vec}\Big(\mathbf{A}^{(n)}\Big)^{\bullet[-2]} + \mu\right\}, \tag{4.49}$$

and a block matrix $\mathbf{H}_\mu^{(n,m)}$ for $n\neq m$ is given by

$$\mathbf{H}_\mu^{(n,m)} = (\mathbf{I}_R\otimes\mathbf{A}^{(n)})\,\mathbf{P}_{R,R}\,\mathrm{diag}(\mathrm{vec}(\boldsymbol{\Gamma}^{(n,m)}))(\mathbf{I}_R\otimes\mathbf{A}^{(m)T}), \tag{4.50}$$

where $\boldsymbol{\Gamma}^{(n,m)} = \circledast_{k\neq m,n}\,\mathbf{A}^{(k)T}\,\mathbf{A}^{(k)}$, and $\mathbf{P}_{R,R}$ is a permutation matrix of $\mathbf{X}$: $\mathrm{vec}(\mathbf{X}) = \mathbf{P}_{R,R}\,\mathrm{vec}\big(\mathbf{X}^T\big)$.

Proof. Proof is directly derived from (4.47) and (4.46) and Theorem 4.3. □

Theorem 4.5 (Low rank Adjustment for the approximate Hessian $\mathbf{H}_\mu$)**.** *The approximate Hessian $\mathbf{H}_\mu$ can be decomposed under the form as*

$$\mathbf{H}_\mu = \mathbf{G} + \mathbf{Z}\,\mathbf{K}\,\mathbf{Z}^T. \tag{4.51}$$

where $\mathbf{K}$ is defined in (4.6), and matrices $\mathbf{G}\in\mathbb{R}^{TR\times TR}$ and $\mathbf{Z}\in\mathbb{R}^{TR\times R^2}$ are given by

$$\mathbf{G} = \mathbf{P}\left(\bigoplus_{n=1}^{N}\bigoplus_{i_n=1}^{I_n}\left(\boldsymbol{\Gamma}^{(n,n)} + \mathrm{diag}\{\alpha_n\boldsymbol{a}_{i_n:}^{(n)\bullet[-2]} + \mu\}\right)\right)\mathbf{P}^T, \tag{4.52}$$

$$\mathbf{Z} = \bigoplus_{n=1}^{N}\Big(\mathbf{I}_R\otimes\mathbf{A}^{(n)}\Big)\,, \tag{4.53}$$

in which $\mathbf{P} = \bigoplus_{n=1}^{N}\mathbf{P}_{R,I_n}$, $\mathbf{P}_{R,I_n}$ is a permutation matrix, $\mathrm{vec}(\mathbf{X}) = \mathbf{P}_{R,I_n}\,\mathrm{vec}\big(\mathbf{X}^T\big)$.

Proof. Proof is derived from Theorem 4.4 and Theorem 4.1. □

Theorem 4.5 allows to inverse the large approximate Hessian via the binomial inverse theorem[94]

$$\mathbf{H}_{\mu}^{-1} = \mathbf{G}^{-1} - \mathbf{G}^{-1}\,\mathbf{Z}\left(\mathbf{K}^{-1} + \mathbf{Z}^T\,\mathbf{G}^{-1}\,\mathbf{Z}\right)^{-1}\,\mathbf{Z}^T\,\mathbf{G}^{-1}. \tag{4.54}$$

We note that $\mathbf{G}^{-1}$ can be efficiently computed as

$$\mathbf{G}^{-1} \;=\; \mathbf{P}\left(\bigoplus_{n=1}^{N}\bigoplus_{i_n=1}^{I_n}\boldsymbol{\Theta}_{i_n}^{(n)}\right)\mathbf{P}^T, \tag{4.55}$$

where $\boldsymbol{\Theta}_{i_n}^{(n)} = \left(\boldsymbol{\Gamma}^{(n,n)} + \text{diag}\left\{\alpha_n \boldsymbol{a}_{i_n:}^{(n)\bullet[-2]} + \mu\right\}\right)^{-1}$. The inverse of the partitioned matrix $\mathbf{K}$ has an explicit expression given in Theorem 4.14[167].

Products of matrices in (4.54) can be computed as

$$\mathbf{L} = \mathbf{G}^{-1}\,\mathbf{Z}^T \;=\; \mathbf{P}\left(\bigoplus_{n=1}^{N}\boldsymbol{\Upsilon}^{(n)}\right)\left(\mathbf{I}_N \otimes \mathbf{P}_{R,R}\right), \tag{4.56}$$

$$\mathbf{Z}^T\,\mathbf{G}^{-1}\,\mathbf{Z} \;=\; \left(\mathbf{I}_N \otimes \mathbf{P}_{R,R}\right)\bigoplus_{n=1}^{N}\mathbf{W}^{(n)}\left(\mathbf{I}_N \otimes \mathbf{P}_{R,R}\right), \tag{4.57}$$

where $\boldsymbol{\Upsilon}^{(n)T} = \left[\boldsymbol{a}_{i_n:}^{(n)T} \otimes \boldsymbol{\Theta}_{i_n}^{(n)T}\right]_{i_n=1}^{I_n}$, and $\mathbf{W}^{(n)} = \left[\mathbf{W}_{r,s}^{(n)}\right]$ is a partitioned matrix of sub-matrices $\mathbf{W}_{r,s}^{(n)} = \sum_{i_n=1}^{I_n}\boldsymbol{\Theta}_{i_n}^{(n)}\,a_{i_n,r}^{(n)}\,a_{i_n,s}^{(n)}$, for $r = 1,2,\ldots,R$ and $s = 1,2,\ldots,R$.

From (4.48), (4.54), (4.55), (4.56), (4.57) and Theorem 4.14, the update rule (4.44) is formulated in a more efficient form as follows

$$\boxed{\mathfrak{a} \leftarrow \mathfrak{a} - \mathbf{G}^{-1}\,\boldsymbol{g} + \mathbf{L}\,\boldsymbol{\Phi}^{-1}\left(\mathbf{L}^T\,\boldsymbol{g}\right),} \tag{4.58}$$

where $\boldsymbol{\Phi} = \mathbf{K}^{-1} + \mathbf{Z}^T\,\mathbf{G}^{-1}\,\mathbf{Z} \in \mathbb{R}^{NR^2\times NR^2}$. We note that for low rank approximation $R \ll I_n, \forall n$, the matrix $\boldsymbol{\Phi}$ is much smaller than the approximate Hessian $\mathbf{H} \in \mathbb{R}^{RT\times RT}$. Inverses of the matrix $\boldsymbol{\Phi}$ requires computational complexity of order $O(N^3R^6)$, while $(\mathbf{H}+\mu\,\mathbf{I})^{-1}$ has a computational complexity of order $O(R^3T^3)$ or $O(N^3R^3I^3)$ for a symmetric tensor $I_1 = \cdots = I_N = I$. As a consequence, solving an inverse problem $\boldsymbol{w} = \boldsymbol{\Phi}^{-1}\boldsymbol{\psi}$, $\boldsymbol{\psi} = \mathbf{L}^T\,\boldsymbol{g}$ in the update rule (4.58) is much less expensive than solving the problem $(\mathbf{H}+\mu\mathbf{I})^{-1}\,\boldsymbol{g}$ in the update rule (4.44). Moreover, because the learning rule (4.58) does not need to construct the approximate Hessian $\mathbf{H}$ and the Jacobian $\mathbf{J}$, this update rule is significantly faster than the rule (4.44).

4.3.3 Selection of Barrier and Sparse Parameters

Paatero[143] suggested an heuristic approach to control the barrier parameter $\alpha = \alpha_n, \forall n$. The parameter α should be initialized by a large enough value, then slowly descends down to near-zero after each 10

iterations. This strategy is efficient in almost all cases. However, we cannot control the convergence speed. An alternative approach is to adaptively select regularization parameters based on the Karush-Kuhn-Tucker (KKT) condition. In this direction, Rojas and Steihaug updated barrier parameter at each iteration[174]. Ding *et al.* derived an optimal formula for regularization parameters in orthogonal NMF[65]. The sparsity parameters β_n are often fixed to a small enough value. By employing this method, regularization parameters α_n and β_n ($\forall n$) are selected based on the KKT slackness condition

$$\mathfrak{a} \geqslant \mathbf{0}, \quad \mathfrak{a} \circledast \boldsymbol{g} = \mathbf{0}, \quad \boldsymbol{g} \geqslant \mathbf{0}. \tag{4.59}$$

From conditions (4.59), and the gradient (4.48), we obtain

$$\mathbf{A}^{(n)} \circledast \left(\mathbf{F}^{(n)} + \alpha_n \left(\mathbf{A}^{(n)} \right)^{\bullet[-1]} - \beta_n \right) = \mathbf{0}, \quad \forall n, \tag{4.60}$$

$$\mathbf{F}^{(n)} + \alpha_n \left(\mathbf{A}^{(n)} \right)^{\cdot[-1]} - \beta_n \leqslant \mathbf{0}, \quad \forall n. \tag{4.61}$$

This leads to solve a constrained LS problem

$$\min_{\boldsymbol{x}} \|\mathbf{B}^{(n)} \boldsymbol{x} - \boldsymbol{u}^{(n)}\|_2^2 \;\text{ such that }\; \begin{cases} \mathbf{B}^{(n)} \boldsymbol{x} \leqslant \boldsymbol{u}^{(n)}, \\ \boldsymbol{x} = [\alpha_n \; \beta_n]^T \geqslant \mathbf{0}, \end{cases} \quad \forall n, \tag{4.62}$$

where $\mathbf{B}^{(n)} = \left[\mathbf{1} \;\; \text{vec}\left(-\mathbf{A}^{(n)}\right)\right]$, and $\boldsymbol{u}^{(n)} = \text{vec}\left(-\mathbf{A}^{(n)} \circledast \mathbf{F}^{(n)}\right)$.

For NTF without the penalty term P_s, conditions (4.59) lead to the result $\mathbf{A}^{(n)} \circledast \mathbf{F}^{(n)} + \alpha_n \leqslant \mathbf{0}, \forall n$. The parameters α_n can be chosen as

$$\alpha_n = \max\left(0, \min\left(\text{vec}\left(-\mathbf{A}^{(n)} \circledast \mathbf{F}^{(n)}\right)\right)\right), \quad \forall n. \tag{4.63}$$

4.4 Complex-valued Tensor Factorization

This section aims to extend the dGN algorithms to complex-valued tensors. Although a real-valued tensor is considered as a complex-valued tensor with zero imaginary part, for simplicity algorithms for real- and complex-valued tensors are introduced into separate sections. For the complex case, CP model is to find complex-valued factors $\mathbf{A}^{(n)} \in \mathbb{C}^{I_n \times R}$.

The damped Gauss-Newton-like update rule (4.2) is rewritten to update complex-valued factors[76;189]

$$\boxed{\mathfrak{a} \leftarrow \mathfrak{a} + \left(\mathbf{J}^H \mathbf{J} + \mu \mathbf{I}\right)^{-1} \mathbf{J}^H \left(\underline{y} - \underline{\hat{y}}\right),} \tag{4.64}$$

where symbol "H" denotes the Hermitian transpose, and the Jacobian $\mathbf{J}$ is given in (4.9). The approximate Hessian $\mathbf{H} = \mathbf{J}^H \mathbf{J}$ slightly changes from that for the real-valued tensors in (4.10). A fast and efficient computation method for the complex-valued Hessian $\mathbf{H}$ will be presented so that the final

update rule does not employ both of the Jacobian and the approximate Hessian. The explicit expression of the approximate Hessian $\mathbf{H}$ is deduced from the following theorems which can be derived in a similar manner presented in the previous sections.

Theorem 4.6 (Block matrices $\mathbf{H}_{r,s}^{(n,m)}$)**.** *Block matrices $\mathbf{H}_{r,s}^{(n,m)}$ of the approximate Hessian $\mathbf{H}$ are diagonal or rank-one matrices given by*

$$\mathbf{H}_{r,s}^{(n,m)} = \delta_{n,m}\, \gamma_{r,s}^{(n,n)}\, \mathbf{I}_{I_n} + (1-\delta_{n,m})\, \gamma_{r,s}^{(n,m)}\, \boldsymbol{a}_r^{(n)}\, \boldsymbol{a}_s^{(m)\,H}, \tag{4.65}$$

where $\gamma_{r,s}^{(n,n)}$ are the (r,s) entries of the Hermitian matrices $\boldsymbol{\Gamma}^{(n,m)} = \circledast_{k\neq n,m} \mathbf{A}^{(k)\,H}\mathbf{A}^{(k)}$.

Theorem 4.7 (Block matrices $\mathbf{H}^{(n,m)}$)**.** *Block matrices $\mathbf{H}^{(n,m)}$ of the approximate Hessian $\mathbf{H}$ are expressed in an explicit form as*

$$\mathbf{H}^{(n,m)} = \delta_{n,m}\boldsymbol{\Gamma}^{(n,n)} \otimes \mathbf{I}_{I_n} + \left(\mathbf{I}_R \otimes \mathbf{A}^{(n)}\right) \mathbf{K}^{(n,m)} \left(\mathbf{I}_R \otimes \mathbf{A}^{(m)\,H}\right), \tag{4.66}$$

where $\mathbf{K}$ is defined as in (4.6).

Theorem 4.8 (Low-Rank Adjustment)**.** *The approximate Hessian $\mathbf{H} = \mathbf{J}^H\mathbf{J}$ can be expressed as a low-rank adjustment given by*

$$\mathbf{H} = \mathbf{G} + \mathbf{Z}\,\mathbf{K}\,\mathbf{Z}^H, \tag{4.67}$$

where sparse matrices $\mathbf{G}$, $\mathbf{Z}$ and $\mathbf{K}$ are defined as in (4.4), (4.5) and (4.6).

The damped Gauss-Newton algorithms for complex-valued tensor factorization are stated in following theorems:

Theorem 4.9 (damped GN algorithm for complex-valued tensor factorizations)**.** *The factors $\mathbf{A}^{(n)}$ are updated using the rule given by*

$$\mathfrak{a} \;\leftarrow\; \mathfrak{a} + \left(\mathbf{H} + \mu\,\mathbf{I}\right)^{-1} \boldsymbol{g}, \tag{4.68}$$

where the approximate Hessian $\mathbf{H}$ is defined in Theorems 4.6 or 4.7, and the gradient $\boldsymbol{g} \in \mathbb{C}^{RT}$ is computed as

$$\boldsymbol{g} = \left[\operatorname{vec}\left(\mathbf{Y}_{(n)}\left(\bigodot_{k\neq n}\mathbf{A}^{(k)*}\right) - \mathbf{A}^{(n)}\,\boldsymbol{\Gamma}^{(n,n)\,T}\right)^T\right]_{n=1}^{N\quad T}, \tag{4.69}$$

where symbol '' denotes the complex conjugate.*

Theorem 4.10 (Learning rule for low rank approximation)**.** *For $NR < T$, the factors $\mathbf{A}^{(n)}$ are updated using the fast update rule given by*

$$\mathfrak{a} \;\leftarrow\; \mathfrak{a} + \mathfrak{a}_{ALS} - \hat{\mathfrak{a}}_{ALS} - \mathbf{L}_\mu\,\mathbf{B}_\mu\,\boldsymbol{w}, \tag{4.70}$$

where $\mathbf{B}_\mu = \left(\mathbf{K}^{-1} + \mathbf{Z}^H\,\mathbf{G}_\mu^{-1}\,\mathbf{Z}\right)^{-1}$ if $\mathbf{K}$ is invertible, or $\mathbf{B}_\mu = \mathbf{K}\,\left(\mathbf{I} + \mathbf{Z}^H\mathbf{G}_\mu^{-1}\,\mathbf{Z}\,\mathbf{K}\right)^{-1}$, $\mathbf{L}_\mu$ is defined in (4.23), and an Levenberg-Marquardt regularization parameter μ, and vector $\boldsymbol{w}$ is computed from damped ALS factors $\mathbf{A}_{ALS}^{(n)}$ and $\hat{\mathbf{A}}_{ALS}^{(n)}$

$$\boldsymbol{w} = \text{vec}\left(\left[\mathbf{A}^{(n)H} \left(\mathbf{A}^{(n)}_{ALS} - \hat{\mathbf{A}}^{(n)}_{ALS} \right) \right]_{n=1}^{N} \right), \tag{4.71}$$

$$\mathbf{A}^{(n)}_{ALS} = \mathbf{Y}_{(n)} \left(\bigodot_{k \neq n} \mathbf{A}^{(k)*} \right) \left(\mathbf{\Gamma}^{(n,n)T} + \mu \mathbf{I} \right)^{-1}, \tag{4.72}$$

$$\hat{\mathbf{A}}^{(n)}_{ALS} = \mathbf{A}^{(n)} \mathbf{\Gamma}^{(n,n)T} \left(\mathbf{\Gamma}^{(n,n)T} + \mu \mathbf{I} \right)^{-1}. \tag{4.73}$$

We note that ALS for complex-valued CP with damping parameter μ is given by

$$\mathbf{A}^{(n)}_{ALS} \leftarrow \mathbf{Y}_{(n)} \left(\bigodot_{k \neq n} \mathbf{A}^{(k)*} \right) \left(\mathbf{\Gamma}^{(n,n)T} + \mu \mathbf{I} \right)^{-1}. \tag{4.74}$$

The update rule (4.70) can be reformulated for each factor $\mathbf{A}^{(n)}$ as

$$\boxed{\mathbf{A}^{(n)} \leftarrow \mathbf{A}^{(n)}_{ALS} + \mathbf{A}^{(n)} \left(\mathbf{I}_R - \left(\mathbf{F}_n + \mathbf{\Gamma}^{(n,n)T} \right) \left(\mathbf{\Gamma}^{(n,n)T} + \mu \mathbf{I}_R \right)^{-1} \right),} \tag{4.75}$$

where $\mathbf{F}_n$ is defined in (4.29). Kronecker product has been eliminated in building up $\mathbf{L}_\mu$.

4.5 Damped Gauss-Newton Algorithm for Tucker Decomposition

Most algorithms for Tucker decomposition minimize the cost function

$$D(\mathcal{Y} \| \mathcal{G} \times \{\mathbf{A}^{(n)}\}) = \| \mathcal{Y} - \mathcal{G} \times \{\mathbf{A}\} \|_F^2, \tag{4.76}$$

via alternating optimization which often accompanies update rules with low computational cost, but face problems of slow convergence. For NTD, the multiplicative algorithms[47;103;131] are based on minimization of the squared Euclidean distance (Frobenius norm) and the Kullback-Leibler divergence.

All-at-once algorithms which simultaneously update all the factors cope with such problems. For Tucker decomposition and NTD, due to high computational cost of Kronecker products and consumption of extremely large temporary extra-storage, the dGN method has not yet been considered.

In this section, an all-at-once algorithm with low complexity will be derived for Tucker decomposition with/without nonnegative constraints based on the damped Gauss-Newton (dGN) iteration. A logarithmic barrier penalty term has been imposed on the cost function (4.76) to enforce nonnegativity constraints. The proposed algorithm is verified to overwhelmingly outperform "state-of-the-art" NTD algorithms for difficult benchmarks.

The proposed algorithm minimizes the cost function (4.76) with a logarithmic penalty function to prevent factors $\mathbf{A}^{(n)}$ and the core tensor $\mathcal{G}$ reaching zeros.

$$D_+ = D(\mathcal{Y} \| \mathcal{G} \times \{\mathbf{A}^{(n)}\}) - \alpha P_l(\{\mathbf{A}^{(n)}\}, \mathcal{G}), \tag{4.77}$$

$$P_l = \sum_{n=1}^{N} \sum_{i_n=1}^{I_n} \sum_{r=1}^{R_n} \log(a^{(n)}_{i_n r}) + \sum_{\boldsymbol{r}=[r_1,r_2,\ldots,r_N]} \log(g_{\boldsymbol{r}}), \tag{4.78}$$

where $\alpha > 0$, $a^{(n)}_{i_n r}$ and $g_{\boldsymbol{r}} = g_{r_1 r_2 \cdots r_N}$ are elements of factors $\mathbf{A}^{(n)}$ and core tensor $\boldsymbol{\mathcal{G}}$, respectively. The update rule derived from (4.77) simultaneously updates all the factors $\mathbf{A}^{(n)}$ and the core tensor $\boldsymbol{\mathcal{G}}$ based on the damped GN iteration[143] given by

$$\mathfrak{a} \leftarrow \mathfrak{a} - \left(\mathbf{H} + \mu \mathbf{I}\right)^{-1} \boldsymbol{g}, \tag{4.79}$$

where $\mathfrak{a}^T = \left[\text{vec}\left(\mathbf{A}^{(1)}\right)^T, \cdots, \text{vec}\left(\mathbf{A}^{(N)}\right)^T, \text{vec}(\boldsymbol{\mathcal{G}})^T\right]$, $\mu > 0$ is the damping parameter, and $\mathbf{I}$ is the identity matrix. The gradient $\boldsymbol{g}$ and the approximate Hessian $\mathbf{H}$ are given by

$$\begin{aligned}
\boldsymbol{g} &= \mathbf{J}^T(\hat{\mathcal{y}} - \mathcal{y}) - \alpha \frac{\partial P_l}{\partial \mathfrak{a}} = \mathbf{J}^T(\hat{\mathcal{y}} - \mathcal{y}) - \alpha\, \mathfrak{a}^{\bullet[-1]}, && (4.80)\\
\mathbf{H} &= \mathbf{J}^T\mathbf{J} - \alpha \frac{\partial^2 P_l}{\partial \mathfrak{a}^2} = \mathbf{J}^T\mathbf{J} + \alpha\ \text{diag}\{\mathfrak{a}^{\bullet[-2]}\}, && (4.81)\\
\mathbf{J} &= \begin{bmatrix} \mathbf{J}_1 & \mathbf{J}_2 & \cdots & \mathbf{J}_N & \mathbf{J}_{N+1} \end{bmatrix}, && (4.82)\\
\mathbf{J}_n &= \begin{cases} \mathbf{P}_n^T \left(\{\mathbf{A}\}^{\otimes -n} \mathbf{G}_{(n)}{}^T \otimes \mathbf{I}_{I_n}\right), & n = 1, 2, \ldots, N, \\ \{\mathbf{A}\}^{\otimes}, & n = N+1, \end{cases} && (4.83)
\end{aligned}$$

The Jacobian matrix $\mathbf{J} \in \mathbb{R}^{(\prod I_n) \times \left(\sum_{n=1}^{N} R_n I_n\right)}$ can be directly utilized in the learning rule (4.79). However, this demands high computational cost for construction of the approximate Hessian $(\mathbf{H} + \mu\mathbf{I})$ due to high computational cost of Kronecker products. In the sequence, we present more efficient computation methods for the learning rule (4.79).

4.5.1 Fast Computation of the Gradient

From (4.83), for $n = 1, 2, \ldots, N$, we have the following result

$$\begin{aligned}
\mathbf{J}_n^T\left(\mathcal{y} - \hat{\mathcal{y}}\right) &= \left(\{\mathbf{A}\}^{\otimes -n\,T} \mathbf{G}_{(n)} \otimes \mathbf{I}_n\right) \mathbf{P}_n^T \text{vec}\left(\boldsymbol{\mathcal{Y}} - \hat{\boldsymbol{\mathcal{Y}}}\right) \\
&= \text{vec}\left(\left(\mathbf{Y}_{(n)} - \hat{\mathbf{Y}}_{(n)}\right) \{\mathbf{A}\}^{\otimes -n\,T} \mathbf{G}_{(n)}\right) \\
&= \text{vec}\left(\langle \boldsymbol{\mathcal{Y}} \times_{-n} \{\mathbf{A}^T\} - \boldsymbol{\mathcal{G}} \times_{-n} \{\mathbf{A}^T\mathbf{A}\}, \boldsymbol{\mathcal{G}} \rangle_{-n}\right), \qquad (4.84)
\end{aligned}$$

where $\langle \boldsymbol{\mathcal{Y}}, \hat{\boldsymbol{\mathcal{Y}}} \rangle_{-n}$ denotes contracted product between $\boldsymbol{\mathcal{Y}}, \hat{\boldsymbol{\mathcal{Y}}}$ along all their modes except mode-n. Tensor products $\boldsymbol{\mathcal{Y}} \times_{-n} \{\mathbf{A}^T\}$ and $\boldsymbol{\mathcal{G}} \times_{-n} \{\mathbf{A}^T\mathbf{A}\}$ are defined in (1.4) and can be efficiently calculated over a hierarchical stage of tensor-matrix multiplications. This avoids Kronecker products which are often computationally demanding, and consume significant temporary extra-storage. For example, Kronecker products $\{\mathbf{A}\}^{\otimes -n}$ produce large-scale matrices of size $\prod_{k \neq n} I_k \times \prod_{k \neq n} R_k$. Therefore, tensor products $\boldsymbol{\mathcal{Y}} \times_{-n} \{\mathbf{A}^T\}$ and $\boldsymbol{\mathcal{G}} \times_{-n} \{\mathbf{A}^T\mathbf{A}\}$ are much less computationally expensive than products $\mathbf{Y}_{(n)} \{\mathbf{A}\}^{\otimes -n\,T}$, $\hat{\mathbf{Y}}_{(n)} \{\mathbf{A}\}^{\otimes -n\,T}$. Moreover, we don't need to build up the tensor $\hat{\boldsymbol{\mathcal{Y}}}$ in (4.84).

Similarly, we have

$$\mathbf{J}_{N+1}^T\left(\mathcal{y} - \hat{\mathcal{y}}\right) = \text{vec}\left(\boldsymbol{\mathcal{Y}} \times \{\mathbf{A}^T\} - \boldsymbol{\mathcal{G}} \times \{\mathbf{A}^T\mathbf{A}\}\right). \tag{4.85}$$

From (4.84) and (4.85), we established a fast computation for the gradient $\boldsymbol{g}$ without computing the Jacobian $\mathbf{J}$.

4.5.2 Construction of Approximate Hessian H

This section presents a low computational cost method to build up the approximate Hessian given in (4.81). For simplicity, we construct the approximate Hessian $\mathbf{H} = \mathbf{J}^T\mathbf{J}$ without the regularization term which can be expressed as concatenation of $(N+1)^2$ block matrices $\mathbf{H}^{(n,m)} = \mathbf{H}^{(m,n)\,T}, m, n = 1, 2, \ldots, N+1$ as

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}^{(1,1)} & \cdots & \mathbf{H}^{(1,N+1)} \\ \vdots & \ddots & \vdots \\ \mathbf{H}^{(N+1,1)} & \cdots & \mathbf{H}^{(N+1,N+1)} \end{bmatrix}. \tag{4.86}$$

4.5.2.1 Off-Diagonal Block Matrices $\mathbf{H}^{(n,m)}(n \neq m)$

Without loss of generality, we consider submatrices $\mathbf{H}^{(n,m)}$, for $1 \leqslant n < m \leqslant N$. From (4.83), for $n = 1, 2, \ldots, N$, we have

$$\begin{aligned} \mathbf{J}_n^T &= \mathbf{P}_{R_n,I_n}\left(\mathbf{I}_{I_n} \otimes \mathbf{G}_{(n)}\left\{\mathbf{A}\right\}^{\otimes_{-n}T}\right)\mathbf{P}_{I_n,K_n}\mathbf{P}_n \\ &= \mathbf{P}_{R_n,I_n}\left(\mathbf{I}_{I_n} \otimes \mathbf{G}_{(n)}\right)\left(\mathbf{I}_{I_n} \otimes \left\{\mathbf{A}\right\}^{\otimes_{-n}T}\right)\mathbf{P}_{I_n,K_n}\mathbf{P}_n \\ &= \mathbf{P}_{R_n,I_n}\left(\mathbf{I}_{I_n} \otimes \mathbf{G}_{(n)}\right)\left(\mathbf{P}_{I_n,R_{n+1:N}} \otimes \mathbf{I}_{R_{1:n-1}}\right)\left(\bigotimes_{k=n+1}^{N} \mathbf{A}^{(k)\,T} \otimes \mathbf{I}_{I_n} \otimes \bigotimes_{k=1}^{n-1} \mathbf{A}^{(k)T}\right), \end{aligned} \tag{4.87}$$

where $K_n = \prod_{k\neq n} I_k$, $R_{n+1:N} = \prod_{k=n+1}^{N} R_k$, $R_{1:n-1} = \prod_{k=1}^{n-1} R_k$, $\mathbf{I}_J$ is an $J \times J$ identity matrix. Permutation matrices $\mathbf{P}_{R_n,I_n}$, $\mathbf{P}_{I_n,K_n}$ and $\mathbf{P}_{I_n,R_{n+1:N}}$ are defined in Appendix A4.1: $\text{vec}(\mathbf{X}_{I\times J}) = \mathbf{P}_{I,J}\ \text{vec}\left(\mathbf{X}_{I\times J}^T\right)$. Let $\mathbf{Q}^{(n)}$ denote the matrix product in (4.87)

$$\mathbf{Q}^{(n)} = \left(\mathbf{I}_{I_n} \otimes \mathbf{G}_{(n)}\right)\left(\mathbf{P}_{I_n,R_{n+1:N}} \otimes \mathbf{I}_{R_{1:n-1}}\right) = \begin{bmatrix} \tilde{\mathbf{G}}_{(n)}^{(n_1)} \\ \vdots \\ \tilde{\mathbf{G}}_{(n)}^{(n_{I_n})} \end{bmatrix}, \tag{4.88}$$

where $\tilde{\mathbf{G}}_{(n)}^{(n_{i_n})}$ $(i_n = 1, 2, \ldots, I_n, n = 1, 2, \ldots, N)$ are the mode-n matricized versions of $(N+1)$-dimensional tensors $\tilde{\mathcal{G}}^{(n_{i_n})}$ of size $R_1 \times R_2 \times \cdots \times R_n \times I_n \times R_{n+1} \times \cdots \times R_N$ in which the tensor $\mathcal{G}$ is a subtensor obtained from the tensor $\tilde{\mathcal{G}}^{(n_{i_n})}$ by fixing the $(n+1)$-th index to i_n, and other entries are set to zeros, that is

$$\tilde{\mathbf{G}}_{(n+1)}^{(n_{i_n})} = \begin{bmatrix} \mathbf{0}_{(i_n-1)\times\prod_{k=1}^{N} R_k} \\ \text{vec}(\mathcal{G})^T \\ \mathbf{0}_{(I_n-i_n)\times\prod_{k=1}^{N} R_k} \end{bmatrix}. \tag{4.89}$$

For each pair of indices (n, m), we define a set of $(N+1)$ matrices $\mathbf{B}^{(k)}$ given by

$$\mathbf{B}^{(k)} = \begin{cases} \mathbf{A}^{(k)\,T}\mathbf{A}^{(k)}, & 1 \leqslant k \leqslant n, \\ \mathbf{A}^{(n)\,T}, & k = n+1, \\ \mathbf{A}^{(k-1)\,T}\mathbf{A}^{(k-1)}, & n+2 \leqslant k \leqslant m, \\ \mathbf{A}^{(m)}, & k = m+1, \\ \mathbf{A}^{(k-1)\,T}\mathbf{A}^{(k-1)}, & m+2 \leqslant k \leqslant N+1. \end{cases}$$

Note that for simplicity, indices n, m are omitted in matrices $\mathbf{B}^{(k)}$. We have the following expression for $1 \leqslant n < m \leqslant N$

$$\begin{aligned} \tilde{\mathbf{G}}_{(n)}^{(n_{i_n})} \left(\{\mathbf{B}\}^{\otimes -n}\right)^T \tilde{\mathbf{G}}_{(m)}^{(m_{i_m})\,T} &= \left(\tilde{\mathcal{G}} \times_{-\binom{n}{n+1}{m+1}} \{\mathbf{B}\} \times_{m+1} \mathbf{A}^{(m)} \times_{n+1} \mathbf{A}^{(n)\,T}\right)_{(n)} \tilde{\mathbf{G}}_{(m)}^{(m_{i_m})\,T} \\ &= \left(\mathcal{W}^{(n,m)} \bar{\times}_{m+1} \boldsymbol{a}_{i_m:}^{(m)T} \times_{n+1} \boldsymbol{a}_{i_n:}^{(n)\,T}\right)_{(n)} \mathbf{G}_{(m)}^T, \end{aligned} \tag{4.90}$$

where $\mathcal{W}^{(n,m)} = \hat{\mathcal{G}} \times_{-\binom{n}{n+1}{m+1}} \{\mathbf{B}\}$, and $\hat{\mathcal{G}}$ is an augmented version of the tensor $\mathcal{G}$ of size $R_1 \times \cdots \times R_n \times 1 \times R_{n+1} \times \cdots \times R_N$. We note that $\text{vec}(\hat{\mathcal{G}}) = \text{vec}(\mathcal{G})$.

From (4.83), (4.87), submatrices $\mathbf{H}^{(n,m)} = \mathbf{J}_n^T \mathbf{J}_m, n < m$ can be expressed as follows

$$\begin{aligned} \mathbf{P}_{R_n,I_n}^T \mathbf{H}^{(n,m)} \mathbf{P}_{R_m,I_m} &= \tilde{\mathbf{G}}_{(n)}^{(n_{i_n})} \left(\left(\bigotimes_{k=m+1}^{N} \mathbf{A}^{(k)T}\mathbf{A}^{(k)} \right) \otimes \mathbf{A}^{(m)T} \otimes \right. \\ &\quad \left. \left(\bigotimes_{k=n+1}^{m-1} \mathbf{A}^{(k)T}\mathbf{A}^{(k)} \right) \otimes \mathbf{A}^{(n)} \otimes \left(\bigotimes_{k=1}^{n-1} \mathbf{A}^{(k)T}\mathbf{A}^{(k)} \right) \right) \tilde{\mathbf{G}}_{(m)}^{(m_{i_m})\,T} \\ &= \mathbf{Q}^{(n)} \left\{ \{\mathbf{B}\}^{\otimes -n\,T} \right\} \mathbf{Q}^{(m)\,T} \\ &= \left[\tilde{\mathbf{G}}_{(n)}^{(n_{i_n})} \left\{ \{\mathbf{B}\}^{\otimes -n\,T} \right\} \tilde{\mathbf{G}}_{(m)}^{(m_{i_m})\,T} \right]_{\substack{i_n=1,2,\ldots,I_n \\ i_m=1,2,\ldots,I_m}} \\ &= \left[\langle \mathcal{W}^{(n,m)} \bar{\times}_{m+1} \boldsymbol{a}_{i_m:}^{(m)T} \times_{n+1} \boldsymbol{a}_{i_n:}^{(n)\,T}, \mathcal{G} \rangle_{-n,-m} \right]_{\substack{i_n=1,2,\ldots,I_n \\ i_m=1,2,\ldots,I_m}}, \end{aligned} \tag{4.91}$$

where $\langle \mathcal{Y}, \mathcal{G} \rangle_{-n,-m}$ denotes the contracted product along all modes except mode-n for tensor $\mathcal{Y}$, and except mode-m for the tensor $\mathcal{G}$. Similarly, we straighforwardly express submatrices $\mathbf{H}^{(n,N+1)} = \mathbf{J}_n^T \mathbf{J}_{N+1}$ as follows

$$\mathbf{H}^{(n,N+1)} = \mathbf{P}_{R_n,I_n} \begin{bmatrix} \left(\mathcal{Z}^{(n)} \times_{n+1} \boldsymbol{a}_{1:}^{(n)\,T}\right)_{(n)} \\ \vdots \\ \left(\mathcal{Z}^{(n)} \times_{n+1} \boldsymbol{a}_{i_n:}^{(n)\,T}\right)_{(n)} \\ \vdots \\ \left(\mathcal{Z}^{(n)} \times_{n+1} \boldsymbol{a}_{I_n:}^{(n)\,T}\right)_{(n)} \end{bmatrix}, \tag{4.92}$$

where $\mathcal{Z}^{(n)} = \hat{\mathcal{G}} \times_{-\binom{n,}{n+1}} \{\mathbf{A}^T\mathbf{A}\}$.

4.5.2.2 Diagonal Block Matrices $\mathbf{H}^{(n,n)}$

A diagonal block matrix $\mathbf{H}^{(n,n)} = \mathbf{J}_n^T \mathbf{J}_n$ can be expressed by

$$\mathbf{H}^{(n,n)} = \begin{cases} \langle \mathcal{G} \times_{-n} \{\mathbf{A}^T\mathbf{A}\}, \mathcal{G} \rangle_{-n} \otimes \mathbf{I}_{I_n}, & n \neq N+1, \\ \{\mathbf{A}^T\,\mathbf{A}\}^{\otimes}, & n = N+1. \end{cases} \tag{4.93}$$

From (4.91), (4.92), and (4.93), the approximate Hessian $\mathbf{H} = \mathbf{J}^T\mathbf{J}$ is fully expressed by products of tensors and contracted products. We note that matrices $\mathbf{A}^{(n)T}\mathbf{A}^{(n)}$ have size of $R_n \times R_n$, and much smaller than matrices $\mathbf{A}^{(n)}$ due to $R_n \ll I_n$. This construction avoids computing large-scale Jacobian $\mathbf{J}$, and Kronecker products of large-scale matrices such as $\{\mathbf{A}\}^{\otimes_{-n}}$ or $\{\mathbf{A}\}^{\otimes}$.

Finally, from Sections (4.3.1) and (4.5.2) we completely bypass the Jacobian and establish a much faster computation for the approximate Hessian and the gradient than their conventional approach given in (4.81) and (4.80). Moreover, the proposed method does not demand significant temporary extra-storage. Selection of the regularization parameter α and the damping parameter μ can employ methods presented in Sections 4.2.4, 4.3.3[167;169]. For nonnegative Tucker composition, the gradient and the diagonal of the Hessian are modified as in (4.80) and (4.81).

4.6 Summary

Most CP and Tucker algorithms incorporated with line-search techniques work well for general data, but often fail for highly collinear data with bottlenecks or swamps. In this chapter, a suit of robust all-at-once algorithms has been proposed based on the Gauss-Newton iteration. However, because the approximate Hessian matrices are rank-deficient, the damped GN (or LM) method has been applied. The proposed algorithms can work with complex-valued tensors, and especially are robust for highly collinear tensors. Extensive experiments for tensor factorizations and in CDMA application in Chapter 6 showed that our algorithms overwhelmingly outperform "state-of-the-art" algorithms for difficult benchmarks with bottlenecks, swamps for both real and complex-valued tensors.

A4.1 Appendix: Permutation Matrix

Vectorization of an $I \times J$ matrix $\mathbf{X}$ can be arranged from its transpose matrix $\mathbf{X}^T$ by an $IJ \times IJ$ permutation matrix $\mathbf{P}_{I,J}$: $\text{vec}(\mathbf{X}) = \mathbf{P}_{I,J}\,\text{vec}(\mathbf{X}^T)$ with $\mathbf{P}_{J,I} = \mathbf{P}_{I,J}^T = \mathbf{P}_{I,J}^{-1}$[95]. The following properties are frequently employed in this chapter.

Theorem 4.11 (Interchange in Kronecker product[95]). *Consider the permutation matrices $\mathbf{P}_{I,P}$ and $\mathbf{P}_{J,Q}$, and $\mathbf{A} \in \mathbb{R}^{I\times J}$ and $\mathbf{B} \in \mathbb{R}^{P\times Q}$. Then*

$$\mathbf{A} \otimes \mathbf{B} = \mathbf{P}_{I,P}^T\,(\mathbf{B} \otimes \mathbf{A})\,\mathbf{P}_{J,Q}\,. \tag{4.94}$$

Theorem 4.12 (Concatenation of Kronecker products)**.** *Concatenation of the Kronecker products forms a Kronecker product of the concatenation matrix*

$$\begin{aligned} [\mathbf{A}\otimes\mathbf{C} \quad \mathbf{D}\otimes\mathbf{C}] &= [\mathbf{A} \quad \mathbf{D}]\otimes\mathbf{C}\,, && (4.95)\\ [\mathbf{A}\otimes\mathbf{B} \quad \mathbf{A}\otimes\mathbf{C}] &= (\mathbf{A}\otimes[\mathbf{B} \quad \mathbf{C}])\,\mathbf{P}\,, && (4.96)\end{aligned}$$

where $\mathbf{A}\in\mathbb{R}^{I\times J}$, $\mathbf{B}\in\mathbb{R}^{P\times Q}$, $\mathbf{C}\in\mathbb{R}^{P\times R}$ *and* $\mathbf{D}\in\mathbb{R}^{I\times K}$ *and* $\mathbf{P}$ *denotes a permutation matrix.*

Proof. Proof of (4.95) is directly derived from the definition of the Kronecker product

$$\begin{aligned}[\mathbf{A}\otimes\mathbf{C} \quad \mathbf{D}\otimes\mathbf{C}] &= \left[\begin{array}{cccccccc} \boldsymbol{a}_1\otimes\mathbf{C} & \cdots & \boldsymbol{a}_J\otimes\mathbf{C} & \boldsymbol{d}_1\otimes\mathbf{C} & \cdots & \boldsymbol{d}_K\otimes\mathbf{C}\end{array}\right]\\ &= \left[\begin{array}{cccccc} \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_J & \boldsymbol{d}_1 & \cdots & \boldsymbol{d}_K\end{array}\right]\otimes\mathbf{C} = \left[\begin{array}{cc}\mathbf{A} & \mathbf{D}\end{array}\right]\otimes\mathbf{C}\,. \qquad (4.97)\end{aligned}$$

By employing Theorem 4.11, the second concatenation (4.96) is proved as follows

$$\begin{aligned}[\mathbf{A}\otimes\mathbf{B} \;\; \mathbf{A}\otimes\mathbf{C}] &= \left[\mathbf{P}_{P,I}^T\,(\mathbf{B}\otimes\mathbf{A})\,\mathbf{P}_{Q,J} \quad \mathbf{P}_{P,I}^T\,(\mathbf{C}\otimes\mathbf{A})\,\mathbf{P}_{R,J}\right] = \mathbf{P}_{P,I}^T\,([\mathbf{B} \;\; \mathbf{C}]\otimes\mathbf{A})\begin{bmatrix}\mathbf{P}_{Q,J} & \\ & \mathbf{P}_{R,J}\end{bmatrix}\\ &= \mathbf{P}_{P,I}^T\mathbf{P}_{P,I}\,(\mathbf{A}\otimes[\mathbf{B} \;\; \mathbf{C}])\,\mathbf{P}_{Q+R,J}^T\begin{bmatrix}\mathbf{P}_{Q,J} & \\ & \mathbf{P}_{R,J}\end{bmatrix} = (\mathbf{A}\otimes[\mathbf{B} \quad \mathbf{C}])\,\mathbf{P}\,.\end{aligned}$$

Hence, $\mathbf{P} = \mathbf{P}_{Q+R,J}^T\,\mathrm{blkdiag}\{\mathbf{P}_{Q,J},\mathbf{P}_{R,J}\}$. □

Theorem 4.12 can be generalized to concatenate multiple Kronecker products. For a special case in which the second terms in the Kronecker products are vectors, we have the following expression

$$\left[\begin{array}{ccccccc}\mathbf{A}\otimes\boldsymbol{b}_1 & \cdots & \mathbf{A}\otimes\boldsymbol{b}_q & \cdots & \mathbf{A}\otimes\boldsymbol{b}_Q\end{array}\right] = (\mathbf{A}\otimes\mathbf{B})\,\mathbf{P}_{J,Q}\,. \qquad (4.98)$$

A4.2 Appendix: Commutation matrix $\mathbf{P}_n$

This appendix presents connection between vectorizations of tensor unfoldings via a permutation matrix. The commutation matrix $\mathbf{P}_n$ often exists in construction of the Jacobian $\mathbf{J}$ and the approximate Hessian $\mathbf{H}$ in dGN algorithms for CP, NTF and Tucker decompositions[168;169].

Lemma 4.1. (mode-n to mode-1 unfolding) *Commutation matrix* $\mathbf{P}_n$ *maps* $\mathrm{vec}\big(\mathbf{A}_{(1)}\big) = \mathrm{vec}(\mathcal{A}) = \mathbf{P}_n\,\mathrm{vec}\big(\mathbf{A}_{(n)}\big)$, *given by*

$$\mathbf{P}_n = \mathbf{I}_{I_{n+1:N}}\otimes\mathbf{P}_{I_n,I_{1:n-1}}, \qquad \textit{with } I_{i:j} = \prod_{k=i}^{j} I_k. \qquad (4.99)$$

Theorem 4.13 (Permutation in a Kronecker product)**.** *Multiplication of* $\mathbf{P}_n$ *with a Kronecker product of N vectors* $\boldsymbol{a}^{(n)}\in\mathbb{R}^{I_n}$, $(n=1,2,\ldots,N)$ *will move the n-th item* $\boldsymbol{a}^{(n)}$ *to the last location, that is*

$$\left(\boldsymbol{a}^{(N)}\otimes\cdots\otimes\boldsymbol{a}^{(2)}\otimes\boldsymbol{a}^{(1)}\right) = \mathbf{P}_n\left(\boldsymbol{a}^{(N)}\otimes\cdots\otimes\boldsymbol{a}^{(n+1)}\otimes\boldsymbol{a}^{(n-1)}\otimes\cdots\otimes\boldsymbol{a}^{(1)}\otimes\boldsymbol{a}^{(n)}\right). \qquad (4.100)$$

Proof. We assume a rank-one tensor $\mathcal{X}$ composed by N components $\boldsymbol{a}^{(n)}$, for $n = 1, 2, \ldots, N$. Noting that $\text{vec}\big(\mathbf{X}_{(1)}\big) = \mathbf{P}_n \text{ vec}\big(\mathbf{X}_{(n)}\big)$, we have

$$\begin{aligned} \text{vec}\big(\mathbf{X}_{(n)}\big) &= \text{vec}\Big(\boldsymbol{a}^{(n)} \Big(\boldsymbol{a}^{(N)} \otimes \cdots \otimes \boldsymbol{a}^{(n+1)} \otimes \boldsymbol{a}^{(n-1)} \otimes \cdots \otimes \boldsymbol{a}^{(1)}\Big)^T\Big) \\ &= \boldsymbol{a}^{(N)} \otimes \cdots \otimes \boldsymbol{a}^{(n+1)} \otimes \boldsymbol{a}^{(n-1)} \otimes \cdots \otimes \boldsymbol{a}^{(1)} \otimes \boldsymbol{a}^{(n)} \qquad (4.101) \\ &= \mathbf{P}_n^T \text{ vec}\big(\mathbf{X}_{(1)}\big) = \mathbf{P}_n^T (\boldsymbol{a}^{(N)} \otimes \cdots \otimes \boldsymbol{a}^{(n+1)} \otimes \boldsymbol{a}^{(n)} \otimes \boldsymbol{a}^{(n-1)} \otimes \cdots \otimes \boldsymbol{a}^{(1)}) \qquad (4.102) \end{aligned}$$

The proof follows directly from (4.101) and (4.102). □

A4.3 Appendix: Inverse of the Kernel Matrix K

Lemma 4.2. *Inverse of a sparse block matrix* $\mathbf{V} = \big[\mathbf{V}^{(n,m)}\big]_{n,m}$, $\mathbf{V}^{(n,m)} = (1 - \delta_{n,m})\, \mathbf{I}_{R^2}$ *is an* $N \times N$ *block matrix* $\widetilde{\mathbf{V}}$ *with*

$$\widetilde{\mathbf{V}}^{(n,m)} = \left(\frac{1}{N-1} - \delta_{n,m}\right) \mathbf{I}_{R^2}\,. \qquad (4.103)$$

Proof. It is straighforward to prove that $\sum_{l=1}^{N} \widetilde{\mathbf{V}}^{(n,l)}\, \mathbf{V}^{(l,m)} = \sum_{l=1}^{N} \left(\frac{1}{N-1} - \delta_{n,l}\right) (1 - \delta_{l,m})\, \mathbf{I}_{R^2} = \delta_{n,m}\, \mathbf{I}_{R^2}$. □

Theorem 4.14. *Inverse of* $\mathbf{K}$ *defined in (4.6) is a partitioned matrix* $\widetilde{\mathbf{K}} = \mathbf{K}^{-1}$ *whose blocks* $\widetilde{\mathbf{K}}^{(n,m)}$, *for* $n = 1, \ldots, N, m = 1, \ldots, N$ *are given by*

$$\widetilde{\mathbf{K}}^{(n,m)} = \left(\frac{1}{N-1} - \delta_{n,m}\right) \text{diag}\left(\text{vec}\Big(\mathbf{C}^{(n)} \circledast \mathbf{C}^{(m)} \oslash \boldsymbol{\Gamma}\Big)\right) \mathbf{P}_{R,R}. \qquad (4.104)$$

Proof. Each block $\mathbf{K}^{(n,m)} = (1 - \delta_{n,m})\mathbf{P}_{R,R} \text{ diag}\left(\text{vec}\Big(\boldsymbol{\Gamma}^{(n,m)}\Big)\right)$ in (4.6) can be rewritten as

$$\mathbf{K}^{(n,m)} = (1 - \delta_{n,m})\mathbf{P}_{R,R} \text{ diag}\,(\boldsymbol{\gamma}) \text{ diag}\left(\mathbf{1} \oslash \text{vec}\Big(\mathbf{C}^{(n)}\Big)\right) \text{ diag}\left(\mathbf{1} \oslash \text{vec}\Big(\mathbf{C}^{(m)}\Big)\right),$$

where $\mathbf{C}^{(n)} = \mathbf{A}^{(n)T}\mathbf{A}^{(n)}$, $\boldsymbol{\Gamma} = \{\mathbf{C}\}^{\circledast}$, $\boldsymbol{\gamma} = \text{vec}(\boldsymbol{\Gamma})$. Therefore, this matrix $\mathbf{K}$ can be decomposed into product of matrices

$$\mathbf{K} = (\mathbf{I}_N \otimes (\mathbf{P}_{R,R} \text{ diag}\,(\boldsymbol{\gamma}))) \text{ diag}(\boldsymbol{\beta})\, \mathbf{V} \text{ diag}(\boldsymbol{\beta}) \qquad (4.105)$$

where $\mathbf{V}$ is the partitioned matrix defined in Theorem 4.2, and $\boldsymbol{\beta} = \mathbf{1} \oslash \Big[\text{vec}\big(\mathbf{C}^{(n)}\big)^T\Big]_{n=1}^{N\ \ T}$. Hence, inverse of $\mathbf{K}$ is given by

$$\widetilde{\mathbf{K}} = \mathbf{K}^{-1} = \text{diag}(\boldsymbol{\beta})^{-1}\, \mathbf{V}^{-1} \text{ diag}(\boldsymbol{\beta})^{-1}\, (\mathbf{I}_N \otimes (\text{diag}(\mathbf{1} \oslash \boldsymbol{\gamma})\, \mathbf{P}_{R,R}))\,. \qquad (4.106)$$

By replacing $\mathbf{V}^{-1}$ given in Lemma 4.2 to that in (4.106), we obtain

$$\begin{aligned} \widetilde{\mathbf{K}}^{(n,m)} &= \text{diag}\left(\text{vec}\Big(\mathbf{C}^{(n)}\Big)\right) \widetilde{\mathbf{V}}^{(n,m)} \text{ diag}\left(\text{vec}\Big(\mathbf{C}^{(m)}\Big)\right) \text{diag}(\mathbf{1} \oslash \boldsymbol{\gamma})\, \mathbf{P}_{R,R} \\ &= \left(\frac{1}{N-1} - \delta_{n,m}\right) \text{diag}\left(\text{vec}\Big(\mathbf{C}^{(n)} \circledast \mathbf{C}^{(m)} \oslash \boldsymbol{\Gamma}\Big)\right) \mathbf{P}_{R,R}. \qquad (4.107) \end{aligned}$$

□

CHAPTER 5

Large-Scale Tensor Factorization

5.1 Introduction and Problem Statement

This chapter presents CP factorization suitable for large-scale problems and fast parallel implementation. The proposed model and algorithms solve till now intractable problem for arbitrary high dimension and large-scale tensors. We divide a given data tensor into a grid of multiple sub-tensors (see Figure 5.2). We factorize all the sub-tensors independently by using efficient CP algorithms in parallel mode and next integrate partial results for the whole tensor to estimate desired factors.

The developed model is referred here to as the grid-tensor factorization (gTF). The simplest case of the gTF is factorization for a large-scale tensor partitioned only into two sub-tensors. If one sub-tensor is considered as a new data coming and expanding the current (actual) data along a single mode (usually time), the problem is referred to as the dynamic tensor analysis (DTA) or dynamic tensor factorization[190;191]. This problem also often arises in factorization of a training dataset.

Tensor factorizations can be used as dimensionality reduction methods in multiway classification, and factors are bases to project a tensor data onto the feature subspace. In practice the training data is often augmented with some new samples, hence the basis factors for the expanded training tensor need to be updated. A simple way is to factorize the new whole training tensor again. However, this approach demands high computational cost. A convenient way is that we update the old bases with factors of the new coming data.

The proposed algorithm calculates Hadamard products, multiplication of small matrices, and avoids Khatri-Rao products. Especially, this new algorithm opens new perspectives to find nonnegative factors for the very large-scale nonnegative tensors that is potentially useful in many applications, such as those in neuroscience and data mining. For specific data, such as spectral data, images, chemical concentrations, in order to provide meaningful representation and clear physical interpretation, sparsity and nonnegative constraints are often imposed on the hidden factors. The extracted basis components are part-based representations of the nonnegative data. However, the constrained nonnegative CP, also called Nonnegative Tensor Factorization (NTF) is not guaranteed to converge to a stationary point. Hence, the NTF often needs to be analyzed with many trials, and different regularized values to choose the desired solutions. This demands high computational cost due to often accessing to the input data. In practice, a large data tensor can be well explained by CP factors, then it is more efficient

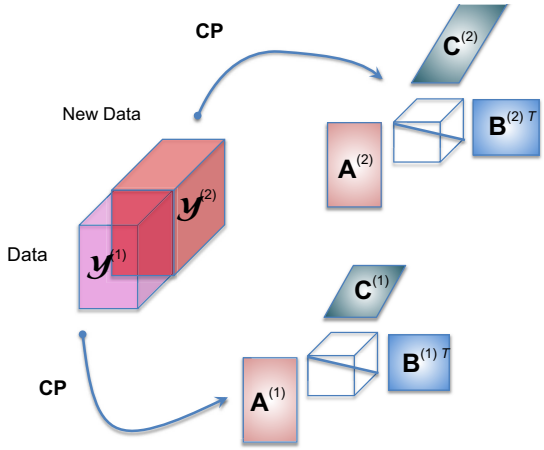


Figure 5.1: Illustration of the dynamic tensor factorization for two sub-tensors. The purpose is to find the common factors for the concatenated tensor $\mathcal{Y}$ constructed from two sub-tensors $\mathcal{Y}^{(1)}$ and $\mathcal{Y}^{(2)}$.

to store the hidden factors, and process them instead of the whole data tensor. The ALS CP algorithm is a "work-horse" algorithm[33;106;203]. However, it is not suitable for very large-scale problems. By using the estimated factors by the modified ALS, we develop the very fast algorithm to retrieve the nonnegative factor hidden from a data tensor.

An important key factor for large-scale factorization is evaluation of stopping criteria to control convergence. The cost function value or the fit error is often used. However, for a large-scale tensor, an explicit computation of the cost function value is impossible due to so much memory requirement to build up the approximate tensor $\hat{\mathcal{Y}}$. We derive a simple low complexity formula for stopping criteria applied to the grid tensor factorizations.

5.2 Dynamic Tensor Factorization

For simplicity of explanation of our basic concept, we shall first consider the grid CP for a three dimensional tensor partitioned into only two sub-tensors. This model can be also considered as a Dynamic Tensor Analysis (Factorization) in which data is augmented along a single mode.

Problem 5.1 (Dynamic Tensor Factorization)
Consider two data tensors $\mathcal{Y}^{(1)} \in \mathbb{R}^{I\times J\times K_1}$, $\mathcal{Y}^{(2)} \in \mathbb{R}^{I\times J\times K_2}$ *with their approximated CP models:*

$$\mathcal{Y}^{(k)} \approx \mathcal{I} \times_1 \mathbf{A}^{(k)} \times_2 \mathbf{B}^{(k)} \times_3 \mathbf{C}^{(k)}, \quad k = 1, 2, \tag{5.1}$$

where $\mathbf{A}^{(k)} \in \mathbb{R}^{I\times R_k}$, $\mathbf{B}^{(k)} \in \mathbb{R}^{J\times R_k}$ *and* $\mathbf{C}^{(k)} \in \mathbb{R}^{K_k\times R_k}$.*The problem is to find CP factors for the concatenated tensor* $\mathcal{Y} \in \mathbb{R}^{I\times J\times K}$ *(with* $K = K_1 + K_2$*):*

$$\mathcal{Y} \approx \mathcal{I} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}, \tag{5.2}$$

without repeating all computation from the beginning, but using already known partial results (factors).

The easiest way is that we factorize the concatenation tensor again. This method provides accurate results. However, for large-scale data, this is not a suitable solution due to high computational cost. In this section, we present three methods to deal with this problem[159]. The final learning rules of all the methods are identical.

1. By minimizing the cost function for the concatenated tensor.
2. By adopting or modifying the ALS algorithm for the concatenated tensor represented by block matrices.
3. By concatenating factors of sub-tensors and then reducing the number of components.

5.2.1 Approach 1: By Directly Minimizing a Cost Function

Assume that the concatenation tensor $\mathcal{Y}$ can be factorized by R components: $\mathbf{A} \in \mathbb{R}^{I\times R}$, $\mathbf{B} \in \mathbb{R}^{J\times R}$ and $\mathbf{C} \in \mathbb{R}^{K\times R}$. If we split the factor $\mathbf{C}$ into two matrices $\mathbf{C} = \left[\mathbf{C}_{(1)}^T \; \mathbf{C}_{(2)}^T\right]^T$, $\mathbf{C}_{(1)} \in \mathbb{R}^{K_1\times R}$, and $\mathbf{C}_{(2)} \in \mathbb{R}^{K_2\times R}$, the subtensors $\mathcal{Y}^{(1)}$ and $\mathcal{Y}^{(2)}$ are now approximated via their common factors as

$$\mathcal{Y}^{(1)} = \mathcal{I} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}_{(1)}\,, \tag{5.3}$$

$$\mathcal{Y}^{(2)} = \mathcal{I} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}_{(2)}\,. \tag{5.4}$$

Generally, a tensor factorization can be solved by minimizing a cost function of the concatenation tensor and its approximation, that is

$$\begin{aligned} D &= \frac{1}{2}\|\mathcal{Y} - \mathcal{I} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}\|_F^2 = \frac{1}{2}\sum_{k=1}^{2} \|\mathcal{Y}^{(k)} - \mathcal{I} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}_{(k)}\|_F^2 \\ &= \frac{1}{2}\sum_{k=1}^{2} \|\mathbf{Y}_{(1)}^{(k)} - \mathbf{A}\,\left(\mathbf{C}_{(k)} \odot \mathbf{B}\right)^T\|_F^2\,. \end{aligned} \tag{5.5}$$

In order to find the learning rule for factor $\mathbf{A}$, we formulate the cost function in the mode-1 matricization (5.5), and compute its gradient with respect to $\mathbf{A}$

$$\begin{aligned} \nabla_{\mathbf{A}} D &= \sum_{k=1}^{2} \left(-\mathbf{Y}_{(1)}^{(k)}\left(\mathbf{C}_{(k)} \odot \mathbf{B}\right) + \mathbf{A}\,\left(\mathbf{C}_{(k)} \odot \mathbf{B}\right)^T\left(\mathbf{C}_{(k)} \odot \mathbf{B}\right)\right) \\ &= \sum_{k=1}^{2} \left(-\mathbf{Y}_{(1)}^{(k)}\,\left(\mathbf{C}_{(k)} \odot \mathbf{B}\right) + \mathbf{A}\,\left(\mathbf{C}_{(k)}^T\mathbf{C}_{(k)}\right) \circledast \left(\mathbf{B}^T\mathbf{B}\right)\right) \\ &= \sum_{k=1}^{2} -\mathbf{Y}_{(1)}^{(k)}\,\left(\mathbf{C}_{(k)} \odot \mathbf{B}\right) + \mathbf{A}\,\left(\mathbf{C}^T\mathbf{C}\right) \circledast \left(\mathbf{B}^T\mathbf{B}\right)\,. \end{aligned} \tag{5.6}$$

By replacing $\mathcal{Y}^{(1)}$ and $\mathcal{Y}^{(2)}$ in (5.6) by their approximations in (5.1), and setting (5.6) to zero, we obtain an update rule for $\mathbf{A}$

$$\begin{aligned}
\mathbf{A} &\leftarrow \left(\sum_{k=1}^{2} \mathbf{Y}_{(1)}^{(k)} \left(\mathbf{C}_{(k)} \odot \mathbf{B}\right)\right)\left(\left(\mathbf{C}^T\mathbf{C}\right) \circledast \left(\mathbf{B}^T\mathbf{B}\right)\right)^{-1} \\
&= \left(\sum_{k=1}^{2} \mathbf{A}^{(k)} \left(\mathbf{C}^{(k)} \odot \mathbf{B}^{(k)}\right)^T \left(\mathbf{C}_{(k)} \odot \mathbf{B}\right)\right)\left(\left(\mathbf{C}^T\mathbf{C}\right) \circledast \left(\mathbf{B}^T\mathbf{B}\right)\right)^{-1} \\
&= \left(\sum_{k=1}^{2} \mathbf{A}^{(k)} \left(\mathbf{P}^{(k)} \oslash \left(\mathbf{A}^{(k)T}\mathbf{A}\right)\right)\right)\left(\mathbf{Q} \oslash \left(\mathbf{A}^T\mathbf{A}\right)\right)^{-1}, \qquad (5.7)
\end{aligned}$$

where

$$\mathbf{P}^{(k)} = \left(\mathbf{A}^{(k)T}\mathbf{A}\right) \circledast \left(\mathbf{B}^{(k)T}\mathbf{B}\right) \circledast \left(\mathbf{C}^{(k)T}\mathbf{C}\right), \qquad (5.8)$$

$$\mathbf{Q} = \left(\mathbf{A}^T\mathbf{A}\right) \circledast \left(\mathbf{B}^T\mathbf{B}\right) \circledast \left(\mathbf{C}^T\mathbf{C}\right). \qquad (5.9)$$

Similarly, learning rule for $\mathbf{B}$ is given by

$$\mathbf{B} \leftarrow \left(\sum_{k=1}^{2} \mathbf{B}^{(k)} \left(\mathbf{P}^{(k)} \oslash \left(\mathbf{B}^{(k)T}\mathbf{B}\right)\right)\right)\left(\mathbf{Q} \oslash \left(\mathbf{B}^T\mathbf{B}\right)\right)^{-1}. \qquad (5.10)$$

Factor $\mathbf{C}$ is estimated via partially updating $\mathbf{C}_{(1)}$, and $\mathbf{C}_{(2)}$. The ALS learning rule to update $\mathbf{C}_{(k)}$ from the tensor $\boldsymbol{\mathcal{Y}}^{(k)}$ is given by

$$\begin{aligned}
\mathbf{C}_{(k)} &\leftarrow \mathbf{Y}_{(3)}^{(k)} \left(\mathbf{B} \odot \mathbf{A}\right)\left(\left(\mathbf{B}^T\mathbf{B}\right) \circledast \left(\mathbf{A}^T\mathbf{A}\right)\right)^{-1} = \mathbf{C}^{(k)} \left(\mathbf{B}^{(k)} \odot \mathbf{A}^{(k)}\right)^T \left(\mathbf{B} \odot \mathbf{A}\right)\left(\left(\mathbf{B}^T\mathbf{B}\right) \circledast \left(\mathbf{A}^T\mathbf{A}\right)\right)^{-1} \\
&= \mathbf{C}^{(k)} \left(\mathbf{P}^{(k)} \oslash \left(\mathbf{C}^{(k)T}\mathbf{C}\right)\right)\left(\mathbf{Q} \oslash \left(\mathbf{C}^T\mathbf{C}\right)\right)^{-1}. \qquad (5.11)
\end{aligned}$$

We note that matrices $\mathbf{P}^{(k)} \in \mathbb{R}^{R_k \times R}$ and $\mathbf{Q} \in \mathbb{R}^{R \times R}$ are relatively small size. Hence, the learning rules (5.7), (5.10) and (5.11) are low computational cost.

5.2.2 Approach 2: By Modifying the ALS Learning Rule

Factors $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ can be estimated by using the ALS algorithm (2.2) for the concatenation tensor $\boldsymbol{\mathcal{Y}}$

$$\mathbf{A} \leftarrow \mathbf{Y}_{(1)} \left(\mathbf{C} \odot \mathbf{B}\right)\left(\left(\mathbf{C}^T\mathbf{C}\right) \circledast \left(\mathbf{B}^T\mathbf{B}\right)\right)^{-1}. \qquad (5.12)$$

By using the following relation, we can convert the ALS update rule to one given in (5.7)

$$\begin{aligned}
\mathbf{Y}_{(1)} \left(\mathbf{C} \odot \mathbf{B}\right) &= \left[\mathbf{Y}_{(1)}^{(1)} \ \mathbf{Y}_{(1)}^{(2)}\right]\left(\begin{bmatrix} \mathbf{C}_{(1)} \\ \mathbf{C}_{(2)} \end{bmatrix} \odot \mathbf{B}\right) = \left[\mathbf{Y}_{(1)}^{(1)} \ \mathbf{Y}_{(1)}^{(2)}\right]\begin{bmatrix} \mathbf{C}_{(1)} \odot \mathbf{B} \\ \mathbf{C}_{(2)} \odot \mathbf{B} \end{bmatrix} \\
&= \sum_{k=1}^{2} \mathbf{A}^{(k)} \left(\mathbf{C}^{(k)} \odot \mathbf{B}^{(k)}\right)^T \left(\mathbf{C}_{(k)} \odot \mathbf{B}\right) = \sum_{k=1}^{2} \mathbf{A}^{(k)} \left(\left(\mathbf{C}^{(k)T}\mathbf{C}_{(k)}\right) \circledast \left(\mathbf{B}^{(k)T}\mathbf{B}\right)\right).
\end{aligned}$$

5.2.3 Approach 3: By Concatenating Sub-factors

Another method to find factors of the concatenation tensor $\mathcal{Y}$ is that we build up a factorization from concatenation factors. The following lemma will show this factorization.

Lemma 5.1 (Concatenation of factors)**.** *Tensor $\mathcal{Y}$ is factorized by factors of $\tilde{R} = R_1 + R_2$ components*

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A}^{(1)} & \mathbf{A}^{(2)} \end{bmatrix}, \qquad \tilde{\mathbf{B}} = \begin{bmatrix} \mathbf{B}^{(1)} & \mathbf{B}^{(2)} \end{bmatrix}, \qquad \tilde{\mathbf{C}} = \begin{bmatrix} \mathbf{C}^{(1)} & \\ & \mathbf{C}^{(2)} \end{bmatrix}, \tag{5.13}$$

that is

$$\mathcal{Y} = \mathcal{I} \times_1 \tilde{\mathbf{A}} \times_2 \tilde{\mathbf{B}} \times_3 \tilde{\mathbf{C}}. \tag{5.14}$$

Proof. We consider the mode-1 matricized version of the concatenation tensor $\mathcal{Y}$

$$\begin{aligned}
\mathbf{Y}_{(1)} &= \begin{bmatrix} \mathbf{Y}_{(1)}^{(1)} & \mathbf{Y}_{(1)}^{(2)} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^{(1)} \left(\mathbf{C}^{(1)} \odot \mathbf{B}^{(1)}\right)^T & \mathbf{A}^{(2)} \left(\mathbf{C}^{(2)} \odot \mathbf{B}^{(2)}\right)^T \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{A}^{(1)} & \mathbf{A}^{(2)} \end{bmatrix} \begin{bmatrix} \left(\mathbf{C}^{(1)} \odot \mathbf{B}^{(1)}\right)^T & \\ & \left(\mathbf{C}^{(2)} \odot \mathbf{B}^{(2)}\right)^T \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{A}^{(1)} & \mathbf{A}^{(2)} \end{bmatrix} \begin{bmatrix} \left(\begin{bmatrix} \mathbf{C}^{(1)} \\ \mathbf{0} \end{bmatrix} \odot \mathbf{B}^{(1)}\right) & \left(\begin{bmatrix} \mathbf{0} \\ \mathbf{C}^{(2)} \end{bmatrix} \odot \mathbf{B}^{(2)}\right) \end{bmatrix}^T \\
&= \begin{bmatrix} \mathbf{A}^{(1)} & \mathbf{A}^{(2)} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} \mathbf{C}^{(1)} & \\ & \mathbf{C}^{(2)} \end{bmatrix} \odot \begin{bmatrix} \mathbf{B}^{(1)} & \mathbf{B}^{(2)} \end{bmatrix} \end{bmatrix}^T = \tilde{\mathbf{A}} \left(\tilde{\mathbf{C}} \odot \tilde{\mathbf{B}}\right)^T.
\end{aligned} \tag{5.15}$$

This leads to a tensor factorization of $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}$ and $\tilde{\mathbf{C}}$ for the tensor $\mathcal{Y}$ given in (5.14). □

Lemma 5.1 ensures that any large-scale tensor can be factorized by concatenating factors of sub-tensors. However, the combined factors often have higher rank than that we need. The next update rule will help us to find R componential factors from factors $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}$.

Factor $\mathbf{A}$ can be updated by using the ALS learning rule (2.2) for tensor $\mathcal{Y}$ as

$$\begin{aligned}
\mathbf{A} &\leftarrow \mathbf{Y}_{(1)} \left(\mathbf{C} \odot \mathbf{B}\right) \left(\left(\mathbf{C}^T\mathbf{C}\right) \circledast \left(\mathbf{B}^T\mathbf{B}\right)\right)^{-1} = \tilde{\mathbf{A}} \left(\tilde{\mathbf{C}} \odot \tilde{\mathbf{B}}\right)^T \left(\mathbf{C} \odot \mathbf{B}\right) \left(\left(\mathbf{C}^T\mathbf{C}\right) \circledast \left(\mathbf{B}^T\mathbf{B}\right)\right)^{-1} \\
&= \tilde{\mathbf{A}} \left(\left(\tilde{\mathbf{C}}^T\mathbf{C}\right) \circledast \left(\tilde{\mathbf{B}}^T\mathbf{B}\right)\right) \left(\left(\mathbf{C}^T\mathbf{C}\right) \circledast \left(\mathbf{B}^T\mathbf{B}\right)\right)^{-1} \\
&= \tilde{\mathbf{A}} \left(\mathbf{P} \oslash \left(\tilde{\mathbf{A}}^T\mathbf{A}\right)\right) \left(\mathbf{Q} \oslash \left(\mathbf{A}^T\mathbf{A}\right)\right)^{-1},
\end{aligned} \tag{5.16}$$

where

$$\mathbf{P} = \left(\tilde{\mathbf{A}}^T\mathbf{A}\right) \circledast \left(\tilde{\mathbf{B}}^T\mathbf{B}\right) \circledast \left(\tilde{\mathbf{C}}^T\mathbf{C}\right). \tag{5.17}$$

We note that the learning rules (5.7), (5.16) are identical due to the following relations

$$\begin{aligned}
\tilde{\mathbf{C}}^T\mathbf{C} &= \begin{bmatrix} \tilde{\mathbf{C}}^{(1)} & \\ & \tilde{\mathbf{C}}^{(2)} \end{bmatrix}^T \begin{bmatrix} \mathbf{C}_{(1)} \\ \mathbf{C}_{(2)} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{C}}^{(1)T}\mathbf{C}_{(1)} \\ \tilde{\mathbf{C}}^{(2)T}\mathbf{C}_{(2)} \end{bmatrix}, \\
\tilde{\mathbf{B}}^T\mathbf{B} &= \begin{bmatrix} \tilde{\mathbf{B}}^{(1)} & \tilde{\mathbf{B}}^{(2)} \end{bmatrix}^T \mathbf{B} = \begin{bmatrix} \tilde{\mathbf{B}}^{(1)T}\mathbf{B} \\ \tilde{\mathbf{B}}^{(2)T}\mathbf{B} \end{bmatrix},
\end{aligned}$$

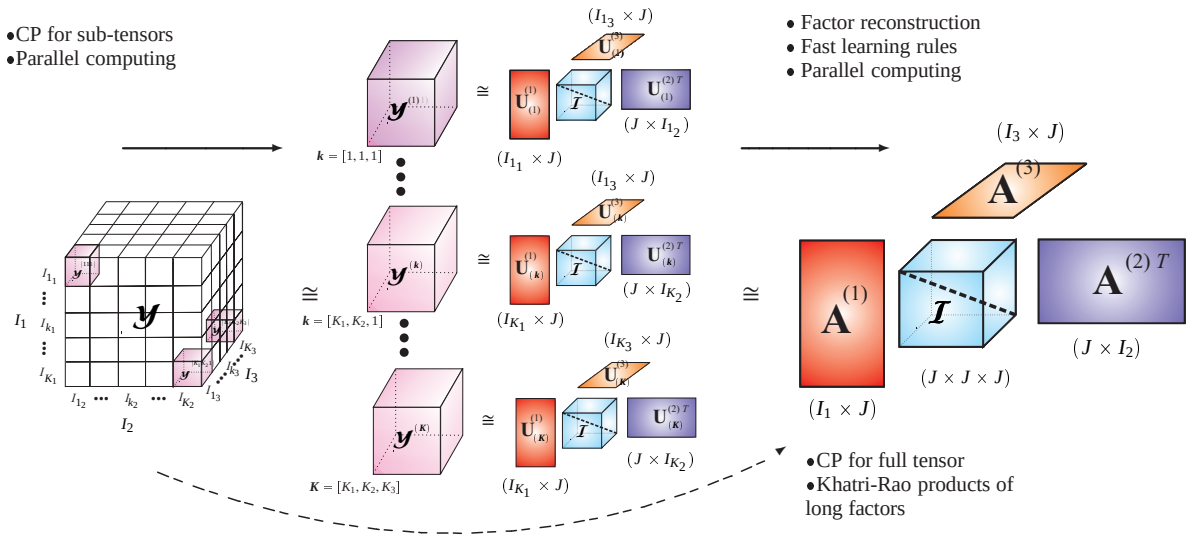


Figure 5.2: Illustration for the standard CP (dash arrow), and grid CP for large-scale tensors (solid arrows) in two stages

and

$$\tilde{\mathbf{A}}\left(\left(\tilde{\mathbf{C}}^T\mathbf{C}\right)\circledast\left(\tilde{\mathbf{B}}^T\mathbf{B}\right)\right)=\sum_{k=1}^{2}\mathbf{A}^{(k)}\left(\tilde{\mathbf{C}}^{(k)T}\mathbf{C}_{(k)}\right)\circledast\left(\tilde{\mathbf{B}}^{(k)T}\mathbf{B}\right).$$

5.3 Grid CP

We consider a tensor $\mathcal{Y}$ is divided into a grid of multiple sub-tensors $\mathcal{Y}^{(\boldsymbol{k})}$ of size $I_{k_1}\times I_{k_2}\times\cdots\times I_{k_N}$, $\sum_{k_n=1}^{K_n} I_{k_n}=I_n$, where vector $\boldsymbol{k}=[k_1,k_2,\ldots,k_N]$ indicates sub-tensor index, $1\leqslant k_n\leqslant K_n$, and K_n is the number of subtensors along the mode-n (see Figure 5.2). We factorize all the subtensors by CP sub-factors $\mathbf{U}^{(n)}_{(\boldsymbol{k})}$ in parallel mode. Finally, the full factors $\mathbf{A}^{(n)}$ for the whole tensor will be estimated from these sub-factors with fast learning rules and parallel computing.

5.3.1 ALS Algorithm for Grid CP

Assuming that the tensor $\mathcal{Y}$ can be approximated by N factors $\mathbf{A}^{(n)}$, we split each factor $\mathbf{A}^{(n)}$ into K_n parts

$$\mathbf{A}^{(n)}=[\mathbf{A}^{(n)\,T}_{(1)}\ \ \mathbf{A}^{(n)\,T}_{(2)}\ \ \ldots\ \ \mathbf{A}^{(n)\,T}_{(K_n)}]^T, \tag{5.18}$$

where sub-factor $\mathbf{A}^{(n)}_{(k_n)}\in\mathbb{R}^{I_{k_n}\times R}, k_n=1,2,\ldots,K_n$.

Lemma 5.2. *A sub-tensor $\mathcal{Y}^{(\boldsymbol{k})}$, $\boldsymbol{k}=[k_1,k_2,\ldots,k_N]$ can be factorized by a set of N sub-factors $\{\mathbf{A}_{(\boldsymbol{k})}\}=\{\mathbf{A}^{(1)}_{(k_1)},\mathbf{A}^{(2)}_{(k_2)},\ldots,\mathbf{A}^{(N)}_{(k_N)}\}$:*

$$\mathcal{Y}^{(\boldsymbol{k})}\ \approx\ \mathcal{I}\times_1\mathbf{A}^{(1)}_{(k_1)}\times_2\mathbf{A}^{(2)}_{(k_2)}\cdots\times_N\mathbf{A}^{(N)}_{(k_N)}=[\![\{\mathbf{A}_{(\boldsymbol{k})}\}]\!]. \tag{5.19}$$

Proof. Proof of this lemma is directly derived from definition of CP model for each entry $y_{\boldsymbol{i}}$, $\boldsymbol{i} = [i_1, i_2, \ldots, i_N]$, $\sum_{l=1}^{k_n-1} I_{k_l} < i_n < \sum_{l=1}^{k_n} I_{k_l}$, i.e., $y_{\boldsymbol{i}} = a^{(1)}_{i_1}\, a^{(2)}_{i_2}\, \cdots\, a^{(N)}_{i_N}$. □

The ALS algorithm for grid CP minimizes the standard Euclidean distance for all the sub-tensors

$$\begin{aligned} D &= \frac{1}{2}\|\mathcal{Y} - \mathcal{I} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)}\|_F^2 = \frac{1}{2}\sum_{k_1=\mathbf{1}}^{K_1} \cdots \sum_{k_N=1}^{K_N} \|\mathcal{Y}^{(\boldsymbol{k})} - [\![\{\mathbf{A}_{(\boldsymbol{k})}\}]\!]\|_F^2 \\ &= \frac{1}{2}\sum_{\boldsymbol{k}=\mathbf{1}}^{\boldsymbol{K}} \|\mathbf{Y}^{(\boldsymbol{k})}_{(n)} - \mathbf{A}^{(n)}_{(k_n)}\, \{\mathbf{A}_{(\boldsymbol{k})}\}^{\odot_{-n}\,T}\|_F^2, \end{aligned} \tag{5.20}$$

where $\boldsymbol{K} = [K_1, K_2, \ldots, K_N]$.

Gradients of (5.20) with respect to sub-factor $\mathbf{A}^{(n)}_{(k_n)}$ are given by

$$\begin{aligned} \nabla_{\mathbf{A}^{(n)}_{(k_n)}} D &= \sum_{\{\boldsymbol{j}|j_n=k_n\}} \left(-\mathbf{Y}^{(\boldsymbol{j})}_{(n)}\, \{\mathbf{A}_{(\boldsymbol{j})}\}^{\odot_{-n}} + \mathbf{A}^{(n)}_{(k_n)}\, \{\mathbf{A}_{(\boldsymbol{j})}\}^{\odot_{-n}\,T}\, \{\mathbf{A}_{(\boldsymbol{j})}\}^{\odot_{-n}}\right) \\ &= \sum_{\{\boldsymbol{j}|j_n=k_n\}} \left(-\mathbf{Y}^{(\boldsymbol{j})}_{(n)}\, \{\mathbf{A}_{(\boldsymbol{j})}\}^{\odot_{-n}} + \mathbf{A}^{(n)}_{(k_n)}\, \left\{\mathbf{A}^T_{(\boldsymbol{j})}\, \mathbf{A}_{(\boldsymbol{j})}\right\}^{\circledast_{-n}}\right), \end{aligned} \tag{5.21}$$

where $\boldsymbol{j} = [j_1, \ldots, j_N]$, and $j_n \equiv k_n$. This leads to the learning rule for sub-factor $\mathbf{A}^{(n)}_{(k_n)}$

$$\mathbf{A}^{(n)}_{(k_n)} \leftarrow \left(\sum_{\{\boldsymbol{j}|j_n=k_n\}} \mathbf{Y}^{(\boldsymbol{j})}_{(n)}\, \{\mathbf{A}_{(\boldsymbol{j})}\}^{\odot_{-n}}\right) \left(\sum_{\{\boldsymbol{j}|j_n=k_n\}} \left\{\mathbf{A}^T_{(\boldsymbol{j})}\, \mathbf{A}_{(\boldsymbol{j})}\right\}^{\circledast_{-n}}\right)^{-1}. \tag{5.22}$$

Due to relatively small sizes of subtensors, computation of $\mathbf{Y}^{(\boldsymbol{j})}_{(n)}\, \{\mathbf{A}_{(\boldsymbol{k})}\}^{\odot_{-n}}$, $\left\{\mathbf{A}^T_{(\boldsymbol{k})}\, \mathbf{A}_{(\boldsymbol{k})}\right\}^{\circledast_{-n}}$ can be quickly executed on parallel workers (labs) or sequentially on a single computer. Moreover, we can eliminate the sub-tensors involving in estimation of sub-factors $\mathbf{A}^{(n)}_{(k_n)}$ to those built up from tubes sampled by the CUR decomposition. We note that subtensors in the grid model don't need to have consecutive tubes. We do not provide a detailed description because this research is beyond the scope of this chapter.

The next section presents optimized algorithm which avoids $\{\mathbf{A}_{(\boldsymbol{k})}\}^{\odot_{-n}}$ in (5.22).

5.3.2 Optimized ALS Learning Rules

For sub-tensor $\mathcal{Y}^{(\boldsymbol{k})}$, we factorize this tensor using the ALS algorithm (2.2) for CP with $R_{\boldsymbol{k}}$ components

$$\mathcal{Y}^{(\boldsymbol{k})} \approx \mathcal{I} \times_1 \mathbf{U}^{(1)}_{(\boldsymbol{k})} \times_2 \mathbf{U}^{(2)}_{(\boldsymbol{k})} \cdots \times_N \mathbf{U}^{(N)}_{(\boldsymbol{k})}. \tag{5.23}$$

The number of rank-one tensors $R_{\boldsymbol{k}}$ should be chosen so that factors $\mathbf{U}^{(n)}_{(\boldsymbol{k})}$ explain as much as possible the sub-tensor $\mathcal{Y}^{(\boldsymbol{k})}$. Because sub-tensor $\mathcal{Y}^{(\boldsymbol{k})}$ has small-size, this factorization can easily achieve high fitness. For a subtensor, we have

$$\mathbf{Y}^{(\boldsymbol{k})}_{(n)}\, \{\mathbf{A}_{(\boldsymbol{k})}\}^{\odot_{-n}} \approx \mathbf{U}^{(n)}_{(\boldsymbol{k})}\, \{\mathbf{U}_{(\boldsymbol{k})}\}^{\odot_{-n}T}\, \{\mathbf{A}_{(\boldsymbol{k})}\}^{\odot_{-n}} = \mathbf{U}^{(n)}_{(\boldsymbol{k})} \left(\mathbf{P}_{(\boldsymbol{k})} \oslash \left(\mathbf{U}^{(n)T}_{(\boldsymbol{k})} \mathbf{A}^{(n)}_{(k_n)}\right)\right), \tag{5.24}$$

where $\mathbf{P}_{(\boldsymbol{k})} = \left\{\mathbf{U}_{(\boldsymbol{k})}^T \mathbf{A}_{(\boldsymbol{k})}\right\}^{\circledast} \in \mathbb{R}^{R_{\boldsymbol{k}} \times R}$. Let $\mathbf{Q}_{(\boldsymbol{k})} = \left\{\mathbf{A}_{(\boldsymbol{k})}^T \mathbf{A}_{(\boldsymbol{k})}\right\}^{\circledast} \in \mathbb{R}^{R \times R}$, from (5.22) and (5.24), we obtain the fast update rule for the sub-factors $\mathbf{A}_{(k_n)}^{(n)}$

$$\mathbf{A}_{(k_n)}^{(n)} \leftarrow \left(\sum_{\{\boldsymbol{j}|j_n=k_n\}} \mathbf{U}_{(\boldsymbol{j})}^{(n)} \left(\mathbf{P}_{(\boldsymbol{j})} \oslash \left(\mathbf{U}_{(\boldsymbol{j})}^{(n)T} \mathbf{A}_{(k_n)}^{(n)}\right)\right)\right)\left(\left(\sum_{\{\boldsymbol{j}|j_n=k_n\}} \mathbf{Q}_{(\boldsymbol{j})}\right) \oslash \left(\mathbf{A}_{(k_n)}^{(n)T} \mathbf{A}_{(k_n)}^{(n)}\right)\right)^{-1}$$
$$= \mathbf{T}\,\mathbf{S}^{-1}. \tag{5.25}$$

where matrices $\mathbf{T}$ and $\mathbf{S}$ are computed for specific subfactors $\mathbf{A}_{(k_n)}^{(n)}$

$$\mathbf{T} = \sum_{\{\boldsymbol{j}|j_n=k_n\}} \mathbf{U}_{(\boldsymbol{j})}^{(n)} \left(\mathbf{P}_{(\boldsymbol{j})} \oslash \left(\mathbf{U}_{(\boldsymbol{j})}^{(n)T} \mathbf{A}_{(k_n)}^{(n)}\right)\right), \tag{5.26}$$

$$\mathbf{S} = \left(\sum_{\{\boldsymbol{j}|j_n=k_n\}} \mathbf{Q}_{(\boldsymbol{j})}\right) \oslash \left(\mathbf{A}_{(k_n)}^{(n)T} \mathbf{A}_{(k_n)}^{(n)}\right). \tag{5.27}$$

Each term of the two summations in (5.26) and (5.27) calculates Hadamard divisions, and performs on small-sized matrices, instead of Khatri-Rao products for tall matrices. Matrices $\mathbf{P}_{(\boldsymbol{k})} \in \mathbb{R}^{R_{\boldsymbol{k}} \times R}$ and $\mathbf{Q}_{(\boldsymbol{k})} \in \mathbb{R}^{R \times R}$ can be calculated only once time, and can be quickly updated after estimating the sub-factors $\mathbf{A}_{k_n}^{(n)}$. Moreover, the matrices $\mathbf{T}$ and $\mathbf{S}$ can be calculated through a parallel loop.

The pseudo-code of the new ALS algorithm is given in Algorithm 5.1. The normalization of components to unit-length vectors are not explicitly displayed in Algorithm 5.1. Parallel FOR-loop denoted by "**parfor**" loop is available with the Matlab Parallel Computing Toolbox.

5.4 Grid CP with Nonnegative Constraints

Factorization of nonnegative tensors such as spectral data, images, data in social networks, email surveillance requires nonnegative factors to provide meaningful components, and physical interpretation. Many efficient algorithms have been proposed or generalized from algorithms for nonnegative matrix factorization (NMF). The ALS[47], HALS[37;152] or multiplicative LS (least squares) algorithms[47;132] can be directly derived from the Frobenius cost function (5.20). However, most efficient algorithms for nonnegative tensor factorization cannot deal with very large-scale data. In this section, we extend grid CP for nonnegative factors, and proposed algorithms based on minimizing the same cost function in (5.20). Sub-tensors are factorized by CP models, then nonnegative factors for the full tensor will be estimated from factors of sub-tensors[154]. We note that if the number of subtensors is only one, the proposed method becomes the two-stage approximation for NTF[155]. That is approximation of tensor by CP first, then estimation of nonnegative components follows.

Algorithm 5.1: Grid CP

Input: $\mathcal{Y}$: input data of size $I_1 \times I_2 \times \cdots \times I_N$,
R: number of basis components
Output: N factors $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$ such that the cost functions (5.20) are minimized.

begin
  initialize $\mathbf{A}^{(n)}_{(k_n)}, \forall n, \forall k_n$
  parfor *sub-tensor* $\mathcal{Y}^{(\boldsymbol{k})}$ **do**
    $[\mathbf{U}^{(1)}_{(\boldsymbol{k})}, \ldots, \mathbf{U}^{(N)}_{(\boldsymbol{k})}] = \texttt{approx\_CP}(\mathcal{Y}^{(\boldsymbol{k})}, R_{\boldsymbol{k}})$
    $\mathbf{P}_{(\boldsymbol{k})} = \left(\mathbf{U}^{(N)T}_{(\boldsymbol{k})}\mathbf{A}^{(N)}_{(k_N)}\right) \circledast \cdots \circledast \left(\mathbf{U}^{(1)T}_{(\boldsymbol{k})}\mathbf{A}^{(1)}_{(k_1)}\right)$
    $\mathbf{Q}_{(\boldsymbol{k})} = \left(\mathbf{A}^{(N)T}_{(k_N)}\mathbf{A}^{(N)}_{(k_N)}\right) \circledast \cdots \circledast \left(\mathbf{A}^{(1)T}_{(k_1)}\mathbf{A}^{(1)}_{(k_1)}\right)$
  endfor

  repeat
    for $n = 1$ *to* N **do**
      for $k_n = 1$ *to* K_n **do**
        $\mathbf{T} = \mathbf{0}, \quad \mathbf{S} = \mathbf{0}$
        parfor $[\boldsymbol{k}]_n = k_n$ **do**
          $\mathbf{P}_{(\boldsymbol{k})} = \mathbf{P}_{(\boldsymbol{k})} \oslash \left(\mathbf{U}^{(n)T}_{(\boldsymbol{k})}\mathbf{A}^{(n)}_{(k_n)}\right)$
          $\mathbf{T} = \mathbf{T} + \mathbf{U}^{(n)}_{(\boldsymbol{k})}\mathbf{P}_{(\boldsymbol{k})}$
          $\mathbf{Q}_{(\boldsymbol{k})} = \mathbf{Q}_{(\boldsymbol{k})} \oslash \left(\mathbf{A}^{(n)T}_{(k_n)}\mathbf{A}^{(n)}_{(k_n)}\right)$
          $\mathbf{S} = \mathbf{S} + \mathbf{Q}_{(\boldsymbol{k})}$
        endfor
        $\mathbf{A}^{(n)}_{(k_n)} \leftarrow \mathbf{T}\,\mathbf{S}^{-1}$ // Update $\mathbf{A}^{(n)}_{(k_n)}$
      end
      parfor *each* $\boldsymbol{k}$ **do**
        $\mathbf{P}_{(\boldsymbol{k})} = \mathbf{P}_{(\boldsymbol{k})} \circledast \left(\mathbf{U}^{(n)T}_{(\boldsymbol{k})}\mathbf{A}^{(n)}_{(k_n)}\right)$
        $\mathbf{Q}_{(\boldsymbol{k})} = \mathbf{Q}_{(\boldsymbol{k})} \circledast \left(\mathbf{A}^{(n)T}_{(k_n)}\mathbf{A}^{(n)}_{(k_n)}\right)$
      endfor
    end
  until *a stopping criterion is met*
end

5.4.1 ALS Algorithm for Grid NTF

A simple extension is that we enforce nonnegativity or strictly speaking positive constraints for factors updated by the ALS algorithm (5.25)

$$\mathbf{A}^{(n)}_{(k_n)} \leftarrow \left[\left(\sum_{\{\boldsymbol{j}|j_n=k_n\}} \mathbf{U}^{(n)}_{(\boldsymbol{j})}\left(\mathbf{P}_{(\boldsymbol{j})} \oslash \left(\mathbf{U}^{(n)T}_{(\boldsymbol{j})}\mathbf{A}^{(n)}_{(k_n)}\right)\right)\right)\left(\left(\sum_{\{\boldsymbol{j}|j_n=k_n\}} \mathbf{Q}_{(\boldsymbol{j})}\right) \oslash \left(\mathbf{A}^{(n)T}_{(k_n)}\mathbf{A}^{(n)}_{(k_n)}\right)\right)^{-1}\right]_{+}. \quad (5.28)$$

Some extended ALS algorithms were applied the Levenberg-Marquardt approach with the regularization parameter λ

$$\mathbf{A}_{(k_n)}^{(n)} \leftarrow \left[\mathbf{T}\left(\mathbf{S}+\lambda\mathbf{I}\right)^{-1}\right]_{+}. \tag{5.29}$$

However, for real-world data, a proper selection of λ decides the performance of the final result.

5.4.2 Multiplicative Algorithm for Grid NTF

From the gradients (5.21), we apply a similar method to derive the multiplicative LS algorithm for NMF[55;63;67;110;113] to obtain the update rule

$$\mathbf{A}_{(k_n)}^{(n)} \leftarrow \mathbf{A}_{(k_n)}^{(n)} \circledast \left[\mathbf{T}\right]_{+} \oslash \left(\mathbf{A}_{(k_n)}^{(n)}\mathbf{S}\right) \tag{5.30}$$

where matrices $\mathbf{T}$ and $\mathbf{S}$ are defined in (5.26) and (5.27), and computed through a parallel loop in Steps 15 and 17 of Algorithm 5.1. Replacing Step 19 by the expression in (5.30) gives us pseudo-code of the multiplicative algorithm for grid NTF.

The multiplicative algorithms have a relative low complexity but they are characterized by rather slow convergence and they sometimes converge to spurious local minima.

5.4.3 Hierarchical ALS Algorithm for Grid NTF

By taking into account that if the tensor $\mathcal{Y}$ in (5.22) is approximated by a rank-one tensor, that means factors $\mathbf{A}^{(n)}$ have only one component, matrix inverse simplifies into scalar division, and the ALS algorithm (5.22) to update a part k of a factor n is given by

$$\boldsymbol{a}_{(k_n)}^{(n)} \leftarrow \frac{\sum\limits_{\{\boldsymbol{j}|j_n=k_n\}} \mathbf{Y}_{(n)}^{(\boldsymbol{j})} \left\{\boldsymbol{a}_{(k_n)}\right\}^{\odot -n}}{\sum\limits_{\{\boldsymbol{j}|j_n=k_n\}} \left\{\boldsymbol{a}_{(\boldsymbol{j})}^{T}\,\boldsymbol{a}_{(\boldsymbol{j})}\right\}^{\circledast -n}}. \tag{5.31}$$

Therefore, if we formulate an ALS update rule for a component of the sub-factor $\mathbf{A}_{(k_n)}^{(n)}$, this learning rule does not require matrix inverse, and hence reduces computational cost, and is more stable.

Assume that, we estimate the r-th component of the sub-factor $\mathbf{A}_{(k_n)}^{(n)}$ denoted as $\left[\mathbf{A}_{(k_n)}^{(n)}\right]_r = \boldsymbol{a}_r^{(k_n)}$, $n = 1,2,\ldots,N, k_n = 1,2,\ldots,K_n, r = 1,2,\ldots,R$. This component only exists in factorizations of sub-tensors $\mathcal{Y}^{\boldsymbol{k}}$ with k_n be the n-th entry of the sub-tensor index $\boldsymbol{k} = [j_1,\ldots,j_{n-1},k_n,j_{n+1},\ldots,j_N]$, $j_m = 1,\ldots,K_m$. We split the approximation of sub-tensors $\mathcal{Y}^{\boldsymbol{k}}$, $[\boldsymbol{k}]_n = k_n$ into two parts:

- one consists of all rank-one tensors in which the specific component $\boldsymbol{a}_r^{(k_n)}$ is not involved

$$\mathcal{Y}_{(-r)}^{(\boldsymbol{k})} = \sum_{j\neq r} \boldsymbol{a}_j^{(k_1)} \circ \boldsymbol{a}_j^{(k_2)} \circ \cdots \circ \boldsymbol{a}_j^{(k_N)}. \tag{5.32}$$

- And a rank-one tensor built up from $\boldsymbol{a}_r^{(k_n)}$

$$\boldsymbol{\mathcal{Y}}_{(+r)}^{(\boldsymbol{k})} = \boldsymbol{a}_r^{(k_1)} \circ \boldsymbol{a}_r^{(k_2)} \circ \cdots \circ \boldsymbol{a}_r^{(k_N)}. \tag{5.33}$$

The approximation for each sub-tensor $\boldsymbol{\mathcal{Y}}^{(\boldsymbol{k})}$ is now rewritten as

$$\boldsymbol{\mathcal{Y}}^{(\boldsymbol{k})} \;=\; \boldsymbol{\mathcal{Y}}_{-r}^{(\boldsymbol{k})} + \boldsymbol{\mathcal{Y}}_{(+r)}^{(\boldsymbol{k})} + \boldsymbol{\mathcal{E}}. \tag{5.34}$$

To exploit the learning rule (5.31), we define a new residual tensor which is approximated by the rank-one tensor $\boldsymbol{\mathcal{Y}}_{(+r)}^{(\boldsymbol{k})}$

$$\tilde{\boldsymbol{\mathcal{Y}}}_{(+r)}^{(\boldsymbol{k})} \;=\; \boldsymbol{\mathcal{Y}}^{(\boldsymbol{k})} - \boldsymbol{\mathcal{Y}}_{(-r)}^{(\boldsymbol{k})} = \boldsymbol{a}_r^{(k_1)} \circ \boldsymbol{a}_r^{(k_2)} \circ \cdots \circ \boldsymbol{a}_r^{(k_N)} + \boldsymbol{\mathcal{E}}. \tag{5.35}$$

Learning rule to update the component $\boldsymbol{a}_r^{(k_n)}$ is given by

$$\boldsymbol{a}_r^{(k_n)} \leftarrow \frac{\displaystyle\sum_{\{\boldsymbol{j}|j_n=k_n\}} \tilde{\boldsymbol{\mathcal{Y}}}_{(+r)}^{(\boldsymbol{j})} \;\bar{\times}_{-n} \left\{\boldsymbol{a}_r^{(\boldsymbol{j})}\right\}}{\displaystyle\sum_{\{\boldsymbol{j}|j_n=k_n\}} \left\{\boldsymbol{a}_r^{(\boldsymbol{j})T}\, \boldsymbol{a}_r^{(\boldsymbol{j})}\right\}^{\circledast -n}}, \tag{5.36}$$

where $\left\{\boldsymbol{a}_r^{(\boldsymbol{k})}\right\} = \left\{\boldsymbol{a}_r^{(k_1)}, \boldsymbol{a}_r^{(k_2)}, \ldots, \boldsymbol{a}_r^{(k_N)}\right\}$.

Each term in the numerator in (5.36) is rewritten as

$$\begin{aligned}
\tilde{\boldsymbol{\mathcal{Y}}}_{(+r)}^{(\boldsymbol{k})} \;\bar{\times}_{-n} \left\{\boldsymbol{a}_r^{(\boldsymbol{k})}\right\} &= \left(\boldsymbol{\mathcal{Y}}^{(\boldsymbol{k})} - \hat{\boldsymbol{\mathcal{Y}}}^{(\boldsymbol{k})} + \boldsymbol{\mathcal{Y}}_{(+r)}^{(\boldsymbol{k})}\right) \;\bar{\times}_{-n} \left\{\boldsymbol{a}_r^{(\boldsymbol{k})}\right\} \\
&= \left(\mathbf{U}_{(\boldsymbol{k})}^{(n)} \left\{\mathbf{U}_{(\boldsymbol{k})}\right\}^{\odot -nT} - \mathbf{A}_{(k_n)}^{(n)} \left\{\mathbf{A}_{(\boldsymbol{k})}\right\}^{\odot -nT} + \boldsymbol{a}_r^{(k_n)} \left\{\boldsymbol{a}_r^{(\boldsymbol{k})}\right\}^{\odot -nT}\right) \left\{\boldsymbol{a}_r^{(\boldsymbol{k})}\right\}^{\odot -n} \\
&= \mathbf{U}_{(\boldsymbol{k})}^{(n)} \left[\left\{\mathbf{U}_{(\boldsymbol{k})}^{T}\, \mathbf{A}_{(\boldsymbol{k})}\right\}^{\circledast -n}\right]_r - \mathbf{A}_{(k_n)}^{(n)} \left[\left\{\mathbf{A}_{(\boldsymbol{k})}^{T}\, \mathbf{A}_{(\boldsymbol{k})}\right\}^{\circledast -n}\right]_r + \boldsymbol{a}_r^{(k_n)} \left\{\boldsymbol{a}_r^{(\boldsymbol{k})T}\, \boldsymbol{a}_r^{(\boldsymbol{k})}\right\}^{\circledast -n} \\
&= \mathbf{U}_{(\boldsymbol{k})}^{(n)} \left[\mathbf{P}_{(\boldsymbol{k})} \oslash \left(\mathbf{U}_{(\boldsymbol{k})}^{(n)T}\, \mathbf{A}_{(\boldsymbol{k})}^{(n)}\right)\right]_r - \mathbf{A}_{(k_n)}^{(n)} \left[\mathbf{Q}_{(\boldsymbol{k})} \oslash \left(\mathbf{A}_{(k_n)}^{(n)T}\, \mathbf{A}_{(k_n)}^{(n)}\right)\right]_r + \boldsymbol{a}_r^{(k_n)} \left\{\boldsymbol{a}_r^{(\boldsymbol{k})T}\, \boldsymbol{a}_r^{(\boldsymbol{k})}\right\}^{\circledast -n},
\end{aligned}$$

where $[\mathbf{A}]_r = \boldsymbol{a}_r$ denotes the r-th column vector of matrix $\mathbf{A}$. Learning rule (5.36) can be expressed in the compact form

$$\begin{aligned}
\boldsymbol{a}_r^{(k_n)} \leftarrow \boldsymbol{a}_r^{(k_n)} &+ \frac{1}{w_{k_n}} \sum_{\{\boldsymbol{j}|j_n=k_n\}} \mathbf{U}_{(\boldsymbol{j})}^{(n)} \left[\mathbf{P}_{(\boldsymbol{j})} \oslash \left(\mathbf{U}_{(\boldsymbol{j})}^{(n)T} \mathbf{A}_{(\boldsymbol{j})}^{(n)}\right)\right]_r - \frac{1}{w_{k_n}} \sum_{\{\boldsymbol{j}|j_n=k_n\}} \mathbf{A}_{(k_n)}^{(n)} \left[\mathbf{Q}_{(\boldsymbol{j})} \oslash \left(\mathbf{A}_{(k_n)}^{(n)T} \mathbf{A}_{(k_n)}^{(n)}\right)\right]_r \\
&= \boldsymbol{a}_r^{(k_n)} + \frac{1}{w_{k_n}} \boldsymbol{t}_r - \frac{1}{w_{k_n}} \mathbf{A}^{(n)}\, \boldsymbol{s}_r,
\end{aligned} \tag{5.37}$$

where the scalar weight $w_{k_n} = \displaystyle\sum_{\{\boldsymbol{j}|j_n=k_n\}} \left\{\boldsymbol{a}_r^{(\boldsymbol{j})T}\, \boldsymbol{a}_r^{(\boldsymbol{j})}\right\}^{\circledast -n}$, and $\mathbf{T}$ and $\mathbf{S}$ are defined in (5.26) and (5.27), respectively.

Finally, the equation given in (5.37) is the update rule for a component of a subfactor $\mathbf{A}_{(k_n)}^{(n)}$. By replacing Step 19 in Algorithm 5.1 by (5.37), we obtain the pseudo-code for the Hierarchical ALS algorithm for grid NTF.

5.5 Stopping Criterion

Stopping criterion takes an important role in identification of convergence of a factorization. For simplicity, the cost function value (2.1) is usually used as stopping criterion. The FIT rate (FIT(%) = $1 - \frac{\|\mathcal{Y}-\hat{\mathcal{Y}}\|_F^2}{\|\mathcal{Y}\|_F^2}$) can also be used. However, due to similar computation, we only mention the Frobenius norm. For a large tensor, an explicit computation of the cost function value (2.1) is impossible due to so much memory requirement to build up the approximate tensor $\hat{\mathcal{Y}}$. In this section, we derive a fast computation for stopping criterion applied to the grid CP. The Frobenius norm of a raw sub-tensor and its approximation is given by

$$D^{(\boldsymbol{k})} = \|\mathcal{Y}^{(\boldsymbol{k})} - \hat{\mathcal{Y}}^{(\boldsymbol{k})}\|_F^2 = \|\mathcal{Y}^{(\boldsymbol{k})}\|_F^2 + \|\hat{\mathcal{Y}}^{(\boldsymbol{k})}\|_F^2 - 2\langle \mathcal{Y}^{(\boldsymbol{k})}, \hat{\mathcal{Y}}^{(\boldsymbol{k})}\rangle \tag{5.38}$$

where $\langle \mathcal{Y}, \hat{\mathcal{Y}}\rangle$ is the inner product of two same-sized tensors

$$\langle \mathcal{Y}^{(\boldsymbol{k})}, \hat{\mathcal{Y}}^{(\boldsymbol{k})}\rangle \;=\; \sum_{\boldsymbol{i}=\boldsymbol{1}}^{\boldsymbol{I}} y^{(\boldsymbol{k})}_{i_1\cdots i_N}\, \hat{y}^{(\boldsymbol{k})}_{i_1\cdots i_N} = \mathrm{vec}\Big(\mathbf{Y}^{(\boldsymbol{k})}_{(N)}\Big)^T \mathrm{vec}\Big(\widehat{\mathbf{Y}}^{(\boldsymbol{k})}_{(N)}\Big)\,. \tag{5.39}$$

Each terms in the expression (5.38) can be computed as follows

$$\|\hat{\mathcal{Y}}^{(\boldsymbol{k})}\|_F^2 \;=\; \|\,\mathrm{vec}\Big(\widehat{\mathbf{Y}}^{(\boldsymbol{k})}\Big)\,\|_2^2 = \|\{\mathbf{A}_{(\boldsymbol{k})}\}^{\odot}\,\mathbf{1}\|_2^2 = \mathbf{1}^T\,\{\mathbf{A}^T_{(\boldsymbol{k})}\mathbf{A}_{(\boldsymbol{k})}\}^{\circledast}\,\mathbf{1} = \mathbf{1}^T\,\mathbf{Q}_{(\boldsymbol{k})}\,\mathbf{1}, \tag{5.40}$$

$$\langle \mathcal{Y}^{(\boldsymbol{k})}, \hat{\mathcal{Y}}^{(\boldsymbol{k})}\rangle \;=\; \mathbf{1}^T\{\mathbf{U}_{(\boldsymbol{k})}\}^{\odot T}\,\{\mathbf{A}_{(\boldsymbol{k})}\}^{\odot}\,\mathbf{1} = \mathbf{1}^T\Big\{\mathbf{U}^T_{(\boldsymbol{k})}\mathbf{A}_{(\boldsymbol{k})}\Big\}^{\circledast}\,\mathbf{1} = \mathbf{1}^T\,\mathbf{P}_{(\boldsymbol{k})}\,\mathbf{1}\,. \tag{5.41}$$

From (5.38), (5.40) and (5.41), we obtain a convenient and fast computing for the cost function

$$\begin{aligned} D = \frac{1}{2}\sum_{\boldsymbol{k}} D^{(\boldsymbol{k})} &= \frac{1}{2}\sum_{\boldsymbol{k}}\Big(\|\mathcal{Y}^{(\boldsymbol{k})}\|_F^2 + \mathbf{1}^T\,\mathbf{Q}_{(\boldsymbol{k})}\,\mathbf{1} - 2\mathbf{1}^T\,\mathbf{P}_{(\boldsymbol{k})}\,\mathbf{1}\Big) \\ &= \frac{1}{2}\|\mathcal{Y}\|_F^2 + \frac{1}{2}\sum_{\boldsymbol{k}}\Big(\mathbf{1}^T\,\mathbf{Q}_{(\boldsymbol{k})}\,\mathbf{1} - 2\,\mathbf{1}^T\,\mathbf{P}_{(\boldsymbol{k})}\,\mathbf{1}\Big)\,. \end{aligned} \tag{5.42}$$

The first term $\|\mathcal{Y}\|_F^2$ is constant, hence can be neglected. The rest terms are additions of all the entries of matrices $\mathbf{Q}_{(\boldsymbol{k})}$, and $\mathbf{P}_{(\boldsymbol{k})}$.

5.6 Communication Cost and Practical Considerations

This section analyzes the communication cost of grid algorithms, and discusses some practical issues on implementation of the grid factorization. Techniques are introduced to deal with small-subtensors, or with the large number of sub-tensors.

5.6.1 Communication Cost

The communication cost of algorithms for grid CP and grid NTF can be evaluated as the total data transferred between labs and the server in a parallel system during the factorization. For illustration,

we will compute the communication cost for the ALS algorithm. The value can be roughly measured as the total data received and sent in all steps in Algorithm 5.1.

For all the grid algorithms, the communication cost mostly concentrates on computing matrices $\mathbf{T}$ and $\mathbf{S}$ to update the sub-factors $\mathbf{A}_{k_n}^{(n)}$. At Step 14, to update a matrix $\mathbf{P}_{(\boldsymbol{k})}$, a lab requires an $R_{\boldsymbol{k}} \times R$ matrix $\mathbf{P}_{(\boldsymbol{k})}$, the $I_{k_n} \times R_{\boldsymbol{k}}$ sub-factor $\mathbf{U}_{(\boldsymbol{k})}^{(n)}$ and the $I_{k_n} \times R$ sub-factor $\mathbf{A}_{(\boldsymbol{k}_n)}^{(n)}$, then send an $R_{\boldsymbol{k}} \times R$ data to the server. Through Steps 14-17, the total transferred data is $I_{k_n}\,(R_{\boldsymbol{k}} + 3\,R) + 2R_{k_n}\,R + 4\,R^2$ or $4\,I_{k_n}\,R + 6\,R^2$ under the assumption that sub-tensors are approximated with the same number of components of their full tensors, that means $R = R_{\boldsymbol{k}}$. For all the sub-tensors, the total transferred data in the "**parfor**" loop in Step 18 is $\left(4\,R\,\sum_n \frac{I_n}{K_n} + 6\,R^2\right)\,\prod_n K_n$, or $4\,R\,K^{N-1}\,\sum_n I_n + 6\,R^2\,K^N$, with an assumption that $K_1 = K_2 = \ldots = K_N = K$. Hence, the total data transferred inside the "**repeat**" loop (in Step 9) is $6\,R\,K^{N-1}\,\sum_n I_n + 10\,R^2\,K^N$. For a specific data tensor, the communication cost of the ALS algorithm (5.25) increases as the number of sub-tensors K^N increases. The smaller the number of sub-tensors, the lower the communication cost. However, the sub-tensor's size is limited by the configuration of its working lab, such as memory, operating system. For 32-bit system, tensor size should not exceed 3GB. Relatively small sub-tensors can be easily factorized with high fitness. When the number of sub-tensors K_n along the modes-n are exactly the tensor size, that means $K_n = I_n$, for $n = 3, \ldots, N$, the sub-tensors simplify into matrices. Factorizations of these sub-tensors will becomes Singular Value Decompositions which can always explain 100% of the data. However, reconstruction of the full factors demands very high communication cost due to the large number of subtensors.

As a consequence, a large-scale tensor should be divided to have a minimum number of sub-tensors. Dividing the large-scale tensor into small sub-tensors could increase the risk of nearly collinear factors which cannot be approximated by the standard ALS algorithm in (2.2). In order to cope with the problem, we can use some modifications such as Compression[9] and Line Search[25;171]. The compression technique compresses the data into a smaller tensor using the TUCKER model. Next, an CP model is fitted to the core tensor of the TUCKER model. The CP for the original data are represented as products of estimated factors.

5.6.2 Multistage Reconstruction for Grid Decomposition

Reconstruction of the full factors from all sub-factors might have high communication cost of the order $O(K^N)$. This section is devoted to speeding up the reconstruction. A multi-stage reconstruction is recommended for this problem. At each stage, we construct factors for the concatenated tensors which are built up from neighbor subtensors; hence this reduces the number of subtensors for the next step.

The $(N-1)$-stage paradigm illustrated in Figure 5.3(a) is an example. Assuming the tensor is split into a $K_1 \times K_2 \times \cdots \times K_N$ grid of subtensors. At the first stage, we estimate the two factors from

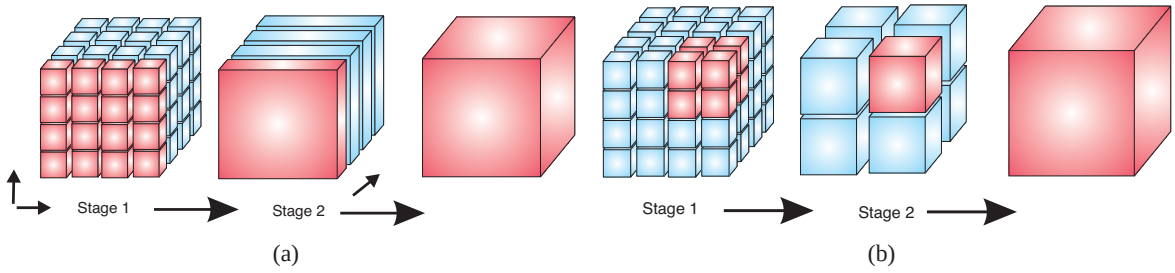


Figure 5.3: Illustration of multistage reconstruction for grid CP.

all sub-factors along the mode-1 and mode-2. That means the grid ALS algorithm will be employed with a grid of $K_1 \times K_2 \times 1 \times \cdots \times 1$. This reconstruction demands a cost of $O(K^2)$, and there are in total $K_3 \times \cdots \times K_N$ separate factorizations processed in parallel. For the second stage, the data can be considered as a concatenation of $1 \times 1 \times K_3 \times \cdots \times K_N$ sub-tensors. We reconstruct the third factor using the grid ALS algorithm with a grid of $1 \times 1 \times K_3 \times 1 \times \cdots \times 1$. The next stage estimates the fourth factor, and so on, until the last mode. This paradigm requires $(N-1)$ stages, and dramatically reduces the communication cost. However, this technique faces a trade-off between accuracy and processing speed. For a 3-D tensor, we need two reconstruction stages as displayed in Figure 5.3(a).

Another multistage paradigm is illustrated in Figure 5.3(b).

5.7 Grid CP for Complex Tensors

CP for complex-valued tensor was proposed and investigated in some real applications such as for DS-CDMA signals[3;139;186], and for MIMO system[214]. This section will extend the proposed algorithms for complex-valued data tensor. The ALS algorithm for complex-valued CP model is extended from the learning rule in (2.2) and given by

$$\mathbf{A}^{(n)} \leftarrow \mathbf{Y}_{(n)} \left\{\mathbf{A}^*\right\}^{\odot -n} \left(\left\{\mathbf{A}^T\mathbf{A}^*\right\}^{\circledast -n}\right)^{-1}, (n = 1, 2, \ldots, N), \tag{5.43}$$

where symbol "*" denotes the complex conjugate operator, and H is the Hermitian transpose. For 3-D complex-valued tensor, this learning rule simplifies into the COMplex parallel FACtor analysis (COMFAC) approach[186].

The ALS algorithm for the grid CP to update a sub-factor $\mathbf{A}^{(n)}_{(k_n)}$ is given by

$$\mathbf{A}^{(n)}_{(k_n)} \leftarrow \mathbf{T}\,\mathbf{S}^{-1}. \tag{5.44}$$

where matrices $\mathbf{T}$ and $\mathbf{S}$ are computed for specific subfactors $\mathbf{A}^{(n)}_{(k_n)}$

$$\mathbf{T} = \sum_{\{\boldsymbol{j}|j_n=k_n\}} \mathbf{U}^{(n)}_{(\boldsymbol{j})} \left(\mathbf{P}_{(\boldsymbol{j})} \oslash \left(\mathbf{U}^{(n)T}_{(\boldsymbol{j})} \mathbf{A}^{(n)*}_{(k_n)} \right) \right), \tag{5.45}$$

$$\mathbf{S} = \left(\sum_{\{\boldsymbol{j}|j_n=k_n\}} \mathbf{Q}_{(\boldsymbol{j})} \right) \oslash \left(\mathbf{A}^{(n)T}_{(k_n)} \mathbf{A}^{(n)*}_{(k_n)} \right), \tag{5.46}$$

based on matrices

$$\mathbf{P}_{(\boldsymbol{k})} = \left\{ \mathbf{U}^{T}_{(\boldsymbol{k})} \mathbf{A}^{*}_{(\boldsymbol{k})} \right\}^{\circledast} \in \mathbb{R}^{R_{\boldsymbol{k}} \times R}, \tag{5.47}$$

$$\mathbf{Q}_{(\boldsymbol{k})} = \left\{ \mathbf{A}^{T}_{(\boldsymbol{k})} \mathbf{A}^{*}_{(\boldsymbol{k})} \right\}^{\circledast} \in \mathbb{R}^{R \times R}. \tag{5.48}$$

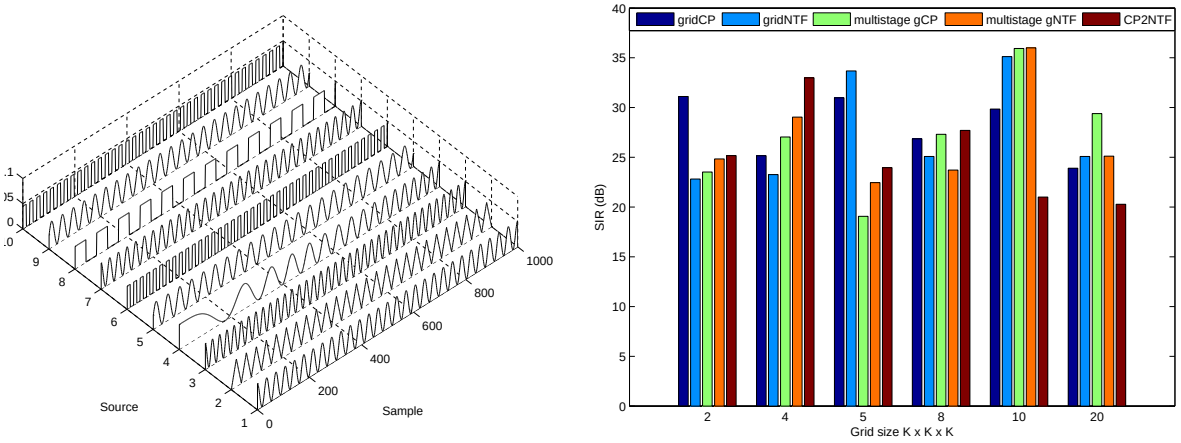
5.8 Experiments

5.8.1 Grid Decomposition with Different Grid Size

In the first set of simulations, we considered a $1000 \times 1000 \times 1000$ dimensional tensor composed from three factors $\mathbf{A}^{(n)} \in \mathbb{R}^{1000 \times 10}$ which were randomly selected from basic sparse and smooth signals such as half-wave rectified sine, cosine, chirp and sawtooth waveforms as illustrated in Figure 5.4(a). This tensor consists of 10^9 entries, and could consume 8GB RAM. We split the tensor into a grid of $K \times K \times K$ sub-tensors, for $K = 2, 4, 5, 8, 10$. That means sub-tensors have 500, 250, 200, 125, 100 entries along each mode, respectively. To evaluate the performance, we employ the Signal-to-Interference Ratio (SIR (dB)) between the estimated and original components. The grid CP and grid NTF achieved almost perfect performance (> 40 dB) as shown in Table 5.1. The results were averaged over 100 runs. The performance almost did not change when varying the grid size.

This tensor was next degraded by an additive Gaussian noise with 10 dB. The grid CP achieved good performance with SIR ≈ 35 dB for different grid size $K = 2, 4, 5, 8, 10$. While the grid NTF achieved SIR ≈ 46 dB. We cannot see the effect of the grid size on the accuracy.

The noise intensities were increased so that SNR = 0 dB. Both the grid CP and grid NTF also obtained good performance with a small grid size $K = 2, 4$. However, the performance decreased as the subtensor's size decreased. The algorithms achieved only 20 dB for the gridsize $K = 20$, or the sub-tensor's size $50 \times 50 \times 50$. Small sub-tensors could not capture the hidden components due to their structures distorted by heavy noise. The comparison of SIR indices for such kind of data are illustrated in Figure 5.4(b). We also analyzed the multistage reconstructions for the grid CP and grid NTF, and the conversion model of CP to NTF factors. Multistage reconstruction achieved slightly lower performance. For example, with a grid size of $K = 5$, the grid CP achieved 29.39 dB, and its multistage factorization achieved 26.85 dB. The grid NTF and its multistage obtained 35.11 dB and



(a) Basic sparse and smooth components

(b) SIR (dB) comparison between models for the 3-D tensor degraded by a 0dB Gaussian noise

Figure 5.4: Illustration of Example 1: (a) components to build up the tensor; (b) comparison of SIR indices obtained by grid CP, grid NTF, multistage grid factorizations, and the conversion model of CP to NTF for different grid sizes. The tensor was degraded by a Gaussian noise with SNR = 0 dB.

32.97 dB, respectively. The performance of the multistage paradigm could be degraded up to 2 dB, but this technique speeded up the processing time. The comparison of processing speed between models is given in Table 5.1. With the same grid size, the multistage reconstructions were always much faster than the grid models. For example, the grid CP approximated the full factors from $512 = 8^3$ sub-tensors in 1131 seconds, and achieved SIR = 25.08 dB, whereas the reconstruction with 3 stages took place in 172 seconds and achieved 23.95 dB.

As analyzed in Section 5.6.1, communication cost of the reconstruction strongly depends on the total number of sub-tensors or the grid size. The larger the number of sub-tensors, the slower the processing speed. The grid CP completed the reconstruction from 4 sub-tensors in 18.65 seconds, and in 249 seconds for 125 subtensors $K = 5$, and in 29,310 seconds for 8,000 sub-tensors $K = 20$. A similar effect of the grid size on the processing speed for the grid NTF, and the multistage grid factorizations.

We replicated this experiment but for a 4-D synthetic tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{1000\times1000\times1000\times1000}$ composed from the similar components. The tensor was also degraded by a Gaussian noise with SNR = 10 dB. The noisy dense tensor consisted of 10^{12} entries, and could consumed 4TB RAM with single precision format. The grid CP and grid NTF factorized the noisy tensor with a $7 \times 7 \times 7 \times 7$ dimensional grid of sub-tensors. That means there were in total 2,401 sub-tensors. Factorizations of all the subtensors took place in 19,156 seconds in a parallel system consisting of 4 nodes and 16 cores. The approximation time will decrease when using more nodes. The reconstructions of factors took 842 seconds using

Table 5.1: Comparison of SIR (dB) indices and running time (second) for grid tensor factorization with different grid size for the synthetic tensor $1000 \times 1000 \times 1000$.

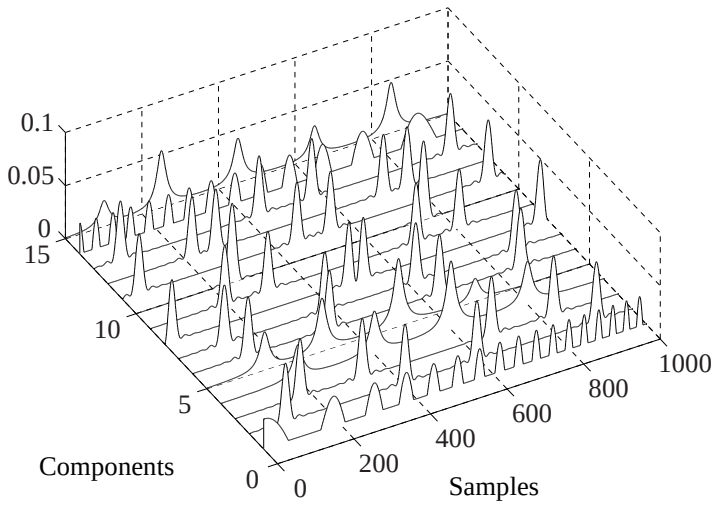
	SIR (dB)						Running time (second)					
	Grid size ($K \times K \times K$)						Grid size ($K \times K \times K$)					
Algorithms	2	4	5	8	10	20	2	4	5	8	10	20
Clean tensors												
gridCP	41.81	**42.54**	42.02	42.53	42.21							
gridNTF	75.30	76.72	**77.77**	77.27	76.60							
10dB Gaussian noises												
gridCP	34.85	35.11	**35.75**	35.42	35.12		24.70	191	337	925	3178	
gridNTF	46.82	46.52	**46.92**	46.35	46.50		46.82	46.52	**46.92**	46.35	46.50	
mgridCP	33.75	32.63	35.72	32.78	34.45		12.53	68.90	98.10	356.30	1395	
0dB Gaussian noises												
Approximation							208	176	536	754	1,335	8,160
gridCP	**31.10**	25.11	29.39	25.08	23.90	21.00	18.65	71.12	249	1131	2,606	29,310
gridNTF	**35.99**	35.92	35.11	29.84	27.70	23.71	18.05	98.07	169	592	1102	8,113
mgridCP	**27.31**	25.08	26.85	23.95	22.44	19.07	14.67	29.39	52.64	172	362	3,437
mgridNTF	**33.67**	30.97	32.97	29.03	27.03	23.25	14.08	33.54	57.69	178	370	3,456
CP2NTF	25.15	**25.16**	24.83	23.52	22.82	20.27	0.27	0.20	0.23	0.23	0.29	0.23

mgridCP multistage grid CP
mgridNTF multistage grid NTF
CP2NTF conversion of CP factors to NTF factors

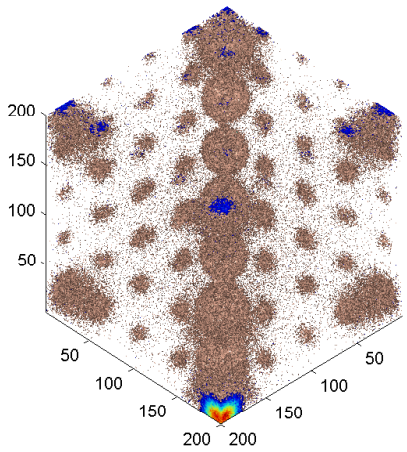
the grid CP, and 3911 seconds using the grid NTF. By using the $(N-2)$-stage paradigm presented in Section 5.6.2, and illustrated in Figure 5.3(b), the grid CP and grid NTF completed the reconstructions in 167 seconds and 165 seconds, respectively. The performance of the experiment is shown in Table 5.2(b). All the algorithms achieved very good performance $>$ 40 dB.

5.8.2 Synthetic Benchmark

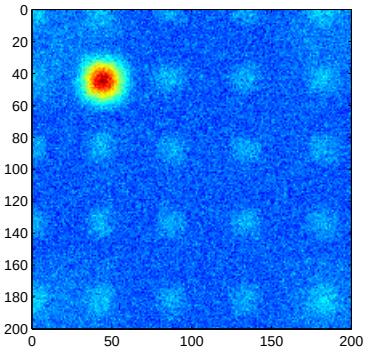
In the next example, we factorized the synthetic tensor $\mathcal{Y} \in \mathbb{R}^{5000\times5000\times5000}$ built up from 15 non-negative components (shown in Figure 5.5(a)), and next degraded by an additive Gaussian noise with SNR = 0 dB. The standard deviation of noise was quickly calculated from synthetic factors based on lemma 1.1 in Chapter 1. A partial data with 200 samples along each dimension is illustrated in Figure 5.5(b), the 45-th slice of this noisy tensor is also given in Figure 5.5(c). This dense tensor with 125 billions of entries could consume 500 GB of memory. Factorizing such tensor using existing CP algorithms is impossible because of large tensor size. However, the proposed algorithm can quickly deal with this problem. In the approximate step, we divided this tensor into 8000 sub-tensors of size $250 \times 250 \times 250$, and simultaneously factorized them with $R_a = 25$ CP components in a parallel system with 16 labs to obtain 8000 sub-factors $\mathbf{U}_{(\boldsymbol{k})}^{(n)} \in \mathbb{R}^{250\times25}$, $n = 1,2,3, \boldsymbol{k} = [k_1,k_2,k_3], k_n = 1,\ldots,20$. The experiment was run on MATLAB ver 2008b and its Distributed Computing Server and Parallel Computing toolboxes.



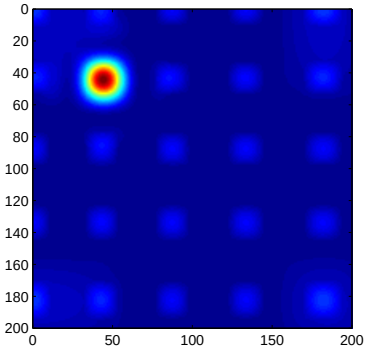
(a) 15 components



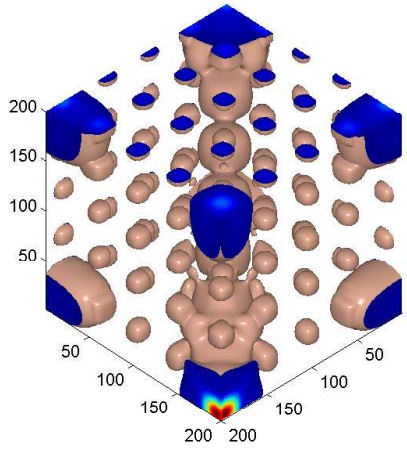
(b) Noisy tensor



(c) Noisy slice (45)



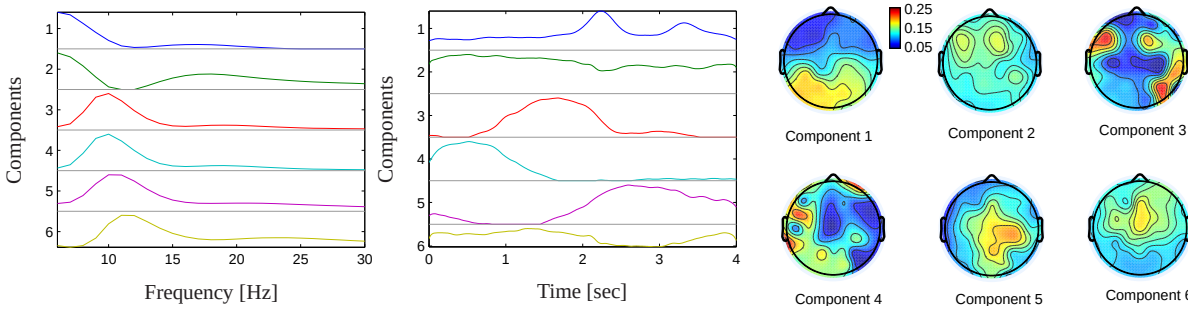
(d) Reconstructed slice (45)



(e) Reconstructed tensor

Figure 5.5: Illustration for Example 2 with a noisy dense tensor of size $5000 \times 5000 \times 5000$ (only 200 samples for each dimensions are shown). (c)-(d) Noisy slice and its reconstruction for Example 2.

The full factors were estimated in two stages to reduce inter-communication between labs. In the first stage, 16 groups of 500 consecutive sub-factors in sub-tensors of size $1250 \times 1250 \times 5000$ were used to simultaneously estimate 16 sets of sub-factors. Then from these sub-factors, we built up the full factors for tensor $5000 \times 5000 \times 5000$. The whole step 2 took 56.51 seconds. With these estimated CP factors, we can quickly retrieve the nonnegative factors under the data by applying the NTF algorithms. This step only took 2.78 seconds. The 15 estimated factors achieved high SIR indices in a range of $[43.64, 54.97]$ dB, and are depicted in Figure 5.5(a). Figure 5.5(d) illustrates the reconstruction of the noisy slice in Figure 5.5(c).



(a) Spectral components (b) Temporal components (c) Topographic maps (spatial components)

Figure 5.6: (a)-(b) Illustration of 6 nonnegative components estimated from 10 CP components for the Graz benchmark. (c) Topographic maps of 6 spatial components for Example 5.8.4 extracted by the grid multiplicative algorithm

5.8.3 Graz EEG Dataset for BCI

The Graz dataset[27] contains EEG signals involving left hand, right hand, foot, tongue imagery movements acquired from 60 channels (with sampling frequency 250 Hz) in a duration of 7 seconds (4 seconds after trigger). The dataset is recorded from 3 subjects, and has 840 trials. All the EEG signals were transformed into the time-frequency domain using the complex Morlet wavelet, to have a spectral tensor 60 channels $\times$ 25 frequency bins (6-30 Hz) $\times$ 250 time frames $\times$ 840 trials. Due to meaningful factorization, the hidden factors under this EEG spectral tensor require nonnegative constraints, and are considered as useful features for successful EEG classification[132]. Therefore, we firstly estimated CP factors of this tensor, then extract nonnegative factors. This dense tensor has a total of 315 millions of entries, and consumes 1.26 GB of memory. Factorization of this full tensor with 10 components took 3900 seconds on a quad core computer (2.67 GHz, 8 GB memory), and achieved FIT = 78.95%. However, the grid ALS algorithm for a grid of 16 sub-tensors divided from the EEG tensor along the 4-th dimension (trials) took only 112 seconds to extract the same number components with FIT = 78.55%. The nonnegative factors quickly derived from both approaches only took 0.41 seconds, and respectively explain 77.08 % and 77.07% of the raw tensor for the full and grid processing. The components of the two spectral and temporal factors are shown in Figure 5.6. In Table. 5.2(a), we compare performances of multiplicative LS, KL, HALS, grid multiplicative LS algorithms.

5.8.4 Visual and Auditory EEG Signals

We illustrate an example with the EEG benchmark `EEG_AV_stimuli`[47] recorded during the 3 stimuli: auditory stimulus; visual stimulus; both the auditory and the visual stimuli simultaneously. EEG signals were recorded from 61 channels during 1.5 seconds after stimulus presentation at a sampling rate of 1 kHz. The observed tensor $\mathcal{Y}$ consists of the EEG spectra in the time-frequency domain using

Table 5.2: Comparison of performance for grid CP models and algorithms.

(a) Grid CP for 4-D noisy tensor in Example 5.8.1.

Model	SIR (dB)	Time (sec)
Subtensor Approx.		19,156
gridCP	75.15	842
gridNTF	87.72	3,911
mgridCP	77.01	167.02
mgridNTF	78.29	164.94
CP2NTF	43.00	0.06

(b) NTF and grid NTF

Algorithms	Example 5.8.3		Example 5.8.4	
	Fit (%)	Time (sec)	Fit (%)	Time (sec)
LS	76.93	20,599	79.40	1,237.4
KL	76.53	29,595	79.30	2,823.7
HALS	76.71	4,820	79.44	417.3
gridLS	77.08	**112.2**	79.37	**68.3**

the complex Morlet wavelet of size 61 channels $\times$ 31 frequency bins ($10-40$ Hz) $\times$ 126 time frames ($0-500$ ms) $\times$ 25 trials $\times$ 3 classes. The multiplicative LS, KL and HALS algorithms[37;47] factorized this tensor with 6 nonnegative components in 1237.4 seconds, 2823.7 seconds and 417.3 seconds with fitness values 79.40%, 79.30% 79.44%, respectively.

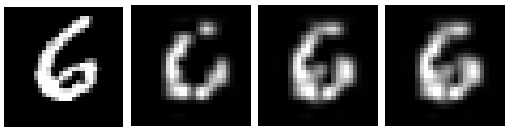
We applied the grid NTF for 75 subtensors of size $61 \times 31 \times 126$ divided from the full data along the 4-th and 5-th dimensions. The approximate step took place in 18 seconds, and the construction step took 50.3 seconds. The running time for the whole factorization was only 68.3 seconds. The full factors explain 79.37% of the original variance. In Figure 5.6(c), we display topographic maps of 6 spatial components. The component 1 relates to the visual stimulus, whereas the components 3 and 4 reflect activations by the auditory stimulus. Comparison of performance of algorithms is given in Table. 5.2(a)

5.8.5 Classification of Handwritten Digits

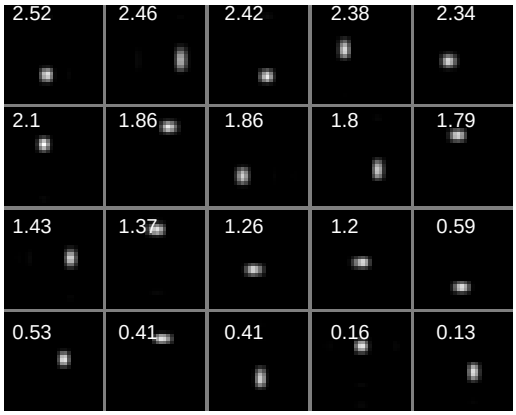
We factorized and classified the MNIST data set of images of handwritten digits (0-9)[112]. This data set is composed 60,000 training images and 10,000 testing images. Each image is a 28×28 grayscale (0-255) labeled representation of an individual digit. Some samples of this dataset are displayed in Figure 7.5(a).

In the training step, we simultaneously factorized 10 sub-tensors corresponding to ten classes (digits) into $R_k = 28$ CP components. This step took place in 185.42 seconds. Then the full nonnegative factors were built from 10 sets of the sub-factors. The two factors $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}$ were used as basis vectors to extract feature of an image $\mathbf{Y}_n$: $\boldsymbol{f}_n = \text{vec}(\mathbf{Y}_n)^T \left(\mathbf{A}^{(2)} \odot \mathbf{A}^{(1)}\right) \left(\left(\mathbf{A}^{(2)T}\mathbf{A}^{(2)}\right) \circledast \left(\left(\mathbf{A}^{(1)T}\mathbf{A}^{(1)}\right)\right)\right)^{-1}$. The number of features is exactly the number of NTF components R.

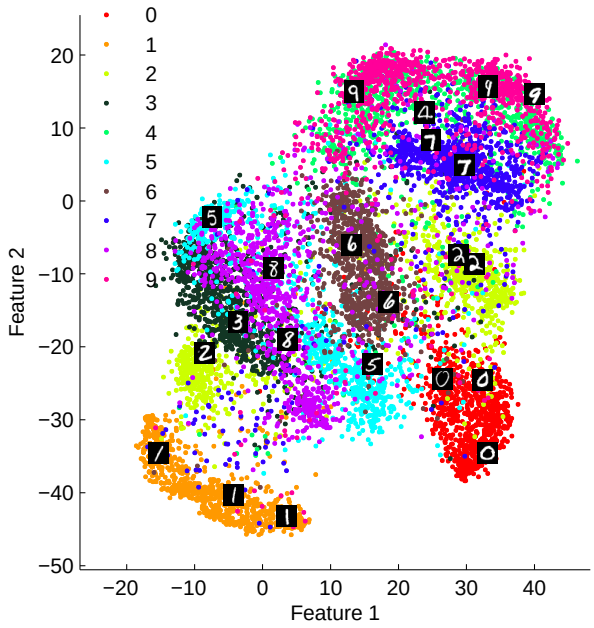
An example for the 36027-th sample (digit six) in the training database is illustrated in Figure 5.7. Training feature of this digit is a vectors which consists of R entries. A pair of basis vectors $\boldsymbol{a}_j^{(1)}, \boldsymbol{a}_j^{(2)}$ forms a basis image $\boldsymbol{a}_j^{(1)} \boldsymbol{a}_j^{(2)T}$. A reconstructed sample of a digit was a linear composition of basis



(a) Digit 6 and reconstructions by 10, 15, 20 features.



(b) Selected basis images to reconstruct the samples ranked in the descending order.



(c) t-SNE visualization for CP features.

Figure 5.7: Feature extraction for digits using CP features: (a) a training digit and its reconstructions by using 10, 15 and 20 features with corresponding bases in (b). (c) Visualization of CP features using the t-SNE components

images in which scaling coefficients were features. In Figure 5.7(b), we displayed basis images which were ranked in the descending order of 20 largest features. Feature coefficients were annotated in their corresponding bases. Superimposes of 10, 15, 20 basis images resulted reconstruction images of this digit shown in Figure 5.7(a), respectively.

We used the k-nearest neighbor classifier (KNN with$k = 3$) for this experiment, and verified the classification accuracy with 10 different numbers of features: $R = 20, 21, \ldots, 28$. Both two classifications based on gNTF and NTF returned quite similar accuracy given in Table 5.3. However, the gNTF-based method significantly reduces the running time. Especially, this method provides a convenient and fast way for Monte Carlo analysis. To build 9 sets of nonnegative factors in this experiment, the total running time for gNTF is $185.42 + 232.59 = 418.01$ seconds, whereas that for NTF is 3340.68 seconds.

From the extracted features, the dataset is visualized via two t-SNE components[207] in Figure 5.7(c). Classification of this benchmark can be performed using the Tucker decomposition[180].

Table 5.3: Comparison of grid NTF and NTF for the digit dataset.

No. Features	Running Time (seconds)		Accuracy (%)	
	gNTF	NTF	gNTF	NTF
20	17.38	394.31	95.60	95.86
21	25.26	400.67	96.12	96.00
22	26.28	313.37	96.02	96.40
23	19.71	357.08	96.43	96.18
24	24.91	479.83	96.24	96.21
25	38.96	520.47	96.51	96.56
26	24.03	271.90	96.74	96.78
27	29.95	349.11	96.73	96.84
28	26.11	253.94	96.95	96.62
Total	418.01	3340.68		

5.9 Summary

We presented the new fast and robust ALS algorithms for large-scale CP. The validity and high performance of the proposed algorithm have been confirmed even for noisy data, and also for the large-scale BCI benchmark, and image classification. The new fast stopping criterion is proposed for this algorithm. Variations of the ALS algorithm for CP with regularized terms such as total variation, sparsity, smoothness, nonnegativity, orthogonality constraints can be applied to the grid CP with some modifications on the learning rule (5.25). Strategy for grid division of a tensor can affect to the performance of factorization, and the running time of parallel computing. Basically, sub-tensors' sizes should satisfy unique conditions of CP[106]. Moreover, sub-tensor should have maximum possible number of entries in its working lab. Increasing the number of sub-tensors from the original tensors increases the communication cost of the algorithms. To deal with this, we can estimate the full factors in multistage as illustrated in Examples 5.8.1, 5.8.2.

CHAPTER 6

Simulations and Results for Algorithms for CP and Tucker Decompositions

The proposed algorithms have be verified and compared with existing algorithms in a variety of benchmarks including real-world data or randomly generated tensors with dense or sparse factors, or factors with bias, factors with collinear components. For CP, our algorithms including HALS (Algorithm 2.1), (fast) dGN or LM (Algorithm 4.1)[167] are compared with ALS, OPT[4]. For NTF, we compared HALS (Algorithm 2.1)[37;152], QALS (Algorithms 3.2, 3.3)[160;166], rK-QALS (Algorithm 3.6)[161], LM$_+$ (4.58)[169] with ALS and multiplicative algorithms. The multiplicative KL (mKL) and LS (mLS) algorithms[132] are based on gradient descent approach applied to the Kullback-Leiber divergence, the Frobenius distance, respectively. For (nonnegative) Tucker decomposition, the algorithms HALS (Algorithm 2.2), LM (Section 4.5)[168], QALS (Algorithm 3.5) are verified. For sparse tensors, the HALS algorithm with regularization term controlling sparseness was tested. Algorithms used in this chapter are listed in Table 6.1.

All the algorithms were initialized using the same method. For low rank approximations, that is leading singular vectors of $\langle \mathcal{Y}, \mathcal{Y} \rangle_{-n}$ which can be computed as in the HOSVD algorithm[58]. For large-scale tensor and relatively high R, random initialization is employed.

Stopping criteria are based on differences of successive relative errors which is lower than 10^{-10}, that is

$$\varepsilon = \frac{\|\mathcal{Y} - \hat{\mathcal{Y}}\|_F}{\|\mathcal{Y}\|_F} < 10^{-10}, \tag{6.1}$$

or when the maximum number of iterations (e.g. 2000) is exceeded.

In order to evaluate the estimation accuracy, the SIR index was calculated for the true and estimated components after permutation matching and normalization

$$SIR = -20 \log_{10} \frac{\|\boldsymbol{a}_r^{(n)} - \hat{\boldsymbol{a}}_r^{(n)}\|_2}{\|\boldsymbol{a}_r^{(n)}\|_2} \text{ dB}. \tag{6.2}$$

In addition, we compute Mean Square Angular Error (MSAE) between orginal and estimated components $\boldsymbol{a}_r^{(n)}, \widehat{\boldsymbol{a}}_r^{(n)}$ after matching order of components defined in Section 6.1.1[108;197;198].

Table 6.1: List of tested algorithms for tensor decompositions and sections in which they are introduced.

Algorithm	CP	NTF	Tucker	NTD
ALS	✓ 2.1	✓ 2.1		
HALS	✓ 2.3	✓ 2.3	✓ 2.8	✓ 2.8
QALS		✓ 3.2		✓ 3.3
rK-QALS		✓3.5		✓ 3.5
LM or dGN	✓ 4.2	✓ 4.3	✓ 4.5	✓ 4.5
mLS[47;131]		✓		✓
mKL[47;131]		✓		✓
OPT[4]	✓			
HOSVD,HOOI[58;60]			✓	
gridCP	✓ 5.3	✓ 5.4		

6.1 Simulations for CP

The dGN algorithm has been successfully confirmed for difficult data (such as collinearity of factors, different magnitudes of factors) by Paatero[143]. Later, the dGN has been validated again by Tomasi[199;200;201;203]. A variation of the dGN algorithm for three way data is the INDAFAC algorithm[199] which computes the approximate Hessian, and gradient, and employs Cholesky decomposition to deal with inverse problems $\mathbf{H}^{-1}$. Therefore, it is straightforward to see that INDAFAC demands much higher computational cost than that of our fast dGN algorithm for three-way data. Moreover, with the same initial values and damping parameters, the dGN and fast dGN algorithms should return similar results. The major difference between the fast dGN algorithm and common dGN algorithms is complexity. However, this task is equivalent to determining how much inverse of an $\left(NR^2 \times NR^2\right)$ matrix in our fast algorithm is faster than inverse of an $\left(R\sum_n I_n \times R\sum_n I_n\right)$ matrix ($R \ll I_n$) in other dGN algorithms. Therefore, we do not intend to compare running time between dGN algorithms.

In this section, we analyze the CP algorithms for difficult data with collinear factors in all modes (swamp). Collinearity degree of factors is controlled by mutual angles between components. Orthonormal factors $\mathbf{U}^{(n)}$ were first randomly generated from the normal distribution, then converted to collinear factors $\mathbf{A}^{(n)}$ by a simple modification

$$\boldsymbol{a}_r^{(n)} = \boldsymbol{u}_1^{(n)} + \nu\,\boldsymbol{u}_r^{(n)}, \qquad \nu \in (0,1], \forall n, \forall r \neq 1\,. \tag{6.3}$$

Mutual angles $\theta_{q,r}$ between components $\boldsymbol{a}_q^{(n)}$ and $\boldsymbol{a}_r^{(n)}$, $q \neq r$ are in a range of $(0, 60^o]$ for $\nu \in (0,1]$, and have

$$\tan(\theta_{q,r}) = \begin{cases} \nu, & q = 1, \\ \nu\sqrt{\nu^2+2}, & q \neq 1, r. \end{cases} \tag{6.4}$$

For example, $\nu = 0.1, 0.2, \ldots, 1$ yield $\theta_{1,r} = 6^o, 11^o, 17^o, 22^o, 27^o, 31^o, 35^o, 39^o, 42^o, 45^o$, and $\theta_{q,r} = 8^o, 16^o, 23^o, 30^o, 37^o, 43^o, 48^o, 52^o, 56^o, 60^o$, $q \neq 1, q \neq r$, respectively. For high ν such as $\nu = 2$, $\theta_{1,r} \approx 63^o$ and $\theta_{q,r} \approx 78^o$, and tensor can be quickly factorized by CP algorithms. The higher the parameter ν, the lower the collinearity of factors. It is more difficult to factorize tensors with lower ν (e.g. $\nu = 0.1, 0.2$). However, $\nu > 3$ arises another issue involving a large difference in magnitude between components. The tensors are still difficult to factorize even thought collinearity of factors is low ($\theta_{1,r} > 71^o$). In addition to (1.21), CP tensors in our simulations can also be expressed as

$$\mathcal{Y} = \sum_{r=1}^{R} \lambda_r \, \boldsymbol{a}_r^{(1)} \circ \boldsymbol{a}_r^{(2)} \circ \cdots \circ \boldsymbol{a}_r^{(N)}, \tag{6.5}$$

where $\|\boldsymbol{a}_r^{(n)}\|_2 = 1, \forall r$, and $\lambda_1 = 1$, and $\lambda_r = (1+\nu^2)^{N/2}, \forall r > 1$. Therefore, for $\nu = 3, 4, 5$ and $N = 3$, $\lambda_r = 31.6, 70.1, 132.6, \forall r \neq 1$, respectively. That means factors have $(R-1)$ large components compared with the first component.

The OPT algorithm has been tested for collinear data with the same mutual angles $\theta_{q,r} = 26^o$ and 60^o, $\forall q \neq r$ in[4]. In our experiments, $\theta_{1,r}$ and $\theta_{q,r}$, $q > 1$ are different, $\theta_{1,r} < \theta_{q,r}$, and are in a wider range $[6^o, 60^o]$. We analyze synthetic tensors for two cases: error-free and noisy data with additive white Gaussian noise at SNR = 20 dB, 30 dB or 40 dB.

The proposed algorithms have been extensively verified and compared with the ALS, ALS plus line seach (LS), and OPT algorithms for large-scale, high dimensional tensors with/without Gaussian noises. All the algorithms are analyzed under the same experimental conditions. Those are leading singular values for initialization, iteration until 10 differences of successive relative errors $\varepsilon = \dfrac{\|\mathcal{Y} - \hat{\mathcal{Y}}\|_F}{\|\mathcal{Y}\|_F}$ are lower than 10^{-12}, or the maximum number of iterations (5000) is exceeded. The ALS algorithm is adapted from the Tensor toolbox[16;17] to accompany with line search and enhanced line search. The line search algorithm was adapted from[25;200] to run with synthetic and high dimensional tensors. For ELS[171], although its published version[50] can support arbitrary dimension, this algorithm is in practice computational demanding. Moreover, results of ELS in our simulations are not significantly different from those of LS. Therefore, we skip ELS in the simulation.

The OPT algorithm[4] provided in the Tensor toolbox[17] was set to run with Hestenes-Stiefel (HS) updates, and adapted to factorize noiseless large-scale tensors and to have the same stopping criteria as other algorithms. Parameter "MaxFuncEvals" was set to $+\infty$, while other parameters of OPT were set to their default values. The fast dGN algorithm with its two variations fLM_a and fLM_b in Section 4.2.3.4 is denoted by fLM. We note that the two variations are equivalent in the sense of performance.

In order to roughly compare complexity of algorithms, we provide the average execution time per iteration for algorithms in addition to their number of iterations. Running time does not reflect appropriate complexity of an algorithm. It mostly depends on programming skills (memory management,

code optimization), and programming language. All algorithms are implemented in Matlab which is relatively slow for doing large number of "for" loops. Moreover, due to huge workload for the whole simulations, we run experiments in parallel system in which workers do not have the same configuration (CPU, memory). We note that speed of an algorithm is proportional to its computational cost for one iteration and the number of iterations in one run. We denote a task in which all factors have been updated once as one iteration. The cost of ALS per one iteration is approximately $O(NI^{N-1}R + NR^3)$ for a tensor with $I = I_1 = \ldots = I_N$. OPT has a cost of $O(NI^{N-1}R)$ per gradient evaluation. However, its cost per one iteration is much higher due to repetition of gradient evaluation for optimization of step size. The fast dGN (fLM) algorithm has a cost of $O(NI^{N-1}R + N^3R^6)$.

6.1.1 Mean Squares Angular Errors and Cramér-Rao Induced Bound

In order to evaluate performance of factorizations, in additional to the relative error, we compute Mean Square Angular Error (MSAE) between orginal and estimated components $\boldsymbol{a}_r^{(n)}, \widehat{\boldsymbol{a}}_r^{(n)}$ after matching order of components defined as[108;197;198]

$$\text{MSAE}(\boldsymbol{a}_r^{(n)}, \widehat{\boldsymbol{a}}_r^{(n)}) = E\left[\arccos^2 \frac{\boldsymbol{a}_r^{(n)H}\widehat{\boldsymbol{a}}_r^{(n)}}{\|\boldsymbol{a}_r^{(n)}\|_2\|\widehat{\boldsymbol{a}}_r^{(n)}\|_2}\right]. \tag{6.6}$$

Cramér-Rao Induced Bound (CRIB) on the MSAE of $\widehat{\boldsymbol{a}}_r^{(n)}$ is computed as in[108;197]. The average MSAEs for all the estimated components are compared against the average CRIB. Figure 6.1(a) illustrates the angular CRIB versus ν given in (6.3) at SNR = 30 dB for various sizes R and dimensions N. Legend describes factor sizes $I_n \times R$ and dimension of tensors N. CRIB for the same tensors at other SNR can be straightforwardly deduced from those in Figure 6.1(a). For example, CRIB at SNR = 20 dB or 40 dB is shifted up or down 10 dB from that at SNR = 30 dB. CRIB for the test case of $N = 3$, $I_n = 50$, $R = 20$ and at SNR = 20 dB deduced from that at SNR = 30 dB (dark-yellow line) is above -30 dB. We also have CRIB for the test case of $N = 3$, $I_n = 50$, $R = 5$ and SNR = 40 dB deduced from that at SNR = 30 dB (pink line) is lower than -30 dB. It is important to note that an MSAE lower than -30 dB, -26 dB or -20 dB means two components are different by an mutual angle less than 2^o, 3^o and 6^o, respectively. This gives us a rough evaluation of a rather good approximation in which MSAE < -30 dB. As seen in Figure 6.1(a), the angular CRIB for the test case of $I_n = 50$, $R = 5$, $N = 3$ and SNR = 30 dB is lower than -30 dB for $\nu = 0.2, 0.3, \ldots, 1$. That means we can obtain a good approximate in which mutual angles of components are less than 2^o. Practical simulations show that MSAE is hard to reach a CRIB $\geqslant$ -30 dB, since collinearity of factors has been destroyed by noise. Discussion on effects of noise on collinear data in Appendix A6.1 gives us insight into when CP algorithms are not stable, and when they succeed in retrieving collinear factors from noisy tensors. Figure 6.1(a) also reveals that we cannot retrieve collinearity components from factorization of 3-D

tensors of rank $R = 15, 20$ at SNR = 20 dB for $\nu < 0.5$. Therefore, we will not analyze simulations having CRIB $\geqslant$ -26 dB.

6.1.2 Factorization of Real-Valued Highly Collinear Tensors

We analyze tensors of size $I_n = 50$, various dimensions $N = 3, 4, 6$ and ranks $R = 5, 10, 15, 20$, and at different $\nu = 0.1, 0.2, \ldots, 1$. MSAE is computed from 100 runs for each combination.

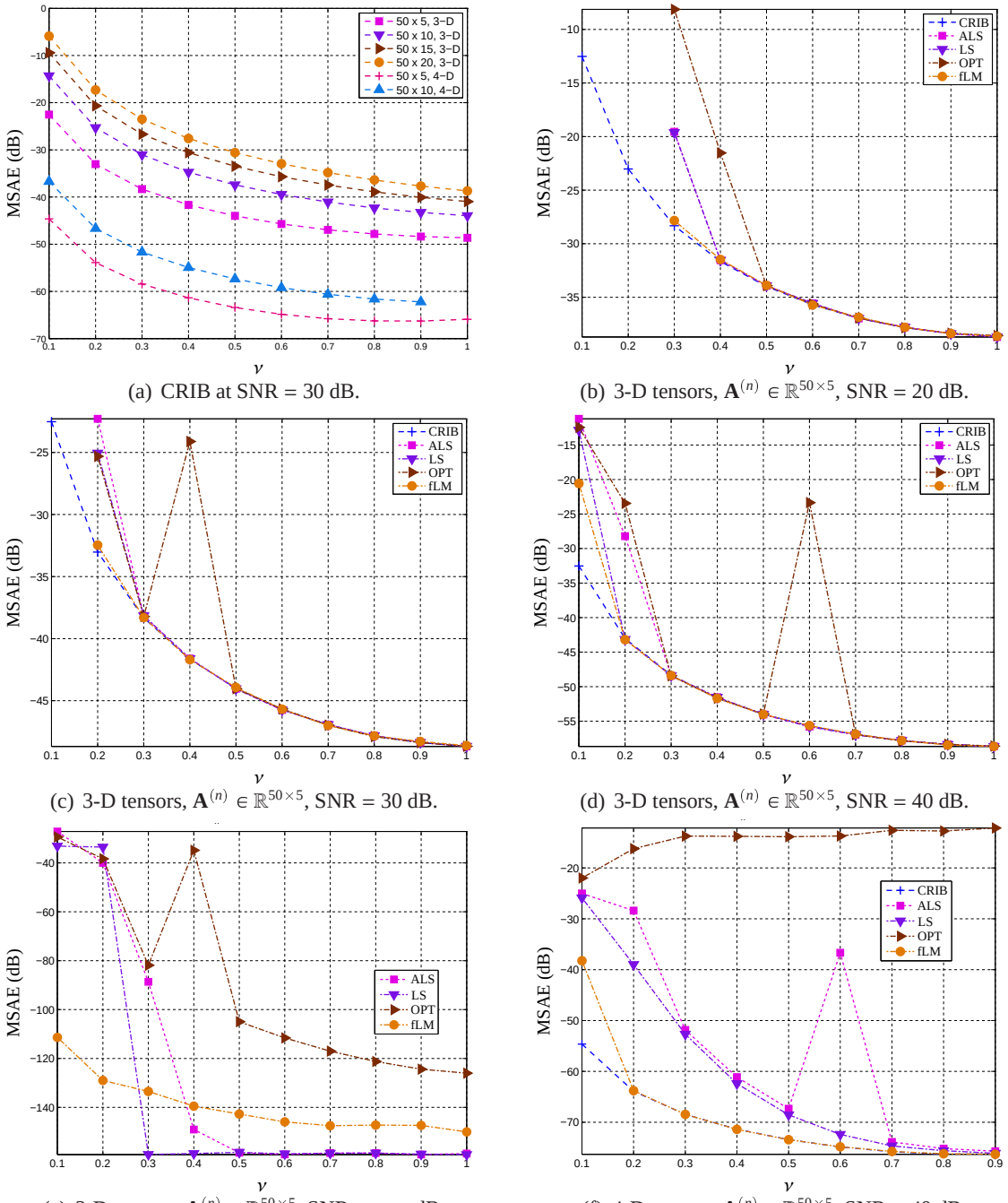
For noiseless tensor factorizations, the average MSAEs of algorithms are compared in Figures 6.1(e), 6.2(e), 6.3, 6.4(a), (c). For 3-D tensors of rank $R = 5$ (in Figure 6.1(e)), ALS, LS and OPT obtained MSAE = -27 dB, -33 dB and -29 dB for high collinearity $\nu = 0.1$, respectively. That means original and approximate components are different by an angle $\approx 2^o$. The algorithms achieved better MSAE for $\nu > 0.5$. We note that CP algorithms are expected to retrieve the exact collinear factors from noiseless tensors. This can be obtained only by dGN (fLM) with perfect MSAE $<$ -100 dB, $\forall \nu$.

LS improved performance of ALS for 3-D noiseless tensors as illustrated in Figures 6.2(e), 6.4(a), (c), and for 6-D noiseless tensors in Figures 6.3. However, the improvement is not sufficient for experiments with $\nu = 0.1$. OPT might still get stuck in local minima, or its updates or chosen parameters were not suitable for the simulations. For some runs, OPT explained the data tensors and achieved low MSAE. However, its performance for the whole analysis reflects the algorithm is not stable in our experiments.

MSAE for factorizations of noisy tensors at SNR = 20 dB, 30 dB and 40 dB are illustrated in Figures 6.1-6.3. CRIB at SNR = 20 dB and 40 dB are shifted $\pm$ 10 dB from that at SNR = 30 dB plotted in Figure 6.1(a).

For 3-D tensors of rank $R = 5$ at SNR = 20 dB, MSAEs of ALS and LS reach CRIB for $\nu \geqslant 0.4$ as illustrated in Figure 6.1(b), whereas OPT's MSAEs approach CRIB for $\nu \geqslant 0.5$. MSAEs of fLM are always close to CRIB even for $\nu = 0.3$. At SNR = 30 dB, algorithms mostly converged to solution for $\nu \geqslant 0.2$. However, ALS, LS and OPT failed to retrieve collinear factors in some runs as $\nu = 0.2$. Hence, their average MSAEs are still far from CRIB (see in Figure 6.1(c)). At SNR = 40 dB, CP algorithms obtained high MSAE comparable to CRIB for most ν. MSAEs of LS are better than those of ALS and OPT, and are identical to those of fLM for $\nu \geqslant 0.2$ as shown in Figure 6.1(d). We note that CP algorithms are not stable for $\nu = 0.1$ in spite of CRIB = -33 dB. Explanation for this test case is discussed in Appendix A6.1 and illustrated in Figure 6.19(b). For this test case, condition number of the approximate Hessian with respect to solutions is 1 due to large damping parameter $\nu > 10^{30}$.

In Figure 6.2, we compare algorithms for higher rank $R = 10$. It should be noted that factorization becomes more difficult for higher rank R. The angular CRIB for $R = 10$ is lower than that for $R = 5$ as seen in Figure 6.1(a). fLM's MSAEs always approach CRIB at various SNR levels in Figures 6.2(a)-



(a) CRIB at SNR = 30 dB.

(b) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 5}$, SNR = 20 dB.

(c) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 5}$, SNR = 30 dB.

(d) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 5}$, SNR = 40 dB.

(e) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 5}$, SNR = $+\infty$ dB.

(f) 4-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 5}$, SNR = 40 dB.

Figure 6.1: (a) angular CRIB at SNR = 30 dB for various sizes and ranks. Legend describes factor sizes $I_n \times R$ and dimensions of tensors N. (b)-(e) the average MSAE for factorization of 3-D tensors with $I_n = 50$, rank $R = 5$ at SNR = 20 dB, 30 dB, 40 dB and $+\infty$ dB (noiseless tensor). Algorithms run until reaching a derivative of successive relative errors of 10^{-12} or 5000 iterations. (f) MSAE for factorization of 4-D tensors with $I_n = 50$, $R = 5$ at SNR = 40 dB evaluated as algorithms reach a derivative of successive relative error of 10^{-8} or 1000 iterations.

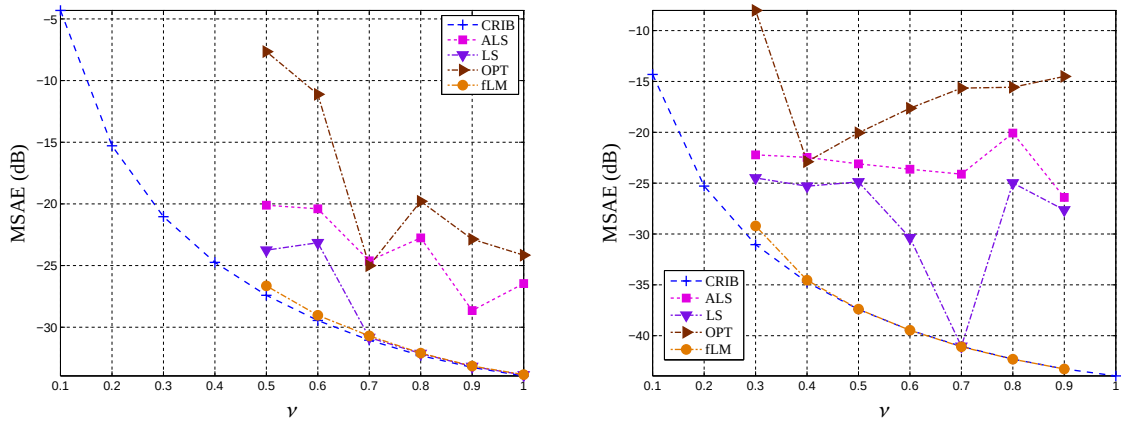
(c). LS seems more stable than ALS and OPT, and sometimes its performances (MSAE) reach CRIB, e.g., $\nu = 0.7, 0.8, 0.9, 1$ at SNR = 20 dB (Figure 6.2(a)), $\nu = 0.8, 0.9, 1$ at SNR = 40 dB (Figure 6.2(c)). Both ALS and OPT are not stable, or have not yet converged under two stopping criteria. In addition to the average MSAE in Figure 6.2(c), Figure 6.2(d) illustrates median MSAE of algorithms which are comparable to CRIB. That means ALS, LS and OPT sometimes achieved good MSAE by 5000 iterations. However, in some runs, they have not yet converged to the solution, and might need more iterations. Figures 6.2(c)-(d) also indicate that ALS, LS and OPT are less stable than fLM. We note that the performance of algorithms might depend on the stopping criteria, and they all should be similar if we increase the number of iterations and discard all other stopping criteria (such as difference of relative errors).

MSAEs of 3-D tensors of the same size but higher ranks R = 15 and 20 at SNR = 40 dB are illustrated in Figures 6.4(b), (d). The higher the tensor rank R, the more difficult the approximation. As seen in Figure 6.1(a) and also in Figures 6.4(b), (d), CRIB increases (worse performance) as increasing rank R. Therefore, for $R = 20$, we cannot obtain accurately collinear components for $\nu = 0.1, 0.2$ despite high SNR = 40 dB. Once again, in this analysis, fLM gives the best performance (MSAE) which is comparable to CRIB.

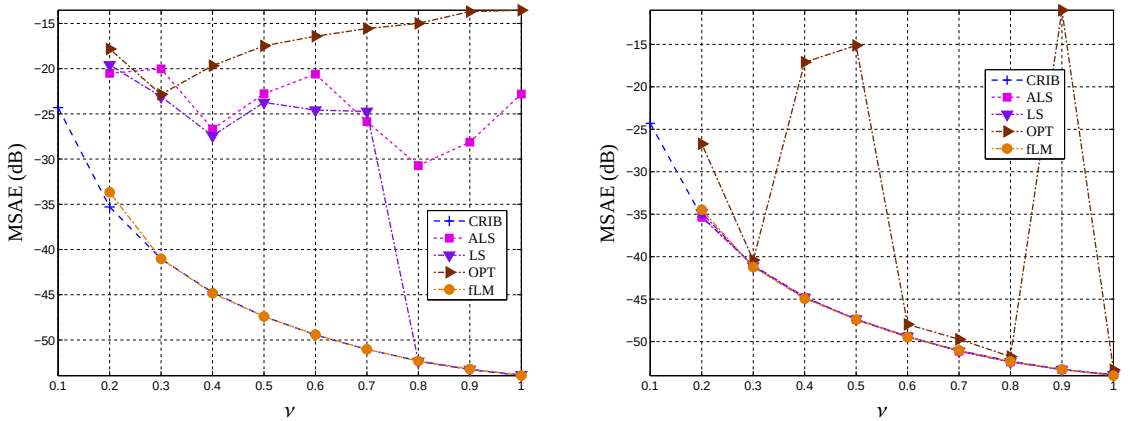
Additional results shown in Figures 6.1(f), 6.2(f) are for 4-D tensors with rank R = 5 and 10 at SNR = 40 dB. For these simulations, algorithms run until differences of successive relative errors are lower than 10^{-8}, or the number of iterations exceeds 1000. MSAE of fLM validates that fLM is superior to other algorithms in our simulations.

In order to compare complexity of algorithms, Figure 6.6 shows the average execution time (milliseconds) per iteration of algorithms for factorization of 3-D tensors with R = 5, 10, 15, 20. The running time was measured in Matlab on two separate computers. The first one (PC1) is a PC that has a Core 2 Duo 2.4 GHz processor, and 2GB memory. The second one is a computing server (PC2) that has 2 quadcore 3.33 GHz processors and 64 GB memory. The running time per iteration for ALS is less than those of other algorithms on PC1. However, OPT's complexity seems less (on average) than that of ALS for high rank R = 15, 20 on the computing server. For fLM, the running time per iteration are often higher than those for ALS and OPT, and increases as ν increases. For example, in order to to factorize 3-D tensors of rank $R = 20$ for $\nu = 0.2$, 1, fLM consumes 406 msecs and 583 msecs (per iteration) on PC1 respectively, whereas it take 125 msecs and 168 msecs on PC2.

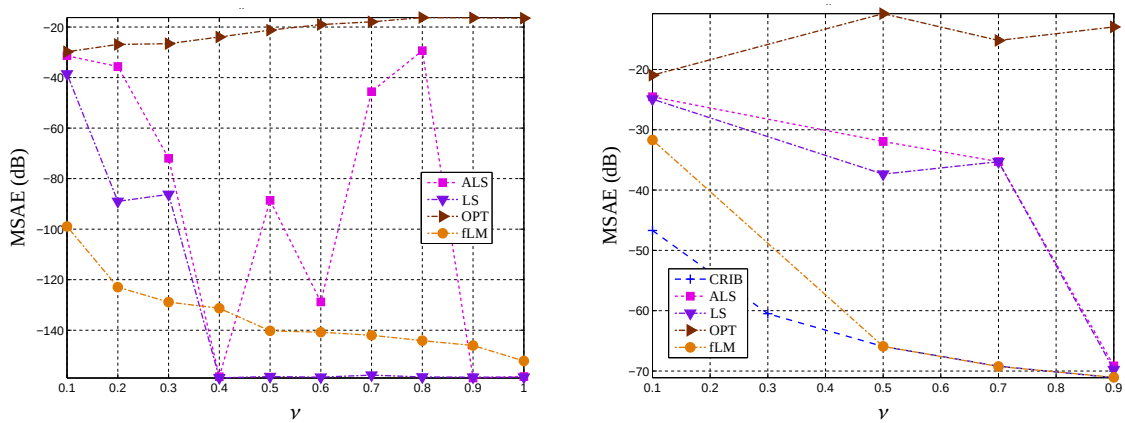
Running time of an algorithm over the whole factorization is proportional to its number of iterations and execution time/iteration. In addition to Figure 6.6, we illustrate the number of iterations of algorithms in Figures 6.5, 6.7. For 3-D tensors $I_n = 50$ and $R = 5$ and $\nu = 0.2$ at SNR = 40 dB, ALS, LS, OPT consumed 99 secs, 185 secs, 151 secs (on average) to finish on PC2, while fLM consumed only 3 secs on the same computer. For 3-D tensors of the same I_n, $\nu = 0.2$, SNR = 40 dB but $R = 10$,



(a) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 10}$, SNR = 20 dB. (b) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 10}$, SNR = 30 dB.



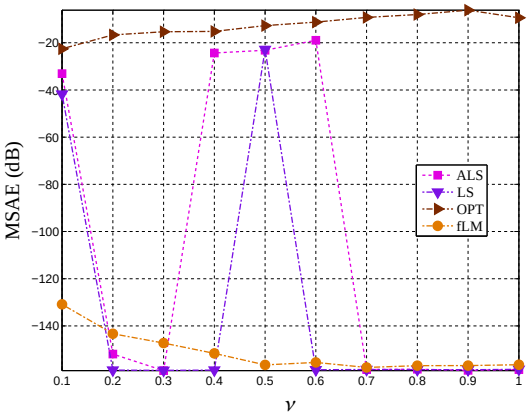
(c) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 10}$, SNR = 40 dB. (d) median MSAE, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 10}$, SNR = 40 dB.



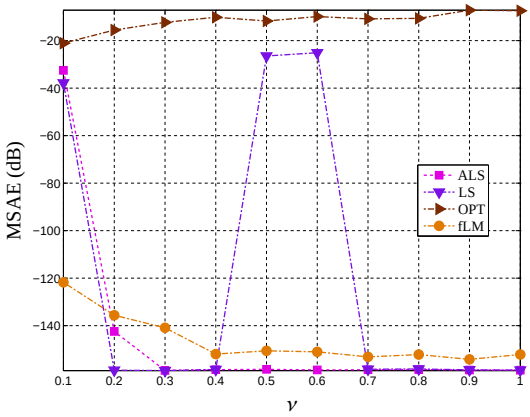
(e) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 10}$, SNR = $+\infty$ dB. (f) 4-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 10}$, SNR = 40 dB.

Figure 6.2: (a)-(e) Average or median MSAE for factorization of $50 \times 50 \times 50$ dimensional tensors with rank $R = 10$ at SNR = 20 dB, 30 dB, 40 dB and $+\infty$ dB (noiseless tensor). (f) MSAE for factorization of 4-D tensors with the same factor size at SNR = 40 dB.

ALS, LS and OPT took 143 secs, 258 secs and 182 secs, respectively, while fLM took only 76 secs on PC2. We can evaluate speed ratio between fLM and ALS for different ν. For 3-D tensors of rank $R = 5$ at SNR = 40 dB, the speed ratios for $\nu = 0.2, 04, 0.6, 0.8, 1$ are approximately 15.3, 9.6, 4.4, 2.6,

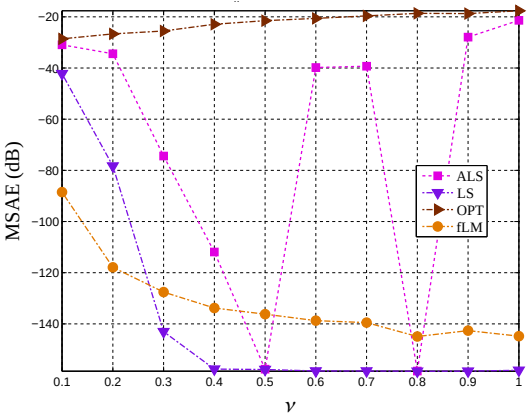


(a) 6-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 5}$, SNR = $+\infty$ dB.

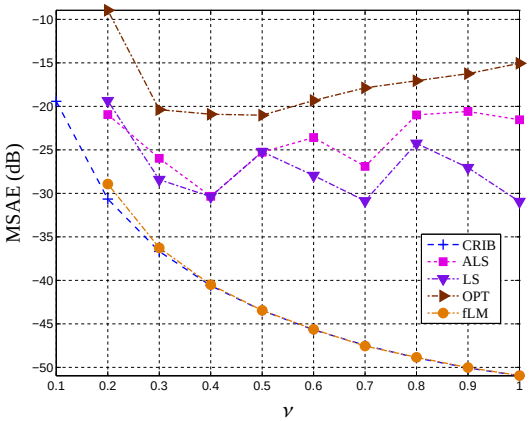


(b) 6-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 10}$, SNR = $+\infty$ dB.

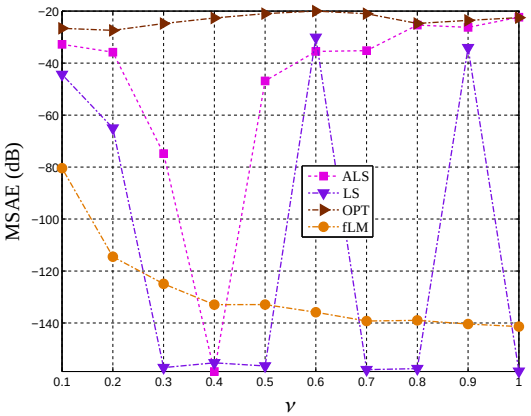
Figure 6.3: Average MSAE for factorization of 6-D noiseless tensors with size $I_n = 50$. Algorithms run until reaching a derivative of successive relative errors of 10^{-12} or 5000 iterations.



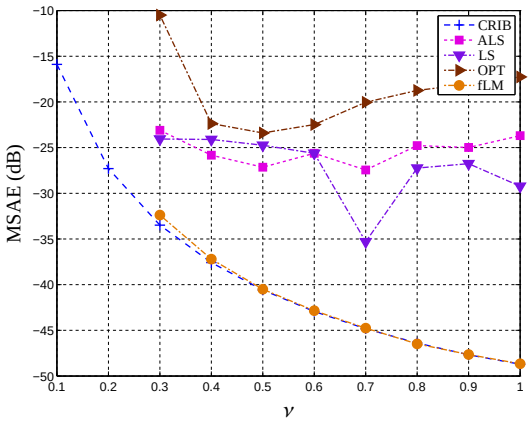
(a) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 15}$, SNR = $+\infty$ dB.



(b) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 15}$, SNR = 40 dB.

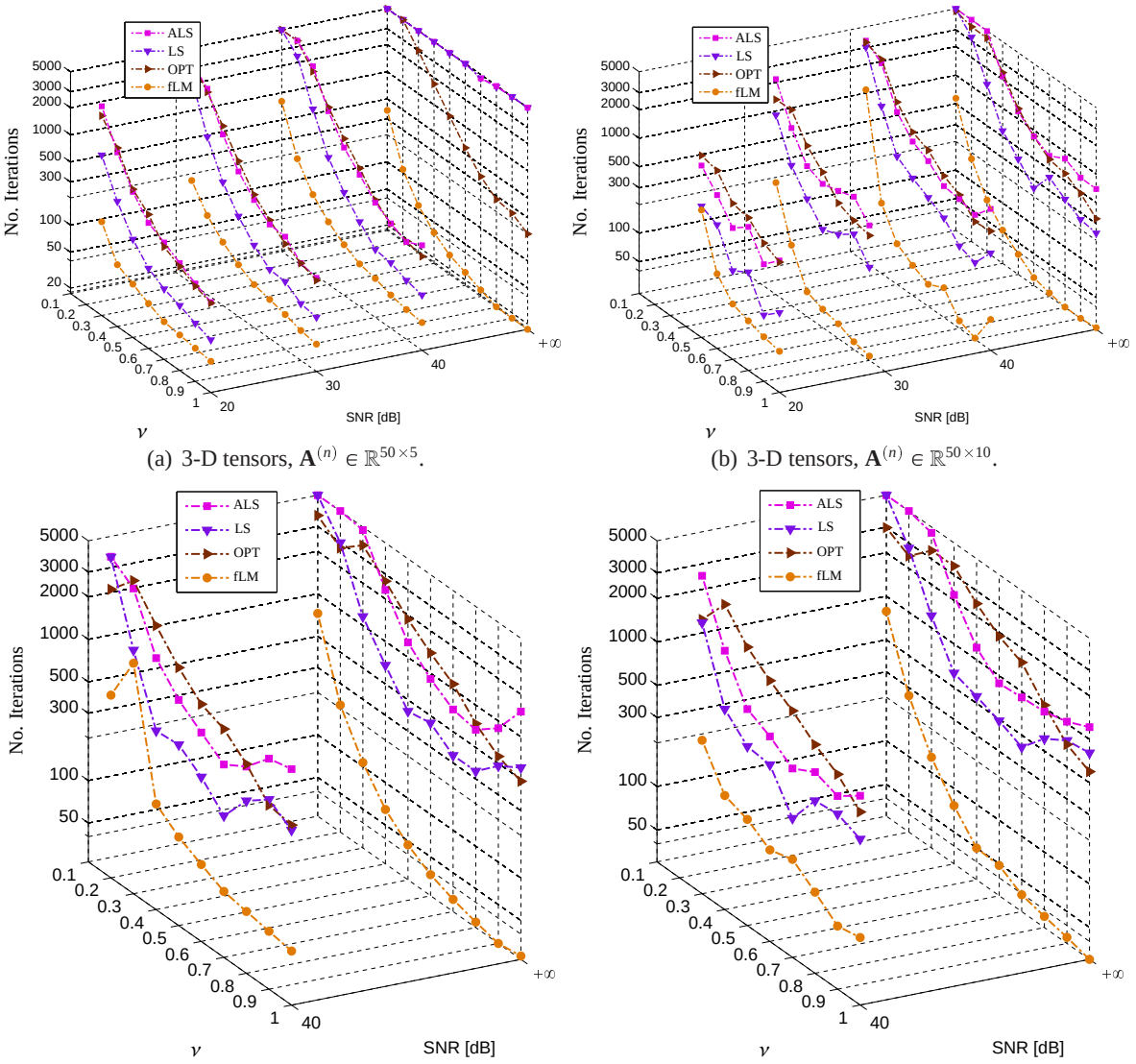


(c) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 20}$, SNR = $+\infty$ dB.



(d) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 20}$, SNR = 40 dB.

Figure 6.4: Average MSAE for factorization of 3-D tensors with size $I_n = 50$ and ranks $R = 15$ and 20 at SNR = 40 dB and $+\infty$ dB (noiseless tensor). Algorithms run until reaching a derivative of successive relative errors of 10^{-12} or the number of iterations of 5000.



(a) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 5}$.

(b) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 10}$.

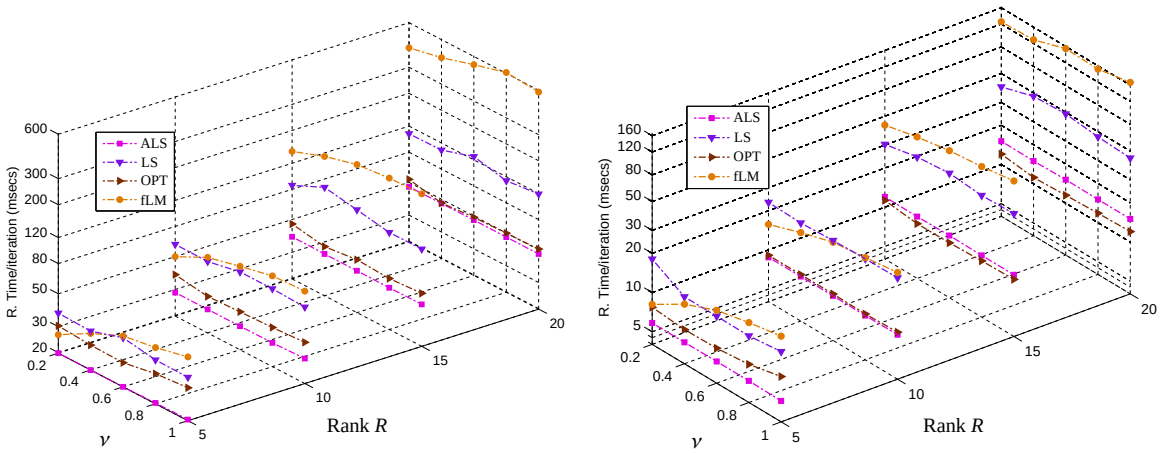
(c) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 15}$.

(d) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 20}$.

Figure 6.5: Number of iterations of CP algorithms as they reach a relative error of 10^{-12} for approximation of 3-D tensors with size $I_n = 50$ and ranks $R = 5, 10, 15, 20$ at SNR = 20 dB, 30 dB, 40 dB, and $+\infty$ dB.

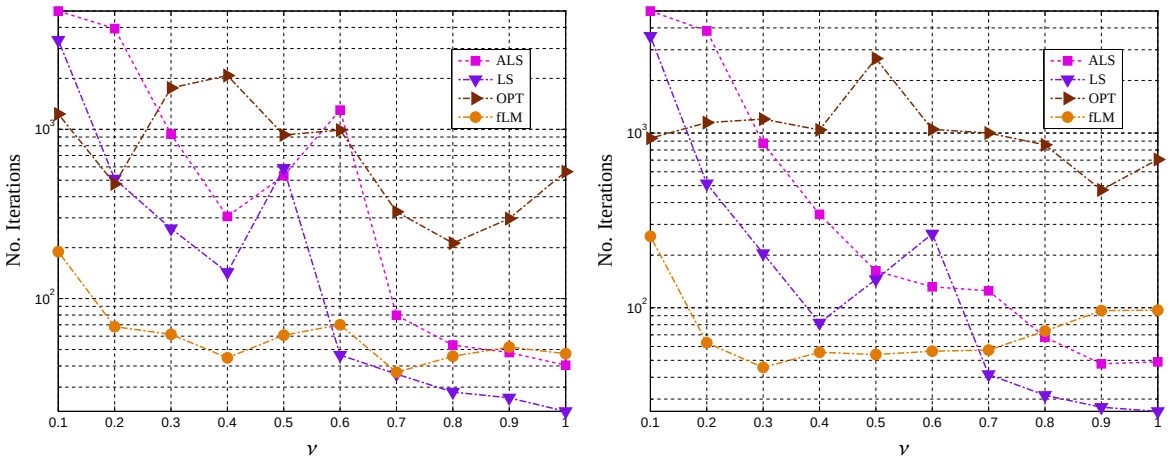
2.7 times measured on PC2. That means fLM is often faster than ALS if they are run under the same conditions. The ratios are 1.9, 10.9 7.8, 6.7 and 5.1 times for factorization of 3-D tensors of $R = 10$ at SNR = 40 dB, and are 2.5, 2.4, 1.6, 1.9, 3.5 times for $R = 15$.

Figures 6.5, 6.7 also reveal that LS is quite effective for ALS. The number of iterations of LS is always lower than that of ALS. That means LS stops at an earlier iteration. However, we cannot ensure it converges to the desired solution, and LS speeds up ALS because LS has higher complexity per one iteration than that of ALS (see in Figure 6.6). If LS runs 5000 iterations, ALS in LS runs 5000



(a) Core 2 Duo, 2.4 GHz. (b) 2 Quadcores, 3.33 GHz.

Figure 6.6: Running time (miliseconds) per iteration of CP algorithms for factorization of 3-D tensors with size $I_n = 50$, ranks $R = 5, 10, 15, 20$. (a) measured on a PC with an Intel Core 2 Duo 2.4 GHz processor, 2 GB memory. (b) measured on computing server that has 2 quadcore 3.33 GHz processors and 64 GB memory.



(a) 6-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 5}$, SNR = $+\infty$ dB. (b) 6-D tensors, $\mathbf{A}^{(n)} \in \mathbb{R}^{50\times 10}$, SNR = $+\infty$ dB.

Figure 6.7: Number of iterations of CP algorithms as they reach a relative error of 10^{-12} for approximation of 4-D and 6-D tensors with size $I_n = 50$ and ranks $R = 5, 10$.

iterations and its line search runs 2500 iterations. Therefore, it is not clear LS is faster than ALS unless the number of iterations of LS is significant less than that of ALS. For example, speed ratios between LS and ALS are 0.5, 1.2, 0.8, 0.7, 1.1 times for $\nu = 0.2, 0.4, 0.6, 0.8, 1$ when factorizing 3-D tensors with $R = 15$ at SNR = 40 dB.

For higher dimensional tensors ($N \geqslant 6$), fLM's complexity is at the same order of those of other algorithms. Hence, fLM might be faster than other algorithms. Number of iterations of fLM is often lower than those of ALS and LS for high collinearity degree as illustrated in Figure 6.7. This ensures that fLM converges faster than other algorithms for such difficult benchmarks even though complexity

Table 6.2: Description of structured matrices of size $n \times n$[69;87;88;122].

Matrix type	Description
1. binomial	Multiple of involutory matrix. An n-by-n matrix with integer entries: $\mathbf{A}^2 = 2^{(n-1)}\,\mathbf{I}_n$
2. Cauchy	An n-by-n positive matrix, $C(i,j) = 1/(i+j)$, $\det(\mathbf{C}) = 0$
3. Chebspec	Chebyshev spectral differentiation matrix of order n. $\mathbf{C}$ is nilpotent $(\mathbf{C}^n = 0)$, and has the null vector $\mathbf{1}_n$
4. ChebVand	Vandermonde-like matrix for the Chebyshev polynomials
4. Chow	Singular Toeplitz lower Hessenberg matrix: $\mathbf{A} = \mathbf{H}(1)$ has $[n/2]$ zero eigenvalues and the remain eigenvalues are equal to $4\cos(k\pi/(n+2))^2$.
6. Circulant	Toeplitz matrix whose first row is the vector $1:n$.
7. dramadah	Toeplitz matrix of zeros and ones, $\lvert\det(\mathbf{A})\rvert = 1$. Inverse has large integer entries.
8. gcdmat	Greatest common divisor matrix $A(i,j) = \gcd(i,j)$.
9. Hilbert	Hilbert matrix is a poorly conditioned matrix: $H(i,j) = 1/(i+j-1)$[69].
10. Lehmer	Symmetric positive definite matrix such that $A(i,j) = i/j,\ j >= i$
11. Lotkin	Lotkin matrix is Hilbert matrix with its first row altered to all ones. The Lotkin matrix A is nonsymmetric, ill-conditioned, and has many negative eigenvalues of small magnitude.
12. minij	Symmetric positive definite matrix: $A(i,j) = \min(i,j)$
13. pei-α	Pei matrix is a symmetric matrix $\alpha\mathbf{I}_n + \mathbf{1}_{n\times n}$.
14. triw	Upper triangular matrix
15. randsvd	A banded (multidiagonal) random matrix with $\mathrm{cond}(\mathbf{A}) = \sqrt{1/\varepsilon}$ and singular values from geometric distribution.
16. tridiag	Tridiagonal matrix (sparse) the Toeplitz tridiagonal matrix with subdiagonal and superdiagonal elements -1, diagonal elements 2.

of fLM per iteration is higher than those of others.

6.1.3 Structured Factors

This section considers difficult benchmarks with structured matrices which can be sparse, highly collinear, or ill-condition. The list of structured matrices is given in Table 6.2. Factors are leading left columns of structured matrices of the same size. That is in order to construct a factor of size 50×10 from the Hilbert matrix[69], a full Hilbert matrix of order-50 is generated, then 10 left columns of this matrix is picked up to form the factor. All the tensors were constructed with $\lambda_1 = \ldots = \lambda_N = 1$.

For pei matrix, α can be one of four values -2, -1, 1, 2. Hence, there might be in total 21 structured matrices. We verified only 3-D tensors with $I_n = 50$ or $I_n = 100$, and factors can consist of 5, 10, 15 or 20 components.

Factorization results were evaluated through SIR indices between estimated and original components. For $I_n = 50$, $R = 5$ and $R = 10$, SIR indices are given in Table 6.3, whereas for higher rank approximations $R = 15$, $I_n = 50$ and $R = 20$, $I_n = 100$, SIRs are in Table 6.4. Relative errors for these experiments are given in Table 6.5. The results show that the ALS algorithm fail in most cases.

Table 6.3: Comparison of SIR indices for algorithms in Example 6.1.3. Tensors were composed from structured factors. A good algorithm provides high SIR between original and estimated components.

Factor type	ALS	LS	ELS	HALS	OPT	fLM
$R = 5, I_n = 50$						
minij	10.85 ± 8.96	10.84 ± 8.95	10.84 ± 8.95	10.82 ± 8.89	9.21 ± 7.79	**12.19 ± 8.64**
Hilbert	20.88 ± 6.33	23.08 ± 9.86	20.88 ± 6.32	20.14 ± 7.96	24.36 ± 5.91	**25.62 ± 8.95**
Lotkin	23.50 ± 6.83	23.34 ± 6.36	23.23 ± 5.86	23.70 ± 7.68	28.07 ± 8.59	**28.88 ± 9.21**
Cauchy	25.28 ± 8.21	24.68 ± 5.88	24.06 ± 5.69	23.49 ± 4.64	30.68 ± 11.29	**29.52 ± 9.23**
Chow	5.17 ± 7.02	8.02 ± 10.16	5.46 ± 8.78	4.58 ± 6.62	3.57 ± 6.18	**30.46 ± 55.84**
Circulant	12.30 ± 3.34	12.84 ± 6.79	12.30 ± 3.34	11.95 ± 4.59	16.47 ± 4.39	**119.26 ± 4.96**
Chebspec	34.49 ± 17.71	39.63 ± 18.49	34.49 ± 17.71	43.45 ± 19.08	24.74 ± 15.21	**134.60 ± 18.22**
ChebVand	9.13 ± 5.98	19.46 ± 10.53	9.13 ± 5.98	16.38 ± 7.11	19.86 ± 9.43	**150.83 ± 42.56**
pei-1	10.17 ± 2.58	13.21 ± 2.51	10.16 ± 2.58	9.91 ± 2.55	11.75 ± 2.58	**164.97 ± 54.87**
Lehmer	17.65 ± 6.65	38.71 ± 19.06	17.65 ± 6.65	11.99 ± 5.55	23.37 ± 10.82	**169.55 ± 82.75**
pei1	9.87 ± 2.59	12.73 ± 2.52	9.89 ± 2.58	9.71 ± 2.56	13.79 ± 2.52	**186.19 ± 71.08**
pei2	21.04 ± 2.40	81.71 ± 2.31	21.04 ± 2.40	18.93 ± 2.19	54.53 ± 2.80	**196.88 ± 75.50**
pei-2	24.72 ± 2.36	98.85 ± 2.29	24.75 ± 2.36	22.02 ± 1.92	84.27 ± 2.22	**197.07 ± 75.35**
gcdmat	35.17 ± 27.54	60.64 ± 41.63	35.17 ± 27.54	24.93 ± 22.26	30.32 ± 23.38	**200.54 ± 123.06**
dramadah	26.38 ± 9.93	51.62 ± 12.72	26.38 ± 9.93	18.65 ± 1.94	54.67 ± 14.98	**201.05 ± 72.45**
triw	61.54 ± 9.07	123.57 ± 8.99	61.54 ± 9.07	121.68 ± 9.02	99.17 ± 7.33	**269.28 ± 63.60**
randsvd	177.49 ± 63.72	230.63 ± 76.80	164.41 ± 55.49	163.72 ± 55.67	137.59 ± 9.33	**280.87 ± 50.48**
binomial	34.97 ± 25.12	98.60 ± 100.36	34.95 ± 25.10	71.97 ± 63.50	48.86 ± 36.29	**300.00 ± 0.00**
tridiag	106.70 ± 5.68	126.58 ± 5.70	106.70 ± 5.68	125.25 ± 5.98	9.02 ± 9.22	**300.00 ± 0.00**
$R = 10, I_n = 50$						
minij	11.21 ± 6.86	11.76 ± 6.92	12.37 ± 7.01	10.43 ± 6.70	10.29 ± 7.18	**16.04 ± 9.23**
ChebVand	7.62 ± 4.14	11.43 ± 6.20	11.23 ± 5.57	16.87 ± 9.23	13.39 ± 5.09	**18.10 ± 10.73**
Cauchy	22.02 ± 6.61	21.60 ± 6.13	22.48 ± 6.29	21.68 ± 6.04	27.01 ± 8.42	**27.37 ± 8.09**
Lotkin	22.50 ± 5.69	21.88 ± 6.47	22.17 ± 6.33	22.70 ± 6.34	27.17 ± 8.87	**28.43 ± 9.96**
Hilbert	18.99 ± 5.64	18.78 ± 5.80	17.94 ± 5.55	19.48 ± 5.17	24.51 ± 6.21	**30.54 ± 11.06**
binomial	13.96 ± 7.59	17.25 ± 10.72	15.45 ± 9.13	14.53 ± 8.82	18.14 ± 9.30	**31.85 ± 19.20**
Chebspec	30.06 ± 18.59	30.07 ± 18.59	30.37 ± 18.54	28.01 ± 19.00	34.90 ± 23.71	**35.46 ± 23.73**
Lehmer	10.80 ± 3.57	13.14 ± 5.38	13.97 ± 5.97	9.10 ± 4.38	16.87 ± 7.98	**65.59 ± 47.36**
Chow	6.84 ± 5.66	6.85 ± 5.36	5.91 ± 5.07	5.28 ± 5.72	7.03 ± 5.86	**73.39 ± 58.45**
pei1	12.15 ± 2.97	16.35 ± 2.82	14.25 ± 2.88	10.97 ± 3.01	11.17 ± 3.09	**110.96 ± 2.65**
Circulant	10.12 ± 3.65	13.07 ± 5.30	11.63 ± 5.02	8.00 ± 4.08	11.56 ± 1.99	**131.18 ± 2.15**
pei-1	12.38 ± 2.96	14.65 ± 2.87	14.63 ± 2.87	11.12 ± 3.01	9.82 ± 3.61	**167.02 ± 53.17**
pei-2	21.47 ± 2.73	76.18 ± 2.58	40.32 ± 2.60	17.58 ± 2.65	57.33 ± 3.20	**172.03 ± 58.30**
tridiag	18.10 ± 8.10	21.47 ± 9.06	58.50 ± 6.66	134.22 ± 5.22	12.63 ± 13.39	**173.34 ± 64.49**
dramadah	14.65 ± 7.96	14.12 ± 6.62	14.16 ± 6.96	20.87 ± 6.97	14.27 ± 5.18	**203.51 ± 74.85**
pei2	19.50 ± 2.77	56.88 ± 2.60	31.94 ± 2.64	16.22 ± 2.74	33.19 ± 2.61	**212.57 ± 77.85**
randsvd	32.39 ± 8.36	123.21 ± 9.45	116.41 ± 9.00	116.28 ± 9.99	17.30 ± 6.66	**214.98 ± 75.75**
gcdmat	13.57 ± 15.34	14.56 ± 16.03	14.65 ± 16.08	15.64 ± 15.47	17.83 ± 15.27	**215.33 ± 102.09**
triw	60.37 ± 15.62	90.19 ± 15.65	90.12 ± 15.55	118.82 ± 15.76	46.01 ± 24.74	**238.74 ± 76.32**

In Tables 6.3 and 6.4, successful factorizations with high SIR > 20 dB are emphasized in blue.

For $R = 5$, $I_n = 50$, ALS achieved good performances for matrices: *triw*, *randsvd*, *tridiag*. Line search can improve the ALS for some experiments, but not all. Although ELS was designed to be better than LS, this algorithm could not reveal its ability for these difficult benchmarks. The performances of ELS were not better than those of LS. The HALS algorithm faces the same problem for the difficult benchmarks. HALS did not explain the data well. However, HALS with linesearch can achieve quite

Table 6.4: Comparison of SIR indices for algorithms in Example 6.1.3. Tensors were composed from structured factors. A good algorithm provides high SIR between original and estimated components.

Factor type	ALS	LS	ELS	HALS	OPT	fLM
$R = 15$, $I_n = 50$						
minij	12.46 ± 6.19	12.08 ± 5.62	12.46 ± 6.19	10.64 ± 5.91	12.13 ± 6.44	**16.08 ± 6.73**
ChebVand	11.40 ± 6.75	12.17 ± 6.91	9.52 ± 6.08	11.75 ± 4.36	14.30 ± 8.74	**18.09 ± 11.42**
Cauchy	20.58 ± 6.68	22.83 ± 6.04	22.76 ± 6.88	21.35 ± 5.66	27.00 ± 9.76	**26.69 ± 8.07**
Hilbert	21.71 ± 5.98	20.96 ± 6.82	21.36 ± 5.03	19.01 ± 4.71	24.25 ± 6.16	**27.44 ± 8.07**
Lotkin	23.61 ± 7.65	26.04 ± 8.13	24.78 ± 6.68	23.35 ± 5.82	27.74 ± 8.08	**29.36 ± 9.92**
Lehmer	10.94 ± 5.11	13.17 ± 8.53	10.94 ± 5.11	8.71 ± 5.17	16.06 ± 8.15	**63.88 ± 75.39**
pei1	13.77 ± 2.91	17.89 ± 2.73	13.79 ± 2.91	12.08 ± 5.24	11.57 ± 3.77	**108.70 ± 2.53**
Chow	7.15 ± 5.03	8.19 ± 5.84	6.69 ± 5.11	6.65 ± 4.72	7.98 ± 5.24	**110.02 ± 81.36**
pei-1	13.99 ± 2.90	17.56 ± 2.74	13.99 ± 2.90	12.54 ± 3.03	8.35 ± 5.52	**123.03 ± 2.52**
Circulant	9.88 ± 3.36	11.30 ± 3.93	10.41 ± 4.43	7.69 ± 4.49	9.81 ± 1.80	**125.87 ± 6.24**
pei2	19.88 ± 2.69	28.24 ± 2.55	19.87 ± 2.69	16.69 ± 7.06	20.11 ± 8.57	**130.68 ± 2.50**
pei-2	21.35 ± 2.65	33.94 ± 2.51	21.35 ± 2.65	17.00 ± 7.28	30.41 ± 2.57	**139.65 ± 2.41**
gcdmat	20.79 ± 12.05	26.15 ± 14.50	20.79 ± 12.05	17.24 ± 15.42	18.62 ± 14.29	**203.94 ± 91.36**
Chebspec	19.49 ± 13.38	18.16 ± 13.48	19.49 ± 13.38	50.35 ± 28.50	58.31 ± 20.48	**227.22 ± 79.27**
triw	75.50 ± 23.12	102.62 ± 20.51	75.50 ± 23.12	107.71 ± 19.61	55.17 ± 24.15	**233.18 ± 75.59**
dramadah	26.65 ± 11.54	29.92 ± 14.93	26.65 ± 11.54	23.73 ± 11.60	28.46 ± 11.88	**237.80 ± 73.61**
randsvd	9.26 ± 4.54	11.21 ± 5.36	9.26 ± 4.54	37.65 ± 8.80	8.11 ± 4.23	**267.44 ± 61.61**
binomial	14.69 ± 7.50	14.83 ± 7.44	14.69 ± 7.50	11.74 ± 5.98	20.23 ± 10.29	**300.00 ± 0.00**
tridiag	45.68 ± 11.08	109.10 ± 6.88	45.68 ± 11.08	120.50 ± 7.52	7.81 ± 4.31	**300.00 ± 0.00**
$R = 20$, $I_n = 100$						
minij	11.12 ± 5.76	11.12 ± 5.49	13.07 ± 6.44	10.73 ± 5.44	12.35 ± 6.02	**15.64 ± 6.00**
ChebVand	8.30 ± 5.24	12.19 ± 7.17	12.63 ± 6.88	11.55 ± 4.23	12.36 ± 4.38	**17.58 ± 12.21**
tridiag	19.85 ± 12.69	19.67 ± 12.74	19.43 ± 12.70	24.37 ± 20.35	12.09 ± 9.37	**18.38 ± 12.11**
Hilbert	21.60 ± 6.28	20.21 ± 5.62	19.78 ± 6.73	19.79 ± 4.79	**27.05 ± 9.75**	23.87 ± 6.11
Lotkin	24.21 ± 7.39	26.45 ± 6.72	23.78 ± 6.92	22.98 ± 5.86	**28.45 ± 9.48**	26.35 ± 7.47
binomial	15.41 ± 6.56	15.19 ± 8.02	13.99 ± 5.91	12.68 ± 6.01	20.74 ± 8.20	**26.83 ± 17.26**
Cauchy	21.29 ± 6.03	26.10 ± 5.54	25.06 ± 7.14	20.73 ± 5.28	26.56 ± 6.53	28.43 ± 8.07
dramadah	16.30 ± 8.19	21.20 ± 9.27	18.44 ± 10.21	15.31 ± 5.88	19.62 ± 3.81	**48.20 ± 18.59**
Chow	6.30 ± 4.84	8.30 ± 6.66	7.50 ± 5.02	6.07 ± 4.25	8.56 ± 4.94	**74.44 ± 43.00**
Lehmer	10.20 ± 4.66	12.25 ± 7.28	10.74 ± 5.98	9.43 ± 4.87	17.70 ± 6.86	**82.31 ± 89.57**
triw	86.52 ± 61.83	77.79 ± 44.69	76.71 ± 43.59	51.46 ± 31.69	49.88 ± 24.89	**84.49 ± 55.61**
pei-1	14.11 ± 2.84	17.17 ± 2.75	15.52 ± 2.79	12.38 ± 3.10	11.95 ± 4.97	**91.89 ± 2.61**
pei1	14.02 ± 2.84	16.71 ± 2.76	15.41 ± 2.80	12.37 ± 3.05	14.19 ± 2.82	**92.04 ± 2.61**
Circulant	10.59 ± 4.26	12.88 ± 4.72	12.81 ± 5.05	7.03 ± 1.99	11.63 ± 1.99	**126.83 ± 8.63**
pei-2	18.94 ± 2.73	30.86 ± 2.62	23.79 ± 2.66	14.55 ± 2.87	21.41 ± 2.69	**132.11 ± 2.52**
pei2	18.51 ± 2.73	28.59 ± 2.63	22.90 ± 2.67	14.36 ± 2.89	21.21 ± 2.68	**156.97 ± 48.24**
gcdmat	22.61 ± 20.34	32.37 ± 25.51	25.35 ± 21.51	23.07 ± 20.13	21.18 ± 15.21	**187.29 ± 84.00**
Chebspec	23.13 ± 15.78	52.07 ± 27.19	50.96 ± 26.64	49.98 ± 25.92	84.60 ± 29.62	**230.27 ± 78.29**
randsvd	150.11 ± 35.44	196.67 ± 71.11	138.55 ± 22.50	141.79 ± 7.08	111.40 ± 5.95	**248.73 ± 70.48**

good performances for some benchmarks such as matrices *Chebspec, triw, randsvd, tridiag*.

The OPT seems better than ALS and HALS algorithms due to update all the entries simultaneously. For low rank $R = 5$, OPT easily outperformed (H)ALS algorithms. For example, for *pei2, per-2*, and *dramadah* matrices, OPT estimated components with averaged SIR = 54 dB, 84.27 dB and 54 dB respectively. However, for higher rank $R = 10, 15$ with higher number of entries to be updated, the performances of OPT were reduced. For example, SIRs for *pei2, per-2* and *dramadah* matrices

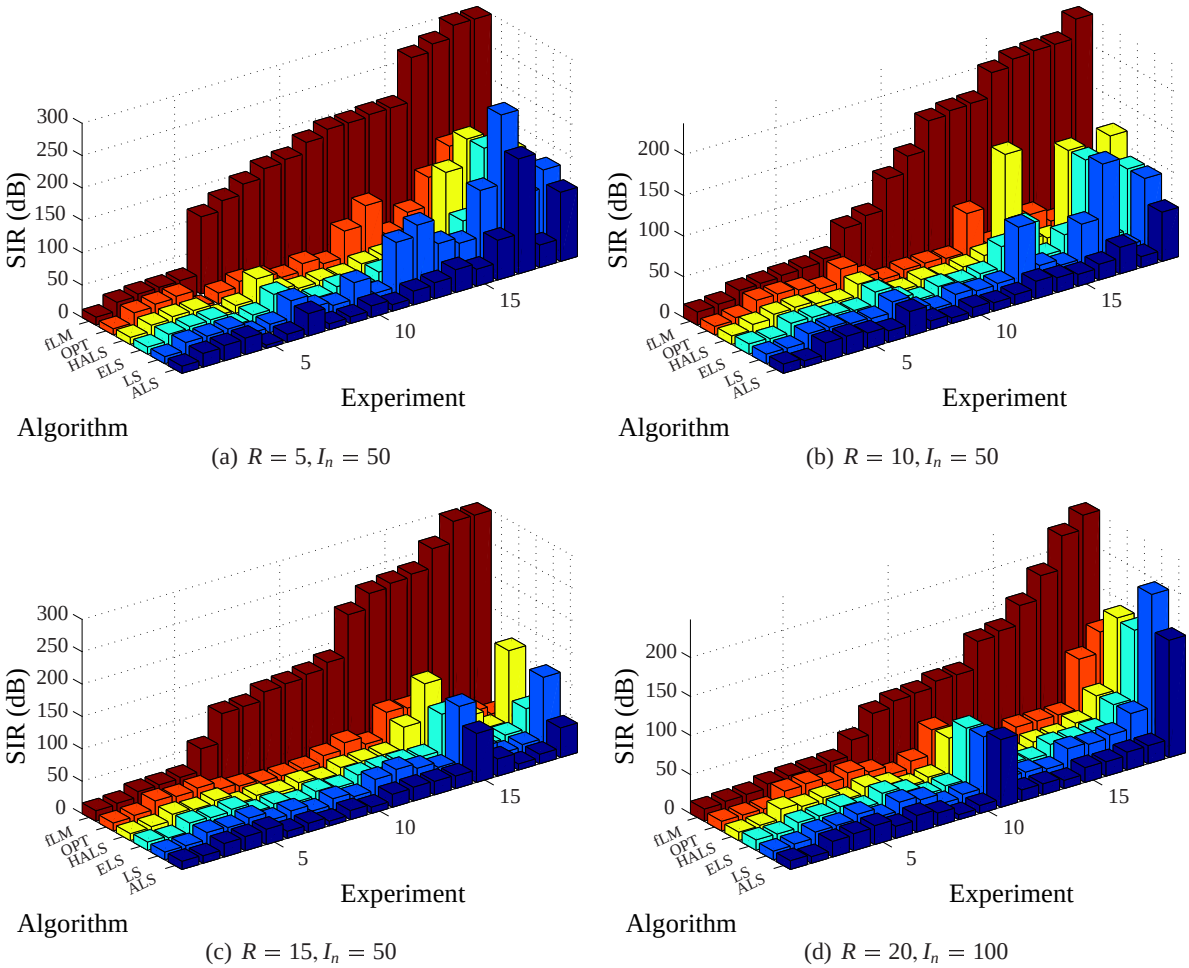
Table 6.5: Fitness comparison for algorithms in Example 6.1.3.

Factor type	ALS	LS	ELS	HALS	OPT	fLM	ALS	LS	ELS	HALS	OPT	fLM
	$R = 5, I_n = 50$						$R = 10, I_n = 50$					
minij	1.83e-4	1.83e-4	1.83e-4	1.83e-4	3.90e-4	**9.90e-6**	7.85e-5	7.33e-5	7.24e-5	7.13e-5	6.34e-4	**8.15e-6**
Hilbert	1.78e-4	7.63e-5	1.78e-4	8.67e-5	4.67e-4	**1.29e-6**	2.32e-5	2.42e-5	5.45e-6	6.63e-7	1.67e-3	**8.76e-8**
Lotkin	3.82e-5	2.33e-6	6.72e-5	1.33e-5	8.42e-5	**6.16e-7**	3.99e-6	4.84e-5	5.62e-6	5.14e-7	4.32e-4	**4.24e-8**
Cauchy	1.73e-5	1.20e-5	1.78e-5	1.38e-5	1.13e-4	**5.16e-7**	3.88e-6	2.28e-6	6.56e-7	3.95e-7	6.50e-4	**7.98e-8**
Chow	4.01e-4	2.85e-4	5.24e-4	5.52e-4	1.14e-3	**0**	2.12e-3	2.07e-3	2.07e-3	2.07e-3	1.82e-3	**0**
Circulant	6.16e-3	6.05e-3	6.16e-3	6.14e-3	1.86e-3	**0**	5.11e-3	3.31e-3	5.78e-3	6.40e-3	3.82e-3	**0**
Chebspec	2.36e-3	1.25e-3	2.36e-3	8.29e-4	6.85e-3	**0**	4.01e-2	4.01e-2	4.01e-2	4.01e-2	3.04e-2	**1.90e-2**
ChebVand	1.94e-2	8.49e-4	1.94e-2	1.33e-3	8.63e-4	**0**	1.94e-2	4.41e-3	6.55e-3	5.20e-4	7.75e-3	**1.16e-5**
pei-1	1.23e-3	7.42e-4	1.23e-3	1.31e-3	9.61e-4	**3.86e-8**	8.70e-4	5.48e-4	5.50e-4	1.26e-3	9.76e-4	**5.48e-8**
Lehmer	3.28e-3	2.83e-4	3.28e-3	8.14e-3	3.04e-3	**0**	1.63e-3	1.33e-3	1.40e-3	2.29e-3	3.60e-3	**1.08e-4**
pei1	1.16e-3	7.06e-4	1.15e-3	1.22e-3	6.16e-4	**0**	8.19e-4	3.72e-4	5.22e-4	1.17e-3	8.77e-4	**0**
pei2	1.87e-3	1.86e-6	1.87e-3	2.45e-3	3.70e-5	**0**	1.84e-3	3.01e-5	4.85e-4	2.92e-3	4.25e-4	**0**
pei-2	1.57e-3	3.27e-7	1.56e-3	2.17e-3	1.73e-6	**0**	1.86e-3	4.14e-6	2.47e-4	3.08e-3	3.54e-5	**4.21e-8**
gcdmat	1.17e-3	5.58e-5	1.17e-3	6.74e-3	2.43e-3	**0**	7.98e-3	7.40e-3	7.41e-3	7.61e-3	6.20e-3	**0**
dramadah	2.18e-2	1.00e-3	2.18e-2	1.51e-2	9.46e-4	**0**	2.75e-2	2.71e-2	2.71e-2	1.31e-2	2.18e-2	**0**
triw	2.02e-4	1.87e-7	2.02e-4	1.99e-7	7.02e-6	**0**	4.30e-4	1.38e-5	1.38e-5	5.17e-7	1.08e-1	**0**
randsvd	1.71e-8	1.71e-8	4.19e-8	3.83e-8	1.03e-7	**0**	6.02e-3	1.84e-7	3.42e-7	4.32e-7	4.95e-2	**0**
binomial	1.10e-3	1.38e-4	1.10e-3	3.97e-4	4.50e-4	**0**	1.50e-3	8.53e-4	8.97e-4	2.87e-3	4.62e-3	**2.61e-5**
tridiag	1.33e-6	1.35e-7	1.33e-6	1.61e-7	3.28e-1	**0**	2.96e-2	2.50e-2	2.63e-4	6.84e-8	4.05e-1	**0**
	$R = 15, I_n = 50$						$R = 20, I_n = 100$					
minij	3.99e-5	3.69e-5	3.99e-5	2.29e-5	7.35e-4	**4.26e-6**	1.57e-5	1.54e-5	1.54e-5	1.06e-5	4.19e-4	**1.24e-6**
ChebVand	3.83e-3	2.27e-3	5.84e-3	7.83e-4	1.17e-2	**9.87e-6**	6.32e-3	1.13e-3	2.64e-3	6.73e-4	5.98e-3	**2.18e-6**
Cauchy	4.19e-7	3.37e-6	2.74e-6	2.88e-7	1.03e-3	**0**	1.86e-6	3.02e-6	1.58e-6	6.19e-7	1.01e-3	**0**
Hilbert	1.93e-5	1.44e-5	2.87e-6	2.38e-6	2.20e-3	**0**	4.20e-6	1.83e-6	2.59e-6	2.39e-6	3.21e-4	**0**
Lotkin	6.03e-7	1.52e-6	1.95e-6	1.53e-6	6.01e-4	**0**	1.74e-6	4.90e-7	6.49e-8	1.92e-7	6.71e-4	**0**
Lehmer	1.25e-3	7.64e-4	1.25e-3	7.11e-4	3.46e-3	**2.18e-5**	8.75e-4	4.81e-4	7.92e-4	5.94e-4	2.26e-3	**6.74e-6**
pei1	6.57e-4	2.79e-4	6.53e-4	2.41e-3	7.29e-4	**0**	2.57e-4	1.24e-4	1.68e-4	5.73e-4	2.40e-4	**0**
Chow	2.04e-3	1.66e-3	2.36e-3	2.08e-3	2.71e-3	**0**	1.03e-3	9.00e-4	1.02e-3	9.17e-4	1.20e-3	**2.51e-8**
pei-1	6.97e-4	3.32e-4	6.95e-4	1.15e-3	9.06e-4	**6.75e-8**	2.65e-4	1.19e-4	1.73e-4	5.98e-4	2.94e-4	**0**
Circulant	4.65e-3	4.54e-3	4.61e-3	5.72e-3	7.89e-3	**0**	2.46e-3	2.26e-3	2.03e-3	2.86e-3	2.87e-3	**0**
pei2	1.57e-3	6.38e-4	1.57e-3	8.86e-3	8.68e-3	**0**	6.78e-4	2.10e-4	3.74e-4	1.82e-3	4.58e-4	**0**
pei-2	1.67e-3	4.55e-4	1.67e-3	1.03e-2	6.65e-4	**0**	7.12e-4	1.90e-4	3.82e-4	1.92e-3	5.04e-4	**0**
gcdmat	3.06e-3	1.99e-3	3.06e-3	9.60e-3	7.84e-3	**0**	6.39e-3	1.79e-3	3.90e-3	5.88e-3	9.53e-3	**0**
Chebspec	2.94e-1	2.93e-1	2.94e-1	1.50e-2	1.40e-3	**0**	2.50e-1	2.63e-2	2.64e-2	2.64e-2	4.30e-4	**0**
triw	2.15e-4	7.28e-6	2.15e-4	3.75e-6	4.35e-2	**0**	2.27e-2	2.27e-2	2.27e-2	3.36e-2	8.54e-2	**0**
dramadah	1.24e-2	1.23e-2	1.24e-2	1.25e-2	1.27e-2	**0**	1.24e-2	8.99e-3	1.05e-2	9.81e-3	3.71e-3	**0**
randsvd	4.10e-2	2.51e-2	4.10e-2	1.98e-3	9.96e-2	**0**	3.54e-8	4.33e-8	7.30e-8	5.31e-8	1.41e-6	**0**
binomial	2.08e-3	1.60e-3	2.08e-3	5.85e-4	3.78e-3	**6.49e-6**	7.99e-4	9.52e-4	8.02e-4	1.52e-4	1.84e-3	**8.52e-7**
tridiag	2.05e-3	7.20e-7	2.05e-3	2.04e-7	3.52e-1	**0**	2.16e-1	2.16e-1	2.16e-1	2.16e-1	3.05e-1	**2.13e-1**

estimated by OPT are 33 dB, 57 dB 33 dB for $R = 10$ and 20 dB, 30 dB and 28 dB for $R = 15$. However, OPT did not succeed in factorizing other structured tensors such as chow, circulant, tridiag matrices.

For all the experiments, the fLM algorithms always achieved the best performances, and successfully estimated components for most structured matrices. The fLM algorithms overwhelmingly outperformed all the other algorithms. Figure 6.8 illustrates SIR indices for CP algorithms.

In a further example, we incorporated 20dB Gaussian noises to structured tensors. CP algorithms



(a) $R = 5, I_n = 50$ (b) $R = 10, I_n = 50$

(c) $R = 15, I_n = 50$ (d) $R = 20, I_n = 100$

Figure 6.8: SIR indices for CP algorithms in Example 6.1.3 for different rank R and dimension I_n.

Table 6.6: Comparison of SIR indices for algorithms factorizing noisy structured tensors in Example 6.1.3. Tensors were composed from structured factors. A good algorithm provides high SIR between original and estimated components.

Factor type	ALS	LS	ELS	HALS	OPT	fLM
$R = 15, I_n = 50$						
Cauchy	22.86 ± 5.11	23.23 ± 4.48	22.86 ± 5.11	23.02 ± 5.13	**26.98 ± 4.84**	**26.97 ± 4.81**
Chebspec	32.07 ± 14.77	35.05 ± 14.06	32.07 ± 14.77	36.91 ± 12.76	20.58 ± 15.02	**41.58 ± 9.19**
dramadah	26.44 ± 9.93	33.85 ± 10.12	26.44 ± 9.93	19.50 ± 4.91	**38.46 ± 3.77**	**38.45 ± 3.72**
Lehmer	16.73 ± 9.19	17.71 ± 9.41	16.73 ± 9.19	12.50 ± 6.44	**24.30 ± 8.24**	**24.38 ± 8.43**
Lotkin	22.85 ± 5.45	24.53 ± 5.64	22.85 ± 5.45	22.48 ± 5.90	**26.69 ± 6.45**	**26.66 ± 6.41**
orthog	42.82 ± 10.41	43.54 ± 9.31	42.82 ± 10.41	**45.62 ± 3.18**	42.02 ± 11.63	**45.62 ± 3.18**
triw	**46.23 ± 2.29**	**46.22 ± 2.33**	**46.22 ± 2.34**	**46.22 ± 2.33**	43.01 ± 10.75	**46.22 ± 2.30**
Hilbert	18.66 ± 5.50	19.37 ± 5.84	18.66 ± 5.50	18.94 ± 5.61	**24.95 ± 7.21**	**24.95 ± 7.20**

factorized the noisy tensors, and SIR indices were evaluated for the estimated factors. The results are given in Table 6.6. For some benchmarks such as *Cauchy, dramadah, Lehmer, Lotkin, Hilbert*, the performances of OPT and fLM are almost similar. However, for benchmarks *chebsepc, orthog, triw*, OPT's performances are even worse than (H)-ALS algorithms. The results also show that fLM outperformed other algorithms for noisy tensors.

6.1.4 Factorization of Complex-valued Tensors

In the next set of simulations, we considered factorization of complex-valued tensors. Factors $\mathbf{A}^{(n)} \in \mathbb{C}^{50\times R}$ were generated in the same manner as for experiments in the previous section. However, they had random real and imaginary parts. In addition to collinearity degrees $\nu = 0.1, 0.2, \ldots, 0.5$, we considered $\nu = 0.05$ for mutual angles $\theta_{1,r} \approx 3^{\text{o}}, r \neq 1$, and $\nu = 3, 4, 5$. We note that although collinearity of factors is low for high $\nu = 3, 4, 5$ ($\theta_{1,r} > 71^{\text{o}}$), the tensors are still difficult to factorize.

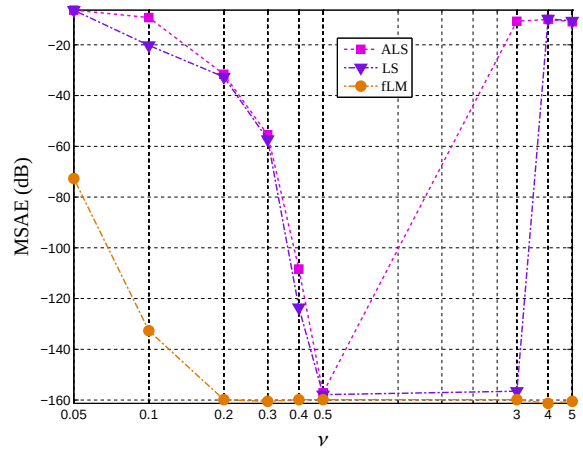
Normally, the CP algorithms can straightforwardly extended to handle complex-valued tensors. However, due to missing extensions of the ELS, OPT algorithms, we only compared ALS, LS and fLM algorithms. Algorithms were stopped as differences between successive relative errors are lower than 10^{-8}, or maximum number of iterations (2000) is exceeded.

In Figures 6.9(a)-(b), we illustrate the average MSAE of all factors for $50 \times 50 \times 50$ dimensional tensors with ranks $R = 5$ and 15 over 100 runs. ALS and LS achieved good performance at $\nu = 0.3$, and excellent MSAE at $\nu = 0.4$ and 0.5. However, for high $\nu = 3$, 4 and 5, ALS completely fails as for $\nu = 0.05$, 0.1 and 0.2. LS seems better for $R = 5$, but it is not efficient for $R = 10$. The fLM algorithm achieved perfect estimations for all test cases. Figures 6.9(c)-(d) indicate that the number of iterations of ALS and LS tends to decrease gradually as ν increases from 0.05 to 0.5 and even to 1. However, ALS and LS still need at least 1000 iterations in order to successfully factorize 3-D tensors of rank $R = 5$. For $\nu = 3$, 4 and 5, the number of iterations of ALS and LS increases again, and quickly passes over the maximum value of 2000 while they still get stuck in local minima. For rank $R = 10$, both ALS and LS stopped after tens of iterations because there is not any significant change in the relative error. Figures 6.9(c)-(d) also reveals that fLM requires less iterations for higher ν. Difference in magnitude between components does not affect to fLM. For tensors with $R = 15$, $\nu = 5$, the number of iterations of fLM is rather high. It can be caused by initial value of damping parameter.

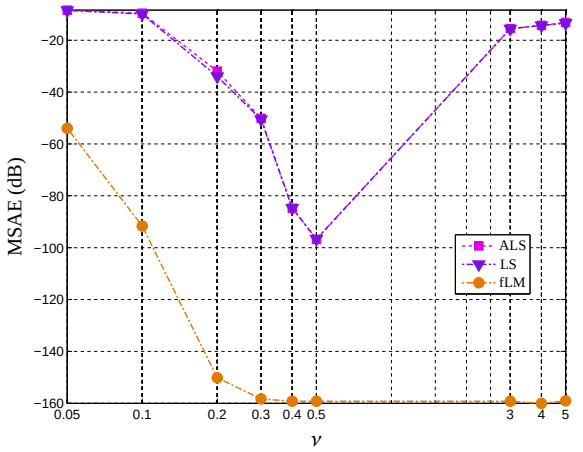
6.2 Simulations for NTF

6.2.1 Random Data

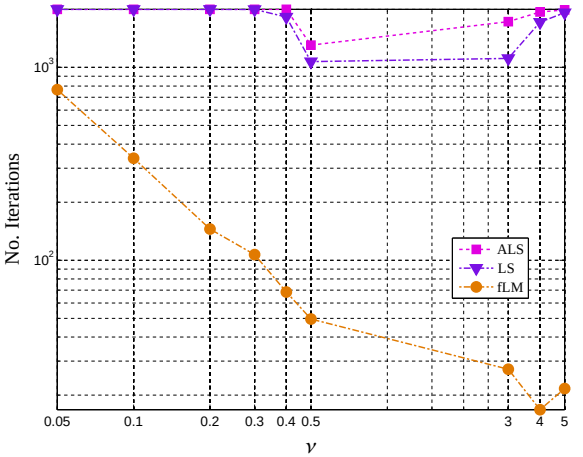
We constructed 3-D synthetic tensors $\mathcal{Y}$ with $I_1 = I_2 = I_3 = 100$ composed from random factors comprising $R = 10$ components. The components were forced to be collinear with others. All the



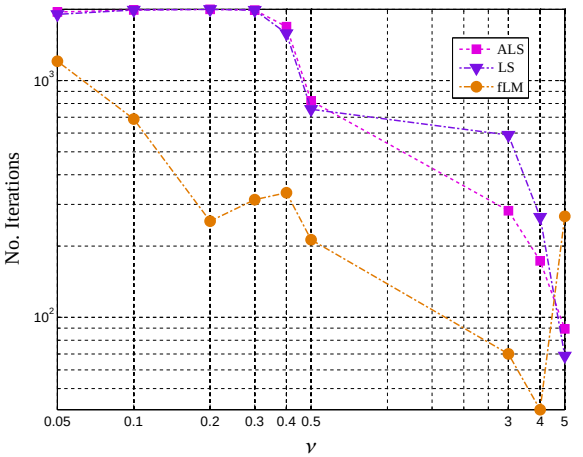
(a) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{C}^{50\times 5}$, SNR = $+\infty$ dB.



(b) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{C}^{50\times 15}$, SNR = $+\infty$ dB.



(c) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{C}^{50\times 5}$.



(d) 3-D tensors, $\mathbf{A}^{(n)} \in \mathbb{C}^{50\times 15}$.

Figure 6.9: Illustration for MSAE for factorization of 3-D complex-valued tensors with size $I_n = 50$ and ranks $R = 5, 15$. Algorithms stop as they reach a derivative of successive relative errors of 10^{-8} or 2000 iterations.

algorithms were initialized using leading singular components, and stopped when difference of the consecutive relative errors ε was lower than 10^{-10}, or the maximum number of iterations (200) was exceeded. Comparison of performances of various algorithms averaged over 100 runs is given as Example 1a in Table 6.8. The proposed algorithm achieved almost perfect performances with $\varepsilon \approx$ 2.87e-9 after only 67 (averaged) iterations. The other algorithms could not estimate hidden components in 200 iterations. To analyze their convergences, we set new stopping criterion with one million iterations, and $\varepsilon \leqslant 10^{-10}$. Most algorithms except the KL algorithm converged to the desired results with averaged SIRs $>$ 30 dB (given in the 3-rd row in Example 1a of Table 6.8). Figure 6.10 illustrates relative errors as functions of iterations for NTF algorithms for one MC run. The fLM$_+$ converged after 104 iterations, whereas the QALS stopped after 50K iterations. To yield comparable performances to that of the fLM$_+$ algorithm, the other algorithms need much more iterations. Moreover, in general the

other algorithms often achieve a different solution because the optimization problem is hard and the algorithms get stacked in a side local minimum more frequently than fLM$_+$.

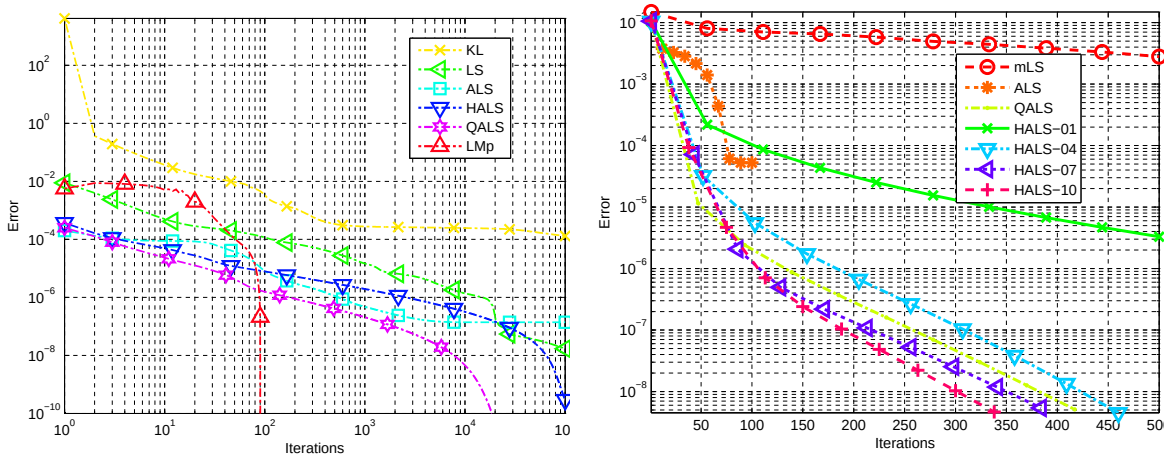
Next, we constructed 5-D synthetic tensors $\mathcal{Y}$ with $I_n = 100, (n = 1, \ldots, N)$. Factors comprising $R = 20$ components were drawn from uniform distribution, or sparse uniform distribution with a density of 0.3. Factors were forced to be collinear with others by the modification in (6.3). All the algorithms were analyzed under the same experimental conditions. Comparison of performances of various algorithms averaged over 100 runs is given in Table 6.8. For all the MC runs, the QALS algorithm achieved perfect factorizations with the lowest relative errors $\varepsilon < 10^{-8}$ after 500 or 1000 iterations. Factors estimated by the QALS algorithm have the highest SIR values for factorizations of dense and sparse tensors compared to those by other algorithms. For the same experiments, we analyzed the convergence after 25.000 iterations, the mLS algorithm achieved relative errors of $7.67\,10^{-7}$ and $9.91\,10^{-7}$ for dense and sparse tensors, respectively, whereas the HALS algorithm achieved $5.83\,10^{-7}$ and $6.95\,10^{-7}$. That means to explain a tensor at an equivalent relative error, the other algorithms need much more iterations than the QALS algorithm. Figs 6.11(a)-6.11(c) illustrate convergences of NTF algorithms for experiments 1-3 in Table 6.8.

In another experiment, we decomposed $1000 \times 1000 \times 1000$ dimensional sparse tensors composed by $R = 100$ collinear components. The proposed algorithm still overwhelmingly outperformed all the other algorithms. The HALS algorithm achieved better performances than those of the multiplicative KL and LS algorithms. A more intuitive visualization for convergences of algorithms is shown in Figure 6.11(d). The proposed algorithm explained the large-scale tensor well, while other algorithms were stacked in local minima, and did not improve the performance after 60 iterations.

In order to illustrate the rK-QALS algorithms, we analyzed a similar experiment in which 5-D synthetic tensors $\mathcal{Y}$ with $I_n = 100, (n = 1, \ldots, N)$ were composed by factors of $R = 10$ sparse random components which were forced to be collinear (6.3). Comparison of performances (relative error and SIR index) of various algorithms averaged over 100 runs is given in Table 6.7. Figure 6.10(b) illustrates convergences of the analyzed algorithms. The mLS algorithm converged very slowly, while the ALS algorithm stopped after 100 iterations. The QALS algorithm returned perfect results with SIR = 67.5dB. With various approximation ranks $\tilde{R} = 1, 2, \ldots, R$, the proposed method outperformed the mLS and ALS algorithms. And with suitable ranks $\tilde{R}$, the rK-QALS converged faster than the QALS algorithm. Although the HALS algorithm with $\tilde{R} = 1$ achieved much better performance than the mLS and ALS algorithms, its performance was still lower than that of the proposed algorithm with higher approximation ranks such as $\tilde{R} = 4, 7, 10$. We note that the r10-QALS algorithm in this experiment is different to the QALS algorithm due to strategy of component selection.

Table 6.7: Performance comparison for various algorithms for Example 6.2.1.

Algorithm	Error	Iteration	SIR (dB)
mLS	$(2.11 \pm 0.47)\ 10^{-4}$	1000 ± 0	29 ± 8
ALS	$(4.32 \pm 0.52)\ 10^{-5}$	$\mathbf{76.5 \pm 15}$	41 ± 10
QALS	$(5.35 \pm 0.20)\ 10^{-9}$	410 ± 17	67 ± 7
r1-QALS	$(2.11 \pm 1.17)\ 10^{-7}$	1000 ± 244	51 ± 11
r4-HQALS	$(4.84 \pm 0.38)\ 10^{-9}$	464 ± 53	68 ± 7
r7-HQALS	$(4.22 \pm 0.59)\ 10^{-9}$	356 ± 18	$\mathbf{68 \pm 7}$
r10-HQALS	$\mathbf{(4.87 \pm 0.79)\ 10^{-9}}$	$\mathbf{331 \pm 21}$	$\mathbf{68 \pm 7}$



(a) 3D tensor (b) 5D tensor

Figure 6.10: Convergence of NTF algorithms for experiments in Section 6.2.1.

6.2.2 Structured Factors

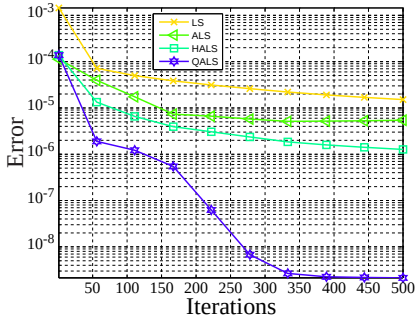
We considered factors $\mathbf{A}^{(n)} \in \mathbb{R}^{100\times 50}, n = 1, 2, 3$ whose components are the first $R = 50$ columns of 100×100 structured matrices such as the Circulant matrix, the Frank matrix, the Helmert matrix, the Hilbert matrix, the Lehmer matrix, the Lotkin matrix[69;87;88;122].

Results of factorizations of such tensors are numbered consecutively from experiments 5 to 10 as given in Table 6.8. Convergence of algorithms for some experiments is illustrated as function of iterations in Figures 6.11(e)-6.11(h). For all the experiments, the QALS always factorized the data tensors well, and converged quicker than all the other algorithms. In order to compare the original factors with those estimated by NTF algorithms, we visualized them as 2-D matrices in Figure 6.12. Figure 6.12(a) shows the circulant factors. The corresponding factors estimated by QALS and shown in Figure 6.12(b) are similar to the source ones, with the averaged SIR = 21.28 dB. The factors estimated by HALS shown in Figure 6.12(c) cannot reveal the circulant matrix structure. The distribution of SIR indices for this experiment is shown in Figure 6.12(d).

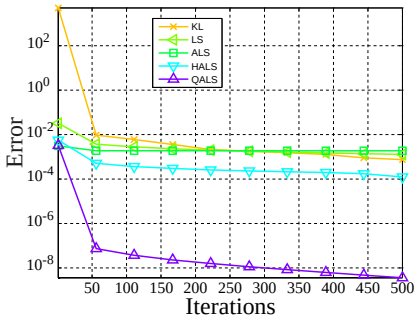
Another visualization for the Helmert factors is shown in Figures 6.12(e)-6.12(h). The verification results here strongly confirm that the proposed algorithm do indeed provide significant improvement

Table 6.8: Performance comparison for various algorithms. The relative errors are expressed in logarithmic scale, SIR indices are in dB unit, and accuracies are given in percentage. A successful algorithm is with low relative error and high SIR index. For experiments 5-10, the numbers of iterations are in side brackets.

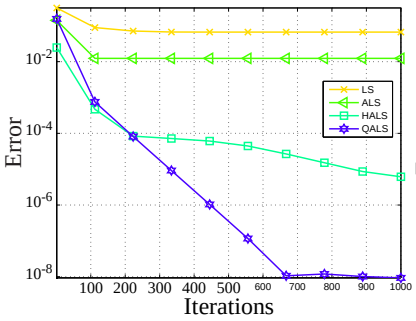
Description	Measure	mKL	mLS	ALS	HALS	QALS	fLM_+
1a. $N = 3, I_n = 100, R = 10,$	Error	(1.56 ± 0.39)e-3	(1.02 ± 0.17)e-4	(7.83 ± 0.62)e-6	(10.5 ± 0.83)e-6	(15.0 ± 7.26)e-7	**(2.87 ± 20.2)e-9**
200 iters., dense, collinear	SIR	5 ± 5	14 ± 2	18 ± 4	14 ± 3	20 ± 2	**97 ± 16**
1a*. (100000 iterations)	SIR*	13 ± 3	33 ± 6	63 ± 37	57 ± 5	140 ± 73	**150 ± 23**
1b. $N = 3, I_n = 100, R = 20,$	Error		(7.67 ± 9.61)e-7		(5.83 ± 7.36)e-7		
25.000 iters., dense, collinear	SIR		45.13 ± 10.65		38.36 ± 4.52		
2a. $N = 3, I_n = 100, R = 20,$	Error	(7.44 ± 0.28)e-4	(11.9 ± 0.64)e-4	(19.5 ± 0.87)e-4	(14.3 ± 0.71)e-5	**(3.37 ± 0.51)e-9**	(2.77 ± 12.4)e-7
500 iters., sparse, collinear	SIR	24.53 ± 11.09	10.91 ± 2.83	8.31 ± 1.72	17.84 ± 4.09	58.45 ± 5.43	**125.25 ± 28.35**
2b. $N = 3, I_n = 100, R = 20,$	Error		(9.91 ± 0.14)e-7		(6.95 ± 0.87)e-7		
25.000 iters., sparse, collinear	SIR		44.46 ± 10.91		37.51 ± 4.77		
3. $N = 5, I_n = 100, R = 10,$	Error		(6.69 ± 0.54)e-2	(1.23 ± 0.07)e-2	(5.81 ± 20)e-6	(0.89 ± 1.43)e-8	**(0.23 ± 0.52)e-9**
1000 iters., sparse, collinear	SIR		28.15 ± 7.5	35.11 ± 9.08	50.81 ± 11.56	**204.9 ± 8.80**	142.76 ± 23.07
4. $N = 3, I_n = 1.000, R = 100,$	Error		(7.77 ± 0.54)e-4	(6.45 ± 0.07)e-4	(6.34 ± 20)e-5	(3.84 ± 1.43)e-9	**3.95e-12 (54)**
500 iters., sparse, collinear	SIR		7.66 ± 1.09	6.96 ± 1.29	11.91 ± 2.04	67.43 ± 4.25	**105.25 ± 5.98**
5. Circulant matrix	Error		1.85e-4 (500)	2.27e-5 (500)	1.50e-5 (500)	2.12e-8 (207)	**6.83e-10 (139)**
$N = 3, I_n = 100, R = 50$	SIR		10.59 ± 1.22	15.61 ± 2.27	14.19 ± 2.53	21.28 ± 7.57	**40.30 ± 8.08**
6. Frank matrix	Error		1.62e-3 (500)	4.92e-4 (500)	3.38e-4 (500)	5.37e-5 (500)	**5.37e-6 (205)**
$N = 3, I_n = 100, R = 50$	SIR		10.86 ± 1.95	12.86 ± 3.02	12.52 ± 3.61	21.74 ± 15.64	**95.47 ± 51.20**
7. Helmert matrix	Error		9.1e-5(500)	1.5e-5(500)	2.92e-5(500)	6.11e-7(500)	**1.99e-7 (356)**
$N = 3, I_n = 100, R = 50$	SIR		16.36 ± 10.40	17.99 ± 8.11	17.34 ± 10.10	30.39 ± 15.98	**94.57 ± 34.80**
8. Hilbert matrix	Error		6.13e-5 (500)	1 (500)	2.5e-7(500)	7.92e-9 (47)	**4.1e-12 (40)**
$N = 3, I_n = 100, R = 50$	SIR		11.92 ± 2.06	0	20.45 ± 5.64	**38.81 ± 6.84**	25.78 ± 5.68
9. Lehmer matrix	Error		7.1e-4 (500)		1.49e-5 (500)	8.07e-10 (500)	**2.91e-10 (63)**
$N = 3, I_n = 100, R = 50$	SIR		11.75 ± 2.35		10.23 ± 2.66	**25.25 ± 9.04**	**22.41 ± 3.57**
10. Lotkin matrix	Error		3.89e-6 (500)		3.38e-4 (87)	5.37e-5 (44)	**1.86e-10 (30)**
$N = 3, I_n = 100, R = 50$	SIR		22.87 ± 2.53		27.77 ± 5.8	**32.63 ± 8.06**	31.11 ± 8.35



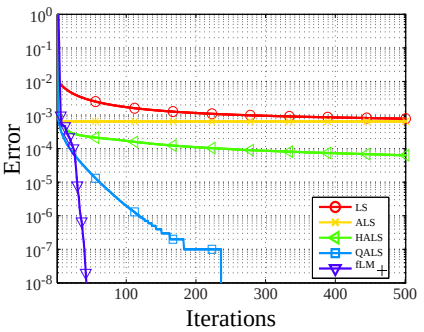
(a) Relative errors for Experiment 1.



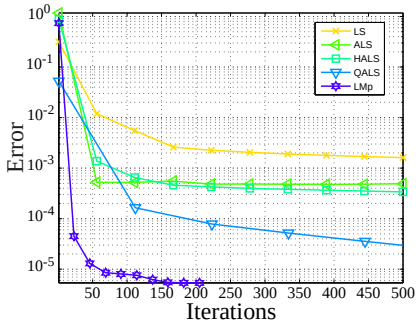
(b) Relative errors for Experiment 2.



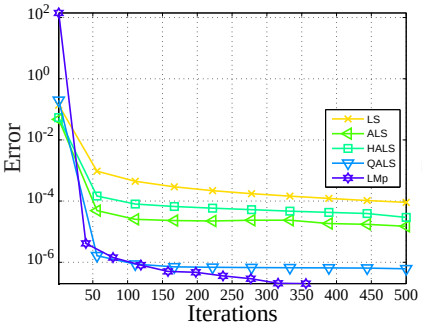
(c) Relative errors for Experiment 3.



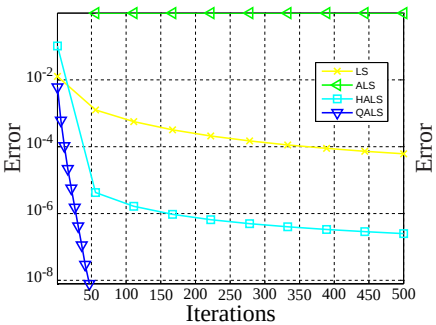
(d) Relative errors for Experiment 4.



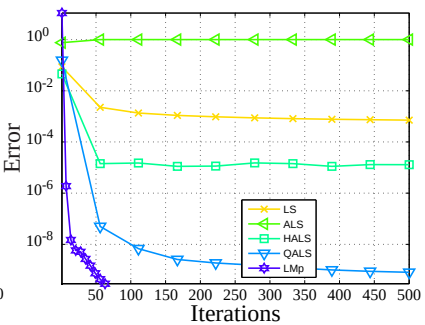
(e) Relative errors for Experiment 6.



(f) Relative errors for Experiment 7.

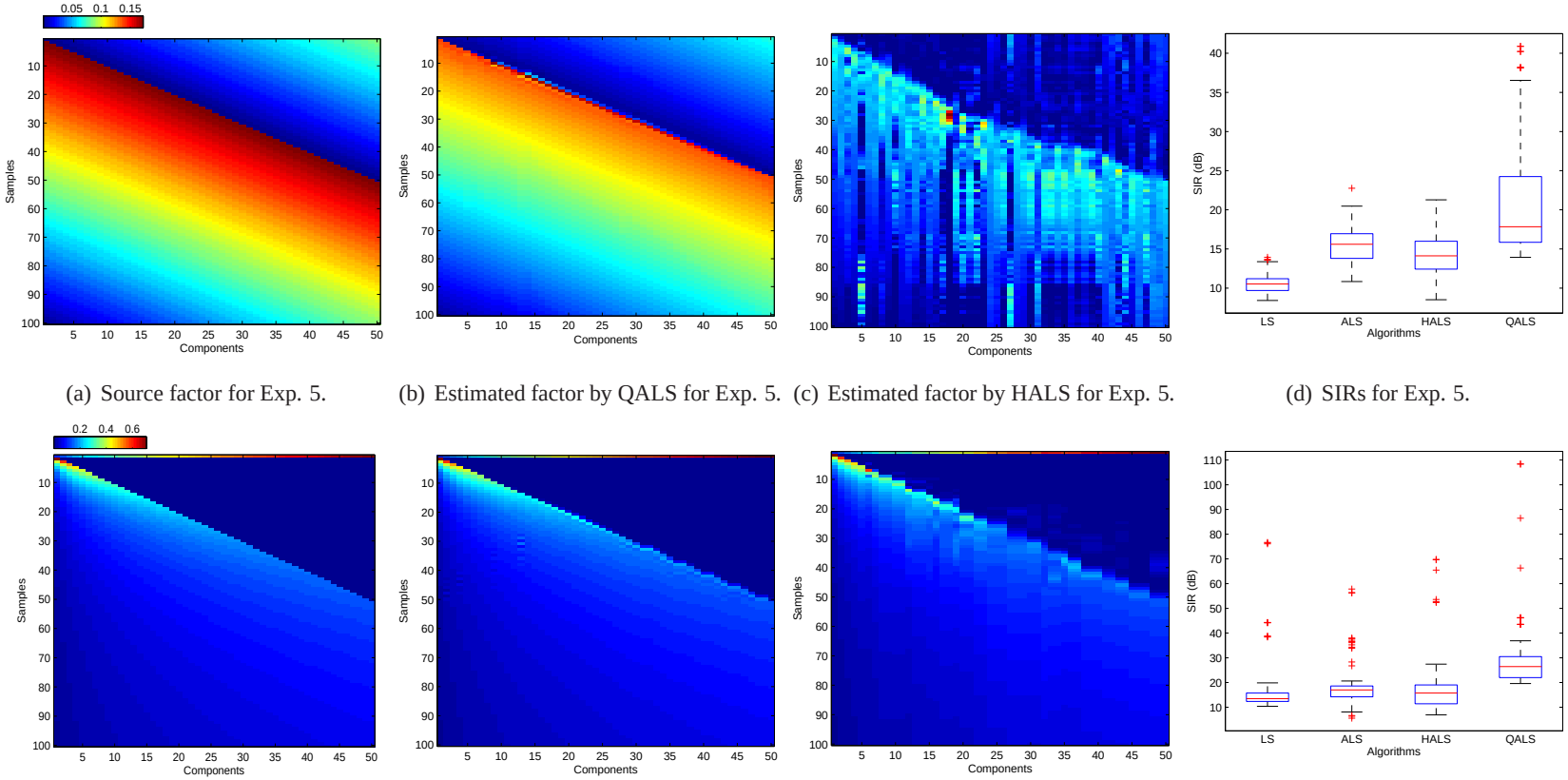


(g) Relative errors for Experiment 8.



(h) Relative errors for Experiment 9.

Figure 6.11: Performance comparison of NTF algorithms for factorization of synthetic tensors.



(a) Source factor for Exp. 5. (b) Estimated factor by QALS for Exp. 5. (c) Estimated factor by HALS for Exp. 5. (d) SIRs for Exp. 5.

(e) Source factor for Exp. 7. (f) Estimated factor by QALS for Exp. 7. (g) Estimated factor by HALS for Exp. 7. (h) SIRs for Exp. 7.

Figure 6.12: Illustration of the source and estimated factors by NTF algorithms and theirs SIR indices for Experiments 5 and 7.

for nonnegative tensor factorizations in the terms of performance and convergence compared to the other algorithms.

6.2.3 Analysis of Number of Recursive Iterations in QALS Algorithms

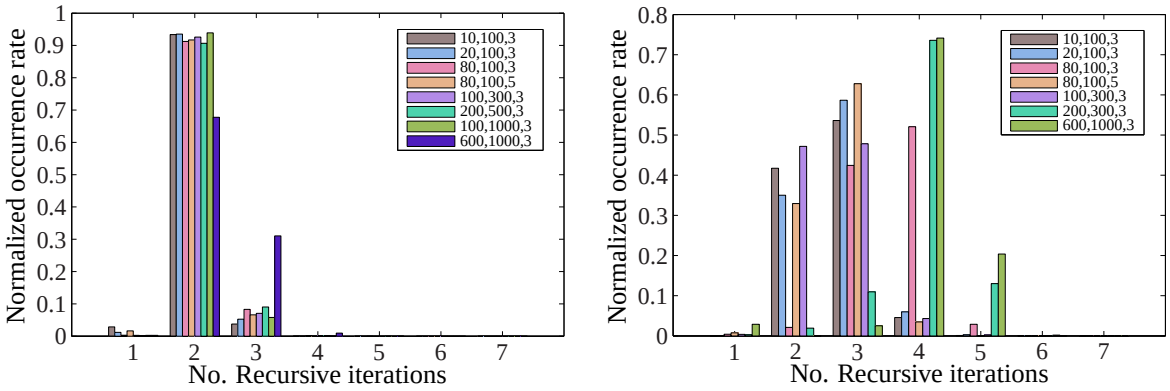
This sections aims to show upper bound for the number of recursive iterations in NQP algorithms used in Algorithms 3.2 and 3.3. Experiments are similar to those in Section 6.2.1. However, the number of recursive iterations and dimensions of matrices $\mathbf{Q}$ in the NQP function were measured. We composed 3-D and 5-D synthetic tensors from random factors which can be dense, or sparse and collinear with $I_n = 100, 300, 500$ or 1000 and R= 10, 20, 80, 100, 200 or 600 components. Figure 6.13 illustrates occurrence rates for the number of recursive iterations in the NQP function (Algorithm 3.1) for various tensor factorizations. Description of tensor factorizations appearing in legend are given in the order: R, I_n, N. The occurrence rates are normalized in [0 1] and also give in Table 6.9. The results show that NQP function repeated only few iterations even for large-scale tensors such as $R = 600$ and $I_n = 1000$. The number of recursive iterations did not exceed $(\log_2(R)+1)$. Dense tensors were quickly factorized with fewer iterations than sparse and collinear tensors. Statistical results also reveal the difference between ALS and QALS algorithms. That is the NQP function in QALS algorithms almost needed at least two iterations as convergence was achieved (Figure 6.13(a)), and higher iterations for collinear data (Figure 6.13(b)). For tensors with $R = 80, 200$ or 600 collinear components, QALS required 4 recursive NQP loops. Note that ALS employs the NQP function with only one iteration, but it is extremely slow for collinear data, or cannot fit such data satisfactorily due to possible trapping in local minima. It also explains why QALS outperforms ALS.

6.3 Simulations for NTD

We compared performance of the LM_+ algorithm with the multiplicative LS (mLS)[131], HALS[156;158]. Synthetic tensors $\mathcal{Y}$ with $I_n = 100$, $N = 3, 4$ were composed from uniformly distributed random factors comprising $R = 5$ or $R = 3$ components. In some experiments, factors were forced to be sparse with density of 30%. Algorithms were initialized using the HOSVD algorithm[58], and stopped when difference between consecutive relative errors $\varepsilon = \frac{\|\mathcal{Y}-\hat{\mathcal{Y}}\|_F}{\|\mathcal{Y}\|_F} \leqslant 10^{-8}$, or the maximum number of iterations was exceeded. Comparison of performances of various algorithms averaged over 100 runs is given in Table 6.10. Although the HALS algorithm achieved better performance than the multiplicative LS algorithm especially for sparse tensors, both theses algorithms could not decompose the random tensors with small error ($\approx 10^{-5}$) in 500 iterations. Their relative errors for decomposition of dense tensors were greater than 10^{-3}, and slightly better for sparse tensors. Whereas, the proposed algorithm achieved almost perfect performances with $\varepsilon \leqslant 10^{-5}$ after only few iterations, even for large-scale

Table 6.9: Occurrence rates for number of recursive NQP loops in QALS algorithms for factorization of random tensors in Section 6.2.3.

Experiment	Number of recursive loops					
R, I_n, N	1	2	3	4	5	6
Dense factors						
10, 100, 3	2.88	**93.36**	3.75	0.01		
20, 100, 3	1.18	**93.52**	5.27	0.04		
80, 100, 3	0.28	**91.23**	8.30	0.18		
80, 100, 5	1.64	**91.70**	6.59	0.07		
100, 300, 3	0.23	**92.59**	7.06	0.12		
200, 500, 3	0.16	**90.66**	9.02	0.16		
100, 1000, 3	0.26	**93.90**	5.80	0.04		
600, 1000, 3	0.26	**67.78**	**31.02**	0.93		
Sparse and collinear factors						
10, 100, 3		**41.73**	**53.61**	4.56	0.10	
20, 100, 3	0.01	**35.00**	**58.67**	6.00	0.30	0.02
80, 100, 3	0.41	2.13	**42.45**	**52.08**	2.91	0.02
80, 100, 5	0.76	**32.95**	**62.79**	3.49	0.01	
100, 300, 3	0.38	**47.17**	**47.84**	4.33	0.27	0.01
200, 500, 3	0.31	1.94	10.96	**73.58**	**13.02**	0.19
600, 1000, 3	2.90		2.52	**74.14**	**20.39**	0.04



(a) Random tensors with dense factors

(b) Random tensors with sparse and collinear tensors

Figure 6.13: Statistical analysis on number of recursive iterations in the NQP function for random tensors with different sizes I_n and N, different number of components R and structures of factors: dense or sparse, collinear factors. Description of tensor factorizations appearing in legend is given in the order: R, I_n, N. Tensors with sparse and collinear factors are harder to factorize than those with dense factors. The number of recursive loops does not exceed $(\log_2(R) + 1)$.

tensor ($I_n = 100, N = 4$). In Figure 6.14, we compared the relative errors of algorithms as functions of iterations for one run of decomposition of a $100 \times 100 \times 100$ dimensional tensor. After few iterations to seek the damping parameter μ and the regularization parameter α, LM_+ quickly explained the data tensor in ≈ 69 iterations. The mLS and HALS algorithms could not explain the benchmarks even if

Table 6.10: Performance comparison for various algorithms for decomposition of synthetic tensors.

Experiment	Error			No. iterations		
(I_n, N, R)	mLS	HALS	LM_+	mLS	HALS	LM_+
(50, 3, 5)	(1.62 ± 0.18)e-2	(1.15 ± 0.12)e-2	**(1.52 ± 6.39)e-7**	500	500	**47**
(100, 3, 5)	(1.76 ± 0.15)e-2	(1.28 ± 0.12)e-2	**(7.27 ± 10.26)e-9**	500	500	**69**
(100, 3, 5), sparse	(8.26 ± 1.21)e-2	(1.49 ± 0.86)e-2	**(1.70 ± 4.28)e-8**	500	500	**77**
(100, 4, 3)	(2.04 ± 0.23)e-2	(2.01 ± 0.83)e-3	**(1.10 ± 2.31)e-8**	5000	5000	**55**
(100, 4, 3), sparse	(1.82 ± 1.01)e-2	(0.49 ± 1.38)e-4	**(3.46 ± 9.57)e-6**	2000	218	**59**

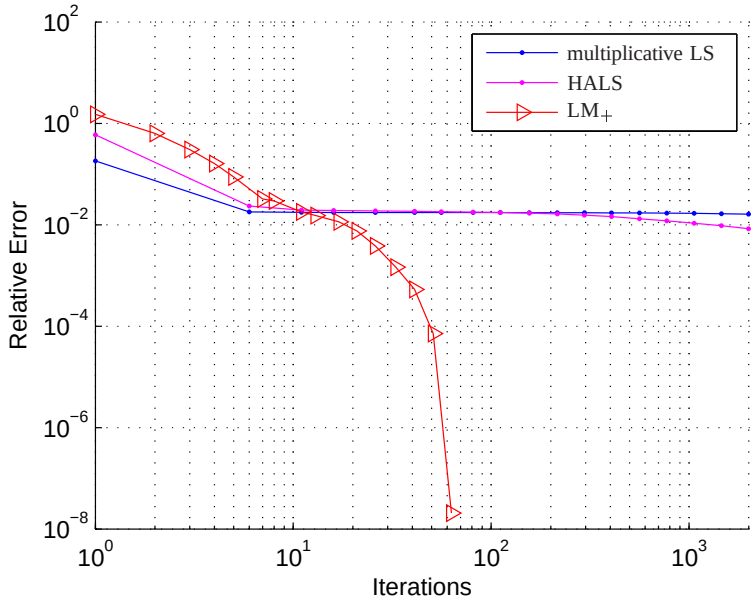


Figure 6.14: Convergence of NTD algorithms for decomposition of 3-D synthetic tensor.

they were run for 2000 or 5000 iterations.

6.4 Applications

6.4.1 Analysis, Clustering and Classification of EEG Dataset

This example illustrates the analysis of real-world EEG data[132] which consists 28 inter-trial phase coherence (ITPC) measurements[194] of EEG signals of 14 subjects during a proprioceptive pull of the left and right hands, and gives a 4-way tensor of 64 channels × 72 time frames × 61 frequency bins (15 to 75 Hz) × 28 measurements. This dataset was analyzed by CP and Tucker components with nonnegative constraints compared with components extracted by NMF and ICA[130]. For the right hand stimuli, the ITPC maxima were observed in the left frontal-parietal central region. On the contrary, for the left hand stimuli, the activities occur on the right central region. Generally, the right hand and left hand stimuli activate similar rhythms which should occur after similar latency periods but distribute over two different regions. That means factorization of such nonnegative tensor will give

collinear spectral and temporal components. In this section, we show that the proposed algorithm not only extracts highly collinear components as expected, but also outperforms the other algorithms, and achieves the highest clustering and classification accuracies.

The ITPC tensor was factorized into $R = 6$ rank-one tensors. Exemplary illustrative results are shown in Figure 6.15 with scalp topographic maps and their corresponding temporal and spectral components, whereas the performance comparisons (relative errors and number of iterations) are given in Table 6.11. The estimated components by mKL, ALS, HALS, r3-QALS and QALS are shown in Figures 6.15(a)-6.15(f). The rK-QALS resulted almost similar components with different $\tilde{R}$. For the factors estimated by QALS, the 3rd components distribute over the left frontal-parietal central region (visualized by the 3rd spatial component), reveal the gamma rhythm with the peak frequency of 37Hz (the 3rd spectral component), and with a latency of 56ms (the 3rd temporal component). The 2nd components also distribute over the same region, but present the beta rhythm between 15-25 Hz with a latency of 87ms. That means an activity caused by right hand stimulation mainly pertains beta rhythm with a latency of 87ms (component 2), and 37Hz gamma rhythm with a latency of 56ms (component 3). Similarly, an activity caused by left hand stimulation can be characterized by beta rhythm with a latency of 83ms (component 4), and 42Hz gamma rhythm with a latency of 56ms (component 5). Temporal components 3 and 5, 2 and 4 are respectively almost identical. Moreover, spectral components 2 and 4 are collinear, spectral components 3 and 5 are shifted in frequency. The other algorithms (mLS, mKL and ALS) did not successfully retrieve both collinear beta and gamma rhythms. The spectral and temporal components corresponding to spatial components distributing over the right or left parietal regions for the ALS are intuitively illustrated in Figure 6.15(c).

In the next step, we performed clustering for selected features corresponding to components which distribute over the right or left parietal regions. We factorized matrices of projected features $\mathbf{F} \in \mathbb{R}_{+}^{28\times 4}$ into 2 factors $\mathbf{A} \in \mathbb{R}_{+}^{28\times 2}$ and $\mathbf{X} \in \mathbb{R}_{+}^{4\times 2}$: $\mathbf{F} \approx \mathbf{A}\,\mathbf{X}^T$. A cluster label of a measurement was specified by the column index consisting of the maximum weight in the corresponding row of the matrix $\mathbf{A}$. The clustering accuracies are given in Table 6.11. Figure 6.16 illustrates scatter plots discriminative features extracted from NTF features for the ALS, HALS, mKL and QALS algorithms. The proposed algorithm achieved an accuracy of 92.86%, and there are only 2 misclassified measurements. The performances for the mKL, mLS and ALS, HALS algorithms are respectively 85.71%, 78.57% and 82.14% and 82.14%. The same performances can be obtained using the K-means clustering.

Another application for this dataset is classification. Each subject was characterized by 6 features and assigned to a label corresponding to the left or right class. The leave-one-out crossvalidation was employed to evaluate the feature extraction by NTF algorithm, and classification. We selected only two significant features based on the Fisher scores. A QDA classifier was trained for training features. Classification accuracies for algorithms are given in Table 6.11. Classification using features extracted

Table 6.11: Performance comparison for various algorithms for Example 6.4.1.

Algorithms	mKL	mLS	ALS	HALS	r3-HQALS	QALS
Error	0.412	0.376	0.378	0.370	**0.369**	**0.369**
No.Iters	100	100	100	100	**59**	61
Clustering Accuracy (%)	85.71	78.57	82.14	82.14	**92.86**	**92.86**
Classification Accuracy (%)	75.71	92.86	75	75		**96.43**

by the proposed algorithm achieved the highest accuracy of 96.43%, that means there was only one subject misclassified.

6.4.2 Clustering of the ORL Face Database

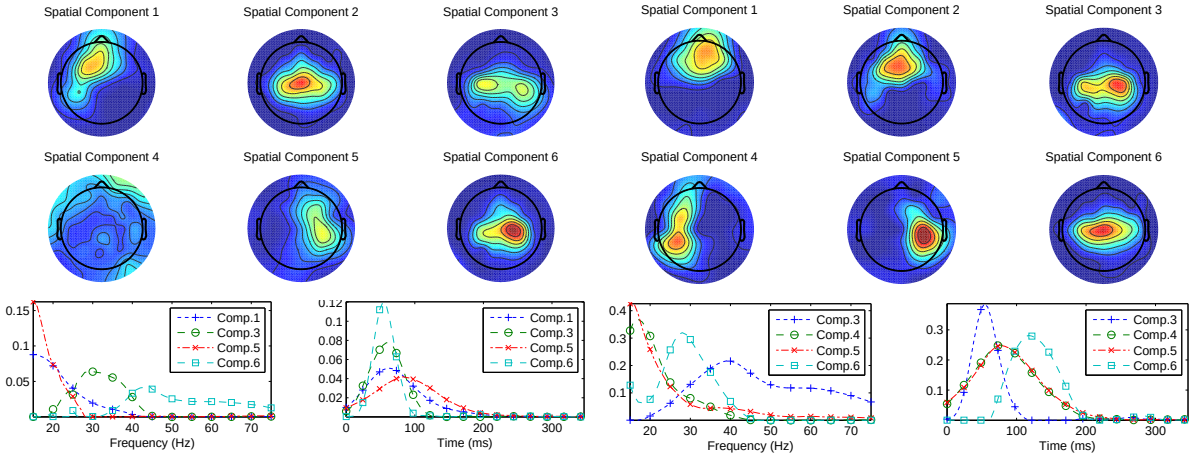
This example considers the ORL face database[178] consisting of 400 faces for 40 subjects. A common way to process this dataset is that faces are vectorized, for example in Fisherface[114], ICA[18], Wavelet + RBF[123]. In the first analysis, 100 faces from the first 10 subjects were down-sampled, then vectorized to give a (400×100) matrix $\mathbf{Y}$. We applied NMF to find $R = 20$ features for each face, and used the K-means algorithm to cluster them. The accuracy (%) and normalized mutual information (NMI) for algorithms are given in Table 6.12. The proposed algorithm explained data with a lowest relative error and achieved a higher clustering accuracy than the other algorithms. Especially, LM_+ with sparsity constraints (LM_+s) successfully clustered the selected faces.

Next, we constructed 32 Gabor feature tensors of 8 orientations at 4 scales which were then down-sampled to $16 \times 16 \times 32 \times 400$ dimensional tensor $\mathcal{Y}$. That means we have a 4-D tensor $\mathcal{Y}$. Because of low correlation or rare common parts between Gabor features which are not in the same levels (orientations and scales), we found common bases $\mathbf{A}^{(n_l)} \in \mathbb{R}^{16 \times R_l}$, $n = 1, 2$ for 3-D sub-tensors $\mathcal{Y}_l = \mathcal{Y}(:, :, l, :) \in \mathbb{R}^{16 \times 16 \times 400}$ $(l = 1, 2, \ldots, 32)$ along the two first dimensions

$$\mathcal{Y}_l \approx \mathcal{I}_l \times_1 \mathbf{A}^{(1_l)} \times_2 \mathbf{A}^{(2_l)} \times_3 \mathbf{A}^{(3_l)}. \tag{6.7}$$

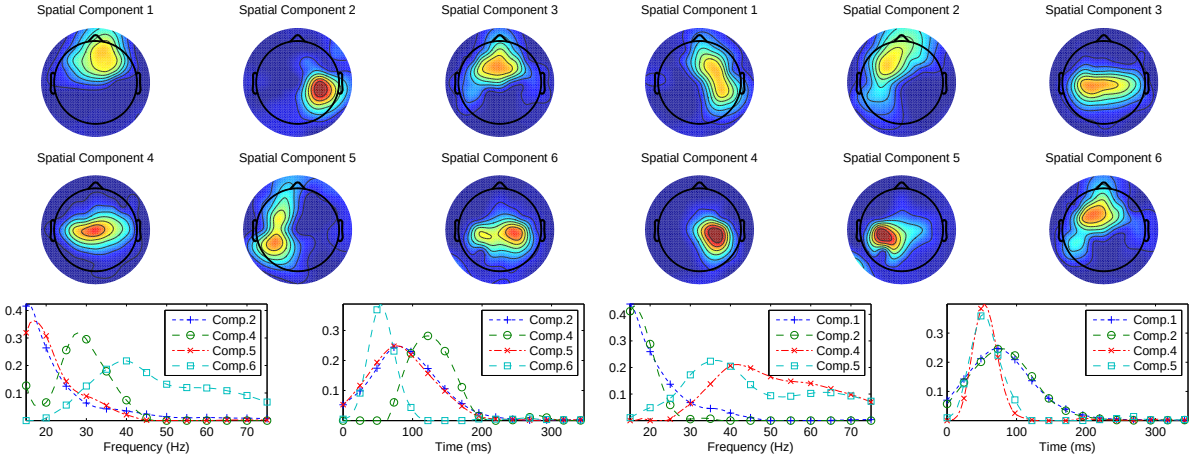
Rows of the factors $\mathbf{A}^{(3_l)}$ represent compressed features of the samples $\mathcal{Y}^{(k)} = \mathcal{Y}_{:,:,:,k}$ at level l. From $L = 32$ decompositions (6.7) for all levels, we obtained $L = 32$ sets of bases $\mathbf{A}^{(1_l)}$ and $\mathbf{A}^{(2_l)}$. Features of each sample (face) at a level $l = 1, 2, \ldots, 32$ can be found via projected filters built up from basis factors of the same level. Concatenation of features in all the levels will form the whole compressed features of a sample after tensor factorizations.

In this experiment, we set $R_l = 8, \forall l$. Hence, a sample (face) had 256 $(= 8 \times 32)$ features compressed from 8192 $(= 16 \times 16 \times 32)$ Gabor features. In the second stage, the matrix of features $\mathbf{X} \in \mathbb{R}^{400 \times 256}$ was factorized to reduce the number of features to the number of classes. Finally, the data was clustered using the K-means algorithm. In Table 6.12, we compare clustering performances for various algorithms. For clustering of faces for the first 30 subjects, LM_+ achieved 99% accuracy,



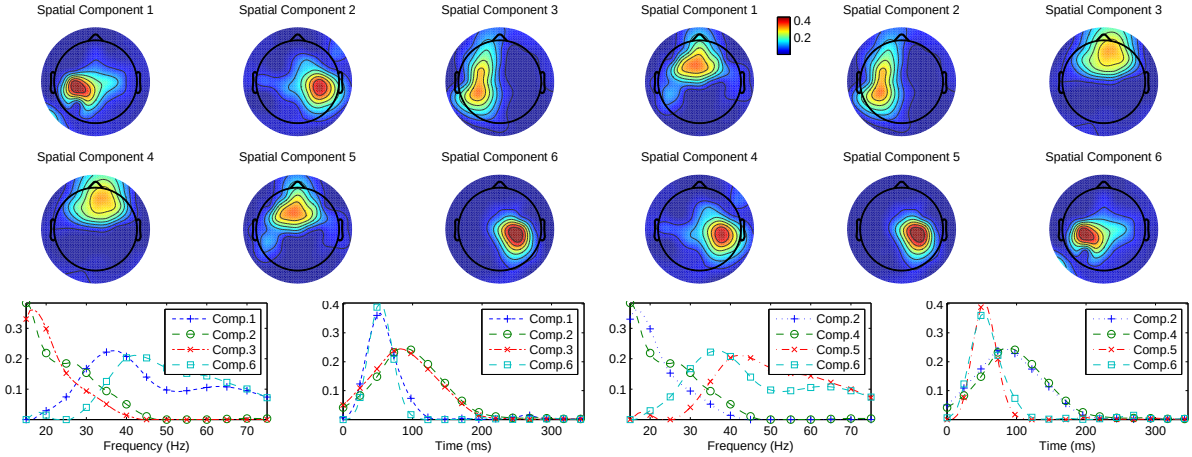
(a) Basis components obtained by the mKL algorithm

(b) Basis components obtained by the mLS algorithm



(c) Basis components obtained by the ALS algorithm

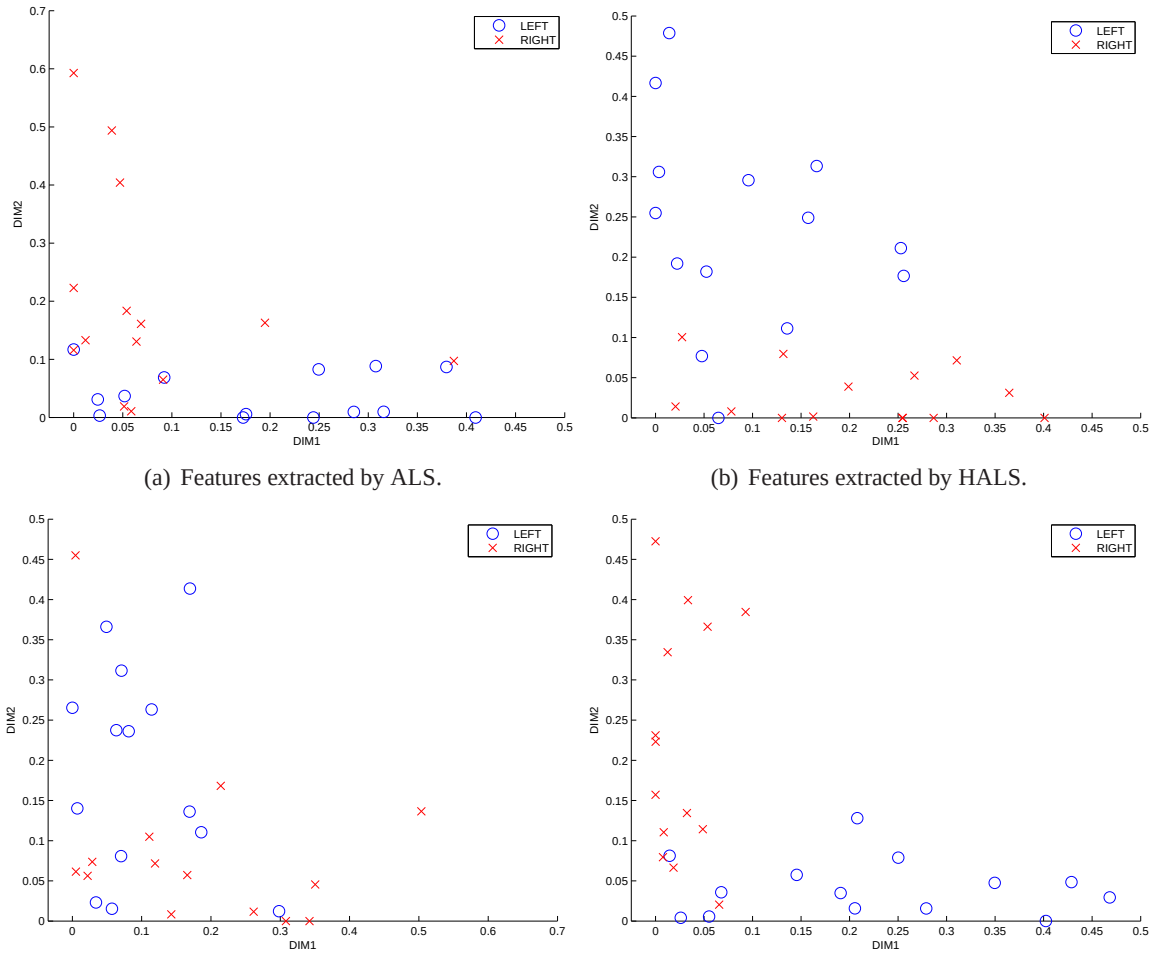
(d) Basis components obtained by the HALS algorithm



(e) Basis components obtained by the r3-QALS algorithm

(f) Basis components obtained by the QALS algorithm

Figure 6.15: Visualization of components retrieved from the ITPC tensor using the mKL, ALS, HALS, and QALS algorithms. A successful algorithm results collinear spectral and temporal components which reveal the beta and gamma rhythms as in Figure 6.15(f).



(a) Features extracted by ALS.

(b) Features extracted by HALS.

(c) Features extracted by mKL.

(d) Features extracted by QALS.

Figure 6.16: Scatter plots illustrate discriminative features for clustering for Example 6.4.1.

outperformed the other algorithms. The LM_+ algorithm with sparsity constraints (LM_+s) slightly improved performance up to an accuracy of 99.67%. Increasing the number of classes to 35 or 40 subjects, LM_+s always gave the highest performances.

We also can extract nonnegative features for faces from Gabor tensors using NTD[157]. The data tensor $\mathcal{Y}$ was reorganized to be a 5-D tensor of size $16 \times 16 \times 8 \times 4 \times 100$ (for 100 faces). The data tensor $\mathcal{Y}$ was decomposed along the first 4 modes to give core tensor's size of $3\times3\times2\times2\times100$. Hence, a face had 36 features compressed from 8192 Gabor features. Finally, the data was clustered using the K-means algorithm. The accuracy (%) and normalized mutual information (NMI) for algorithms are given in Table 6.12. The LM_+ algorithm achieved 92% accuracy. Increasing number of features to $72 = 3 \times 3 \times 4 \times 2$, our algorithm achieved 99% accuracy. For both cases, the obtained accuracies for the mLS algorithm were 91% and 98%, respectively. The presented results also confirm the superiority

Table 6.12: Accuracies (Acc) and normalized mutual information (NMI) for various algorithms for Example 6.4.2.

(a) NTF model

Algorithm	10 classes			30 classes		40 classes	
	Error	Acc (%)	NMI	Acc (%)	NMI	Acc (%)	NMI
KL	4.51e-2	80	8.64e-1	92.67	9.49e-1	82.25	9.08e-1
LS	1.27e-2	88	9.06e-1	96.33	9.75e-1	85.75	9.42e-1
ALS	6.79e-2	93	9.20e-1	95.33	9.71e-1	85.50	9.30e-1
HALS	**1.17e-2**	91	9.10e-1	97	9.76e-1	86.25	9.47e-1
QALS	1.18e-2	92	9.21e-1	98.33	9.84e-1	88	9.50e-1
fLM_+	**1.17e-2**	94	9.44e-1	99	9.90e-1	87.25	9.44e-1
fLM_+s	1.24e-2	**100**	**1**	**99.67**	**9.97e-1**	**92.75**	**9.69e-1**

(b) NTD model

Algorithm	10 classes, 36 features				10 classes, 72 features			
	Error	No. iters.	Acc (%)	NMI	Error	No.iters.	Acc (%)	NMI
mLS	0.4768	300	91	89.65	0.4430	300	98	97.09
HALS	**0.4745**	300	**92**	**91.45**	0.4369	300	96	94.76
LM_+	**0.4745**	68	**92**	**91.45**	**0.4368**	**86**	**99**	**98.54**

of features extracted by NTD over features by NTF.

6.4.3 BSS in DS-CMDA Systems

This section aims to illustrate an application of factorization of complex-valued tensors of received signals in direct sequence code division multiple access (DS-CDMA) system. Consider a DS-CDMA system of R users and K antennas over a flat Rayleigh fading, each information sequence of user r $\boldsymbol{s}_r \in \mathbb{C}^P$ is spread using a code $\boldsymbol{c}_m \in \mathbb{C}^Q$ before transmission over fading channels. At the receiver side, an array of K antennas is employed to receive and decode the signals. Sidiropoulos *et al.*[186] established the model of wireless transmission as a three-way diversity tensor $\boldsymbol{\mathcal{X}} \in \mathbb{C}^{K\times P\times Q}$ whose an entry $x_{k,p,q}$ denotes the baseband output of the k-th antenna, for symbol p and chip q

$$x_{k,p,q} = \sum_{r=1}^{R} a_{k,r}\, s_{p,r}\, c_{q,r}\,, \tag{6.8}$$

where $a_{k,r}$ fading/gain between user r and antenna element k. This model can be expressed as composition of 3 factors given by

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{I}} \times_1 \mathbf{A} \times_2 \mathbf{S} \times_3 \mathbf{C} + \boldsymbol{\mathcal{E}}, \tag{6.9}$$

where $\boldsymbol{\mathcal{E}}$ is tensor of additive Gaussian noise, $\mathbf{A} \in \mathbb{C}^{K\times R}$ denotes the compound flat fading/array response pattern, $\mathbf{S} \in \mathbb{C}^{P\times R}$ is the information bearing signal matrix, and $\mathbf{C} \in \mathbb{C}^{Q\times R}$ is the spreading code matrix[186]. Approximation of the output of antennas returns the signal matrix $\hat{\mathbf{S}}$. Then with an appropriate demodulation for each column $\hat{\boldsymbol{s}}_r$, the user information sequences will be retrieved. The compression technique using the ALS and LS algorithms was recommended to factorize complex-valued

tensors[186]. However, it's worth noting that when two or more user sequences are closely identical, or antenna responses are collinear, the ALS algorithm will fail to retrieve accurate signals.

In our experiments, user signals were modulated by M-DPSK ($M = 2, 4, 8$), then spread with Hadamard(64) codes. In order to solve the sign (scaling) and permutation ambiguities, all the user sequences were augmented by 1 sign-correction bit, and $I = [\log_2(R)]$ user ID bits. That means each user signal consisted of $(P + I + 1)$ bits. An alternative method is to use the greedy least squares matching algorithm[186]. However, this method is only useful to evaluate the performance.

The compression version of the LM algorithm was compared with algorithms cALS, cLS. In Table 6.13, we present performances for different DS-CDMA systems over 100 MC runs and different white noise levels. For DS-CMDA system with $R = 15$ users, $K = 10$ antennas, we considered a particular case in which array responses $\boldsymbol{a}_r$ were set to be collinear so that their mutual angles were about $\theta \approx 6^o$. $R = 15$ different spreading codes were randomly chosen from Hadamard(64) bases $\mathbf{C} \in \mathbb{C}^{64\times 15}$. For SNR $>$ 10 dB, we received exact user signals using the LM algorithm, with BER = 0. The ALS algorithm completely failed to decode the signals even for clean systems (without noise). The LS technique improved the ALS performance. However, its performances were rather unstable. Note that the analyzed DS-CMDA system had number of users more than number of antennas.

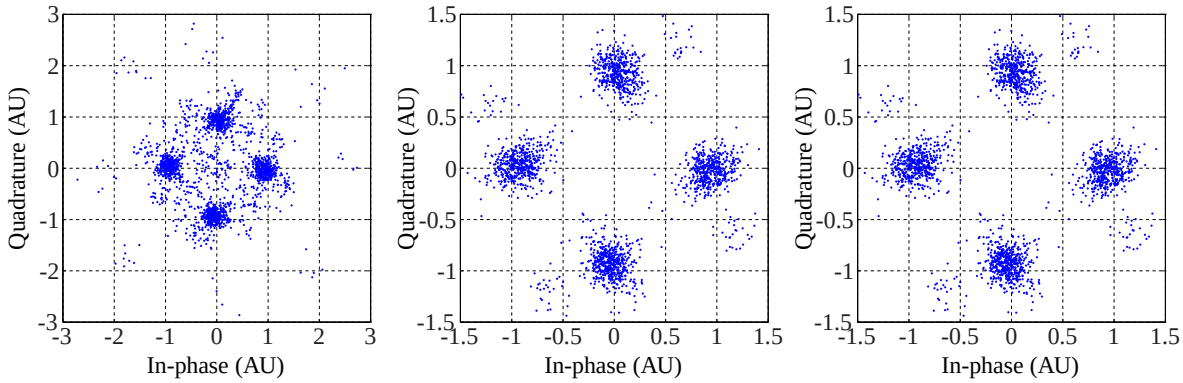
For DS-CDMA system with $K = 20$ receivers (antennas) and $R = 20$ users, user sequences were modulated by DQPSK. The performance index BER was evaluated varying the SNR at the receiver input $SNR = 0, 2, 4, \ldots, 20, 30$ dB. The cLS algorithm almost gave the same results to those of the LM algorithm. However, the LM algorithm converged after a considerable smaller number of iterations. In Figure 6.17, we illustrate the constellations at the receiver output with additive white Gaussian noise SNR = 10 dB decomposed by cALS, cLS and cLM for one MC run. Sequences estimated by cALS algorithm were still overlapped and not separated from each other (Figure 6.17(a)). The corresponding error rate was BER = 0.0747. Both cLS and cLM returned quite similar results, and achieved high performances without any error (BER = 0).

Additional results are also given in Table 6.13 for DS-CDMA system using 8-DPSK modulation. For all the experiments, the proposed LM algorithm for complex-valued tensor factorization achieved the best performance with the smallest number of iterations.

6.4.4 Estimation of System MIMO Responses Using the Fourth-Order Statistics

We considered an $N_o \times N_i$ MIMO system with $N_i = 20$ inputs and $N_o = 20$ outputs. The system output is modeled as:

$$\mathbf{X} = \sum_{l=0}^{L-1} \mathbf{H}_l \overset{l\leftarrow}{\mathbf{S}} + \mathbf{W}, \tag{6.10}$$



(a) ALS algorithm, BER = 0.0747 (b) LS algorithm, BER = 0. (c) LM algorithm, BER = 0.

Figure 6.17: Illustration of signal constellation for outputs obtained by cALS, cLS and cLM algorithms in a DS-CDMA system with $R = 20$ users, $K = 20$ antennas, and Hadamard(64) codes. User sequences were modulated using DQPSK.

Table 6.13: Performance comparison for decoding user sequences modulated by D-MPSK in DS-CDMA systems for Example 6.4.3.

		SNR (dB)				
	Algorithm	0	4	10	20	30
		$10 \times 100 \times 64$, $R = 15$ users, $\theta = 6^o$, DBPSK				
	cALS	2.47e-1 (5e-2)	6.48e-2 (6.99e-2)	4.30e-3 (1.28e-2)	8.80e-3 (2.48e-2)	3.00e-3 (1.08e-2)
BER	cLS	2.15e-1 (7e-2)	1.05e-2 (3.17e-2)	0 (0)	1.80e-3 (8.05e-3)	0 (0)
	cLM	**2.07e-1 (8e-2)**	**4.63e-3 (7.79e-3)**	**0 (0)**	**0 (0)**	**0 (0)**
	cALS	1.59e-1 (3.17e-3)	1.03e-1 (1.76e-3)	5.14e-2 (1.36e-3)	1.89e-2 (4.31e-3)	5.51e-3 (6.44e-3)
Error	cLS	1.56e-1 (2.99e-3)	1.01e-1 (1.30e-3)	5.12e-2 (7.06e-4)	1.62e-2 (3.74e-4)	5.13e-3 (6.52e-5)
	cLM	**1.56e-1 (2.98e-3)**	**1.01e-1 (1.34e-3)**	**5.12e-2 (7.06e-4)**	**1.62e-2 (1.97e-4)**	**5.13e-3 (6.52e-5)**
	cALS	2000 (0)	2000 (89)	2000 (182)	2000 (0)	2000 (106)
Iteration	cLS	2000 (0)	675 (469)	192 (87)	170 (439)	164 (251)
	cLM	**895 (394)**	**180 (198)**	**75 (21)**	**73 (24)**	**66 (38)**
		$20 \times 100 \times 64$, $R = 20$ users, $\theta = 6^o$, DQPSK				
	cALS	4.57e-1 (1e-2)	2.25e-1 (1.87e-1)	5.83e-2 (1.18e-1)	1.29e-2 (3.78e-2)	2.41e-2 (5.37e-2)
BER	cLS	4.08e-1 (1e-1)	8.71e-2 (1.11e-1)	**1.25e-5 (5.59e-5)**	**0 (0)**	**0 (0)**
	cLM	**4.00e-1 (1e-1)**	**8.29e-2 (1.08e-1)**	**1.25e-5 (5.59e-5)**	**0 (0)**	**0 (0)**
	cALS	2.27e-1 (2.85e-3)	1.45e-1 (1.38e-3)	7.34e-2 (1.15e-3)	2.32e-2 (1.20e-3)	7.38e-3 (4.06e-3)
Error	cLS	**2.24e-1 (2.29e-3)**	**1.44e-1 (9.87e-4)**	**7.29e-2 (5.57e-4)**	**2.31e-2 (1.73e-4)**	**7.31e-3 (4.73e-5)**
	cLM	**2.24e-1 (2.33e-3)**	**1.44e-1 (9.87e-4)**	**7.29e-2 (5.57e-4)**	**2.31e-2 (1.73e-4)**	**7.31e-3 (4.73e-5)**
	cALS	2000 (0)	2000 (0)	2000 (21)	2000 (13)	2000 (0)
Iteration	cLS	2000 (22)	340 (212)	153 (103)	141 (94)	148 (233)
	cLM	**2000 (515)**	**176 (98)**	**65 (13)**	**46 (12)**	**50 (29)**
		$20 \times 100 \times 64$, $R = 20$ users, $\theta = 6^o$, 8-DPSK				
	cALS	4.57e-1 (2e-2)	3.56e-1 (1.42e-1)	1.25e-1 (1.49e-1)	3.73e-2 (7.42e-2)	1.00e-1 (1.60e-1)
BER	cLS	4.47e-1 (3e-2)	2.77e-1 (1.43e-1)	**1.63e-2 (1.71e-2)**	4.61e-3 (2.06e-2)	**0 (0)**
	cLM	**4.46e-1 (3e-2)**	**2.60e-1 (1.42e-1)**	**1.63e-2 (1.71e-2)**	**0 (0)**	**0 (0)**
	cALS	1.60e-1 (3.31e-3)	1.03e-1 (1.90e-3)	5.22e-2 (2.05e-3)	1.64e-2 (4.55e-3)	5.20e-3 (9.64e-3)
Error	cLS	1.57e-1 (3.43e-3)	1.01e-1 (1.61e-3)	5.11e-2 (5.42e-4)	1.62e-2 (9.65e-4)	**5.14e-3 (6.26e-5)**
	cLM	**1.56e-1 (3.60e-3)**	**1.01e-1 (1.59e-3)**	**5.11e-2 (5.42e-4)**	**1.62e-2 (1.41e-4)**	**5.14e-3 (6.26e-5)**
	cALS	2000 (0)	2000 (1.55e+2)	2000 (1.16e+1)	2000 (8.94e-1)	2000 (2.73e+1)
Iteration	cLS	2000 (492)	519 (708)	177 (131)	173 (418)	172 (143)
	cLM	**1118 (569)**	**167 (131)**	**83 (29)**	**73 (16)**	**58 (21)**

where $\mathbf{H}_l \in \mathbb{C}^{N_o \times N_i}$, for $l = 0, 1, \ldots, L-1$ is the MIMO system impulse response matrix, $\mathbf{S} \in \mathbb{C}^{N_i \times T}$ contains the input signals, $\mathbf{X} \in \mathbb{C}^{N_i \times T}$ is a given output data matrix, $\overleftarrow{\mathbf{S}}^{l}$ denotes the l positions (columns) shifting operator to the left, with the columns shifted in from outside the matrix set to zero, $\overleftarrow{\mathbf{S}}^{0} = \mathbf{S}$. $\mathbf{W}$ is the observation noise.

The inputs $\boldsymbol{s}_j$, $j = 1, 2, \ldots, N_i$ were taken to be i.i.d. BPSK signals. The channel length was $L = 20$. The purpose is to estimate the $\tilde{\mathbf{H}}_{i,j,:}$ $(N_o \times N_i)$ be the N_f-point Discrete Fourier Transform (DFT) of $\mathbf{H}_{i,j,:}$, with $N_f = 128$, then recover the impulse responses.

We computed the discretized 4-th order cross-spectrums, defined as the $K-1$ dimensional DFT of the 4th-order cross cumulants. Concatenation of all the cross-spectrum tensors formed a 5-D tensor $\tilde{\boldsymbol{C}} \in \mathbb{C}^{N_o \times No \times N_o \times N_o \times N_f}$. Factorization of this tensor with N_o components allow to retrieve the system frequency response $\tilde{\boldsymbol{\mathcal{H}}}$

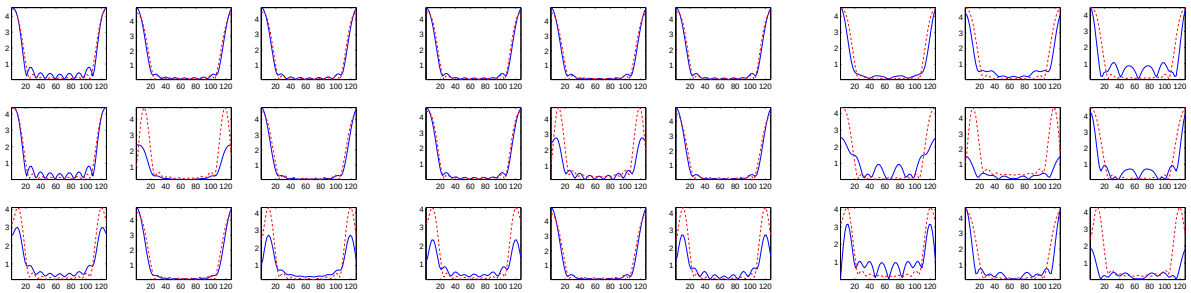
$$\tilde{\boldsymbol{C}} \approx \boldsymbol{\mathcal{I}} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \mathbf{A}^{(3)} \times_4 \mathbf{A}^{(4)} \times_5 \mathbf{A}^{(5)} \tag{6.11}$$

with the condition $\mathbf{A}^{(1)} = \mathbf{A}^{(2)*} = \mathbf{A}^{(3)}$. The tensor of frequency responses $\tilde{\boldsymbol{\mathcal{H}}}$ is computed as

$$\tilde{\boldsymbol{\mathcal{H}}} = \boldsymbol{\mathcal{I}} \times_1 \mathbf{A}^{(1)} \times_3 \mathbf{A}^{(5)}. \tag{6.12}$$

Yu and Petropulu[214] suggested to factorize only one of cross-spectrum tensors by the 4-D CP model. Hence, for large scale systems with large numbers of inputs and outputs, this technique cannot provide a good solution. Experiments were analyzed by Yu and Petropulu[214] for very small MIMO system with few numbers of outputs and inputs such as 2, 3, 4. In this section, we will emphasize the grid CP for such kind of application with large system. The performance index used here is the overall normalized mean-square error (ONMSE)[214]. For $N_o = 20$, and $N_f = 128$, the observed tensor consisted of 20.48 millions of entries. Factorization of the whole tensor to find 5 factors took **59665** seconds, and achieved an ONMSE = **0.1810**. We applied the grid CP with a grid of $128 \times 1 \times 1 \times 1 \times 1$. That means all the cross-spectrum tensors were independently factorized in a parallel system. The reconstruction factors took placed only 247 seconds, and achieved a performance of ONMSE = 0.1792. Although Yu and Petropulu' s method[214] processed the data only in 346 seconds, its estimated impulse responses were distorted from the original responses as illustrated in Figure 6.18(c) and 6.18(f). This method provided an ONMSE = 0.4485. The grid CP significantly outperformed the other methods.

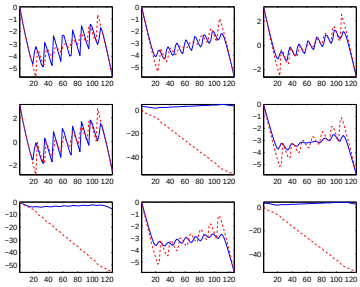
In Figure 6.18, we displayed some selected responses for some first inputs and outputs. The phase, constant permutation and scalar ambiguities were corrected to match with the original responses. In each plot, the original magnitude or phase responses were represented by dot lines, the estimated responses were shown by dashed lines.



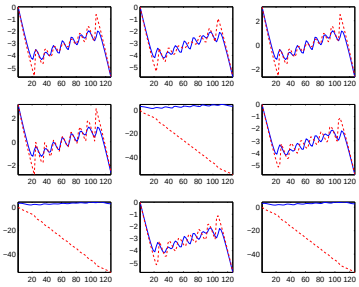
(a) Magnitude response obtained by 5-D CP

(b) Magnitude response obtained by the grid CP

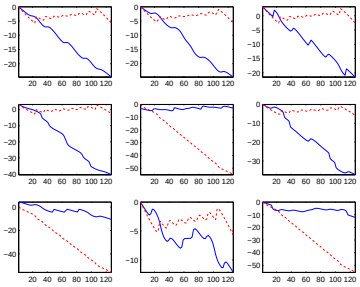
(c) Magnitude response obtained by Yu and Petropulu's method[214]



(d) Phase response obtained by 5-D CP



(e) Phase response obtained by the grid CP



(f) Phase response obtained by Yu and Petropulu's method[214]

Figure 6.18: Illustration of frequency responses of the 20×20 MIMO system for Example 6. (a)-(c) magnitude responses of the original and estimated impulse functions obtained the 5-D CP, the grid CP and Yu and Petropulu's method[214]. (d)-(f) phase responses of the original and estimated impulse functions for different methods. For each plot, the x-axis denotes the frequency points, and the y-axis represents the intensity. We displayed only some responses for the first three inputs and outputs.

6.5 Summary

Novel algorithms for tensor decompositions with/without constraints have been proposed, and verified for difficult benchmarks, and real-world applications. Especially, the proposed algorithm copes with highly collinear factors. For general data, the HALS algorithm can give satisfied results compatible with ALS but it should be faster due to low computational cost. HALS faces the same problem for collinear data as ALS. Although linesearch or rotation methods for ALS can be employed for HALS, its performance is often lower than that of ALS. The proposed fast dGN/LM algorithms works well for all the dataset. For tensor decompositions with nonnegative constraints, QALS based on the recursive algorithm for the nonnegative quadratic programming problem is proved to be a robust ALS algorithm which can work well for collinear and ill-conditioned factors. A variation of QALS is rK-QALS which sequentially updates a subset of $1 \leqslant \tilde{R} \leqslant R$ components of factors. For $\tilde{R} = 1$, the rK-QALS algorithm simplifies to the HALS algorithm[37]. The rK-QALS algorithm has been experimentally confirmed its validity and high performance for difficult data, and real-world EEG dataset. Adaptive choice of the

number of components being updated is a possible future work. Our algorithms not only works well for dense data, but also for sparse data without any additional regularization parameter as other NTF algorithms. The fLM$_+$ algorithm outperforms the multiplicative and (H)ALS algorithms, and even the QALS algorithm.

A6.1 Appendix: Effects of Noise on Collinear Data

This section discusses briefly effects of noise on factorization of collinear tensor generated by the modification (6.3). Consider matrix factorization of the mode-n tensor unfolding

$$\mathbf{Y}_{(n)} = \mathbf{A}^{(n)} \left(\bigodot_{k \neq n} \mathbf{A}^{(k)} \right)^T + \mathbf{E}_{(n)}, \tag{6.13}$$

leading left singular components of $\mathbf{Y}_{(n)}$ are good initialization for $\mathbf{A}^{(n)}$ [59;60;106]. Moreover, analysis of singular values of $\mathbf{Y}_{(n)}$ or eigenvalues of $\mathbf{Y}_{(n)}\,\mathbf{Y}_{(n)}^T$ allow predicting whether factorization succeeds in retrieving collinear factors from noisy tensors. This also gives insight into when CP algorithms are not stable, and yield non-unique solution.

The modification (6.3) can be expressed as $\mathbf{A}^{(n)} = \mathbf{U}^{(n)}\,\mathbf{Q}$, where $\mathbf{Q} = \begin{bmatrix} 1 & \mathbf{1}_{R-1}^T \\ \mathbf{0}_{R-1} & \nu\,\mathbf{I}_{R-1} \end{bmatrix} \in \mathbb{R}^{R\times R}$. In theory, for noisy tensors $\boldsymbol{\mathcal{Y}}$ with $I_n = I, \forall n$, we have

$$\mathbf{Y}_{(n)}\,\mathbf{Y}_{(n)}^T = \mathbf{A}^{(n)}\,\boldsymbol{\Gamma}^{(n,n)}\,\mathbf{A}^{(n)\,T} + \mathbf{E}_{(n)}\,\mathbf{E}_{(n)}^T = \mathbf{U}^{(n)}\,\boldsymbol{\Sigma}\,\mathbf{U}^{(n)\,T} + \sigma^2\,I^{N-1}\,\mathbf{I}_{I_n}\,. \tag{6.14}$$

where $\boldsymbol{\Sigma} = \mathbf{Q}\,\left(\mathbf{Q}^T\,\mathbf{Q}\right)^{\bullet[N-1]}\,\mathbf{Q}^T$, $[\mathbf{A}]^{\bullet[p]}$ denotes element-wise power, and

$$\sigma^2 = \frac{\|\boldsymbol{\mathcal{Y}}\|_F^2}{10^{\mathrm{SNR}/10}\,I^N} = \frac{R^2 + (R-1)\,xy - 1}{10^{\mathrm{SNR}/10}\,I^N}, \qquad x = 1+\nu^2, y = x^{N-1}. \tag{6.15}$$

It is straightforward to prove that $\boldsymbol{\Sigma} = \begin{bmatrix} R^2 + (R-1)\,(y-1) & \nu\,(R+y-1)\,\mathbf{1}_{R-1}^T \\ \nu\,(R+y-1)\,\mathbf{1}_{R-1} & (x-1)\,\left(\mathbf{1}_{R-1}\,\mathbf{1}_{R-1}^T + (y-1)\mathbf{I}_{R-1}\right) \end{bmatrix}$ has $(R-2)$ identical eigenvalues $\lambda_r = (x-1)(y-1), r = 2,\ldots,R-1$, and its largest and smallest eigenvalues $\lambda_1 > \lambda_r > \lambda_R$ are solutions of a quadratic equation

$$\lambda_1 + \lambda_R = xy + (R-2)\,(R+x+y) + 3, \tag{6.16}$$

$$\lambda_1\,\lambda_R = (x-1)(y-1) = \lambda_r\,, \qquad 2 \leqslant r \leqslant R-1\,. \tag{6.17}$$

Figure 6.19(a) illustrates λ_r $(r = 1,\ldots,R)$ for 3-D noiseless tensors with $I = 50$ and $R = 5$ compared with the noise levels $\sigma^2\,I^{N-1}$ at SNR = 20 dB, 30 dB and 40 dB. The higher the collinearity degree of factor, the smaller the eigenvalues λ_r. If eigenvalues λ_r are considerably lower than the noise level $\sigma^2\,I^{N-1}$, the factorization becomes infeasible, e.g., as $\nu \leqslant 0.3$ at SNR = 20 dB, $\nu \leqslant 0.2$ at SNR = 30 dB.

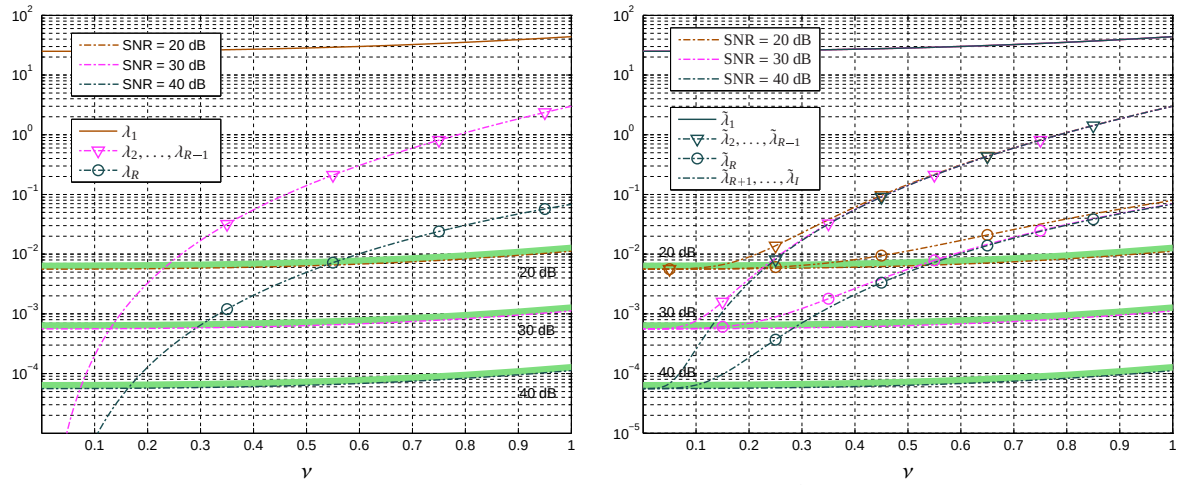
Because $\mathbf{U}^{(n)}$ are orthonormal, $\mathbf{Y}_{(n)}\,\mathbf{Y}_{(n)}^T$ has R leading eigenvalues $\tilde{\lambda}_r = \lambda_r + \sigma^2 I^{(N-1)}, r = 1,\ldots,R$, and $(I-R)$ eigenvalues $\tilde{\lambda}_i = \sigma^2 I^{(N-1)}, i = R+1,\ldots,I$. In Figure 6.19(b), we plot eigenvalues $\tilde{\lambda}_i$ for noisy tensors having the same dimension as that of tensors illustrated in Figure 6.19(a). The largest eigenvalue $\tilde{\lambda}_1$ significantly exceeds the noise levels. Whereas $\tilde{\lambda}_R$ is quite close to the noise level at SNR = 20 dB for $\nu \leqslant 0.3$, or at SNR = 30 dB for $\nu \leqslant 0.2$.

Practical simulations show that $\mathbf{E}_{(n)}\,\mathbf{E}_{(n)}^T$ is not a scale multiple of the identity matrix due to variable length I^{N-1} not large enough. However, it can be approximated as a diagonal matrix $\frac{1}{I^{N-1}}\mathbf{E}_{(n)}\,\mathbf{E}_{(n)}^T \approx \sigma^2$ diag $\{\rho_1,\ldots,\rho_I\}$, in which $\rho_i \sim N(1,\rho^2)$, and ρ depends on I^{N-1}. For 3-D tensors, $\rho \approx 0.145$, 0.01, 0.032, 0.011, 0.007 for I = 50, 100, 1000, 10000, 20000. Therefore, eigenvalues $\tilde{\lambda}_i$ ($i \geqslant R+1$) are not completely identical, they mostly fluctuate around $\sigma^2 I^{N-1}$ within a variation range of $\pm 2\rho\sigma^2 I^{N-1}$. In Figure 6.19(b), the variation range is illustrated by green shading. As a consequence, eigenvalues $\tilde{\lambda}_i$ which correspond to the signals can drop down into the noise region (green shading). In this case, leading eigenvectors might reflect noise, not the signals, and the factorization can yield undesired solutions in which some components explain noise. This also causes unstable performance when some leading eigenvalues are equal to the noise level such as $\tilde{\lambda}_R = \tilde{\lambda}_{R+1}$. Factorization of such tensor by only R components tends to give non-unique solution even if discarding the permutation and scale ambiguities. Component of factors might be different over runs, and depend on the initial values which are often eigenvectors of $\mathbf{Y}_{(n)}\,\mathbf{Y}_{(n)}^T$. We note that due to $\tilde{\lambda}_R = \tilde{\lambda}_{R+1}$, we can have multiple (at least two) selections of R leading eigenvectors. Some selections can yield components which explain noise, while some can lead to the appropriate solution.

As seen in Figure 6.19(b), at SNR = 20 dB and $\nu = 0.1$, all eigenvalues except $\tilde{\lambda}_1$ are in the noise zone. Hence, we cannot retrieve exactly all collinear components from such noisy tensors. We note that the average angular CRIB for this case is around -12.5 dB as seen in Figure 6.1(a). Even for $\nu = 0.2$, it is still hard to approach CRIB = -23 dB because λ_R is in the noise zone. At SNR = 30 dB, due to the same reason, CP algorithms often fail to estimate factors for $\nu = 0.1$ in spite of CRIB = -22 dB.

Some techniques are suggested to improve performance for such difficult data. We compute eigenvalues $\tilde{\lambda}_i$ of the mode-n tensor unfolding. If the first R eigenvalues are clearly different to the rest ones $\tilde{\lambda}_R \gg \tilde{\lambda}_{R+1}$, the factorization is feasible, we can obtain appropriate solution by using leading eigenvectors.

If $\tilde{\lambda}_R$ is approximately close to adjacent ones, $\tilde{\lambda}_{R-1} \gg \tilde{\lambda}_R \simeq \tilde{\lambda}_{R+1}$, one eigenvalue related to signal $\tilde{\lambda}_{\tilde{R}}$ is hidden under the noise level. That is $\tilde{\lambda}_{\tilde{R}} \lesssim \tilde{\lambda}_R$, with $\tilde{R} \geqslant R$. The tensor factorization becomes difficult. Approximation of the data tensor by rank-R tensors is not stable, and can yield component which reflects noise. As mentioned previously, we have multiple selections of R leading eigenvectors chosen from I components. Each initialization comprises $(R-1)$ leading eigenvectors



(a) Eigenvalues $\lambda_r, r = 1, \ldots, R\ (= 5)$, $I_n = 50$, $N = 3$.

(b) Eigenvalues $\tilde{\lambda}_i, i = 1, \ldots, I_n\ (= 50)$, $R = 5$, $N = 3$.

Figure 6.19: Analysis of eigenvalues of $\mathbf{Y}_{(n)}\,\mathbf{Y}_{(n)}^T$ for 3-D tensors of size $I_n = 50$ and rank $R = 5$. R leading eigenvalues λ_r for noiseless tensors and $\tilde{\lambda}_r (r = 1, \ldots, R)$ for noisy tensors are compared with noise levels (green shading) at SNR = 20 dB, 30 dB and 40 dB. The more the eigenvalues are in the noise zone, the more difficult the factorization of noisy tensors to retrieve collinear factors become.

and one corresponding to $\tilde{\lambda}_i$ with $i = R, R+1, \ldots, I$. There may exist one initialization (or more) which can lead to the desired solution. However we don't know which one to select. A possible method is that we initialize factors by a selection of R leading eigenvectors. If resulting factors do not satisfy prior knowledge of the solution such as the number of collinear components, we replace that last column of the initial factor by the adjacent eigenvector, then factorize data again. This process stops when factors satisfy the collinearity condition. We can also factorize data simultaneously using all or some potential selections of R leading eigenvectors. Among the solutions, we can select the suitable solution. Even if we don't know the number of collinear components, components related to noise can be straightforwardly identified, and its solution can be ignored. The feasible solutions should have the highest number of collinear components.

An alternative technique is that we approximate the data tensor by rank $\tilde{R} > R$ because the data tensor no longer has rank R. Increasing $\tilde{R}$ until a feasible solution is achieved. It is also possible to combine both of the methods.

CHAPTER 7

Applications for Feature Extraction and Classification

7.1 Introduction - Problem Formulation

Supervised and un-supervised dimensionality reduction and feature extraction methods with tensor representation have recently attracted great interest[47;106;192;195]. Given that many real-world data (e.g., brain signals, images, videos) are conveniently represented by tensors, traditional algorithms such as PCA, LDA, and ICA could treat the data as matrices or vectors[71;72;92;93;96;97;98;99;124], and are often not efficient. Since the curse of high dimensionality is often a major cause of limitation of many practical methods, dimensionality reduction is a prerequisite to practical applications in classification, data mining, vision and pattern recognitions fields.

In classification and pattern recognition problems, there are three main stages: feature extraction, feature selection, and classifier design. The key issue is to extract and select statistically significant features, which allow us to discriminate different classes or clusters. Classifier design involves choosing an appropriate method such as Fisher discriminant analysis, k-nearest neighbor (KNN) rule, or support vector machines (SVM). In a nutshell, the classifier computes distance or similarity among extracted features for training and test data in order to assign the test data to specific class.

In this chapter we propose a suite of algorithms for feature extraction and classification, especially suitable for large scale problems. In our approach, we first decompose multi-way data under the TUCKER decomposition with/without constraints to retrieve basis factors and significant features from the core tensors. In addition, by revisiting the TUCKER decomposition, we have developed family of algorithms referred to as Higher Order Discriminant Analysis (HODA). Examples in this chapter especially ones for BCI can be found in the NFEA toolbox[150].

7.2 Feature Extraction for 2-D Samples via Approximative Simultaneous Matrix Factorizations

We shall first illustrate the basic concept of feature extraction on a set of large-scale sample matrices. The problem of feature extraction for a set of 2-D training samples can be described as follows

Problem 7.1 (Feature extraction for 2-D samples illustrated in Figure 7.1(a))
Consider a set of K data matrices (2-D samples) $\mathbf{X}^{(k)} \in \mathbb{R}^{I_1 \times I_2}, (k = 1, \ldots, K)$ *that belong to C*

different data/sample classes. In order to perform model reduction and extract the features for all the training samples we apply simultaneous (approximative) matrix factorizations:

$$\mathbf{X}^{(k)} \approx \mathbf{A}^{(1)}\,\mathbf{F}^{(k)}\,\mathbf{A}^{(2)\,T}, \qquad (k = 1, 2, \ldots, K), \tag{7.1}$$

where the two common factors (basis matrices) $\mathbf{A}^{(1)} \in \mathbb{R}^{I_1 \times R_1}$ *and* $\mathbf{A}^{(2)} \in \mathbb{R}^{I_2 \times R_2}$, $R_n \leqslant I_n$ *code (explain) each sample* $\mathbf{X}^{(k)}$ *simultaneously along the horizontal and vertical dimensions. The extracted features are represented by matrices* $\mathbf{F}^{(k)} \in \mathbb{R}^{R_1 \times R_2}$, *typically with* $R_1 << I_1$ *and* $R_2 << I_2$.

The common method to solve simultaneous matrix factorizations is to minimize the cost functions $||\mathbf{X}^{(k)} - \mathbf{A}^{(1)}\,\mathbf{F}^{(k)}\,\mathbf{A}^{(2)\,T}||_F^2, \quad \forall k$ sequentially with respect to all the factor matrices. To introduce an alternative and more effective method to deal with the simultaneous matrix factorization problem, we first perform concatenation of all the samples $\mathbf{X}^{(k)}$ along the third dimension to form an $I_1 \times I_2 \times K$ dimensional data tensor $\mathcal{X}$. In other words, the frontal slices of the concatenation tensor are built up from the training matrices $\mathbf{X}^{(k)}$ (see Figure 7.1(b)). The mode-1 matricization of the concatenation tensor is expressed by the following matrix factorization:

$$\begin{aligned} \mathbf{X}_{(1)} &= \left[\begin{array}{cccc} \mathbf{X}^{(1)} & \mathbf{X}^{(2)} & \cdots & \mathbf{X}^{(K)} \end{array}\right] \approx \mathbf{A}^{(1)} \left[\begin{array}{cccc} \mathbf{F}^{(1)} & \mathbf{F}^{(2)} & \cdots & \mathbf{F}^{(K)} \end{array}\right] \left(\mathbf{I}_K \otimes \mathbf{A}^{(2)}\right)^T \\ &= \mathbf{A}^{(1)}\,\mathbf{F}_{(1)}\,\left(\mathbf{I}_K \otimes \mathbf{A}^{(2)}\right)^T, \end{aligned} \tag{7.2}$$

and similarly, for mode-2 matricization we have

$$\mathbf{X}_{(2)} \approx \mathbf{A}^{(2)} \left[\begin{array}{cccc} \mathbf{F}^{(1)\,T} & \mathbf{F}^{(2)\,T} & \cdots & \mathbf{F}^{(K)\,T} \end{array}\right] \left(\mathbf{I}_K \otimes \mathbf{A}^{(1)}\right)^T = \mathbf{A}^{(2)}\,\mathbf{F}_{(2)}\,\left(\mathbf{I}_K \otimes \mathbf{A}^{(1)}\right)^T, \tag{7.3}$$

where $\mathbf{F}_{(1)}$ and $\mathbf{F}_{(2)}$ are mode-1 and mode-2 matricized versions of the concatenation core tensor $\mathcal{F}$ comprising the feature matrices $\mathbf{F}^{(k)}$. Simultaneous matrix factorizations (7.1) can now be expressed as a decomposition of a 3-D tensor into two factors and a core tensor as the TUCKER-2 decomposition[205;206] illustrated in Figure 7.1(b)

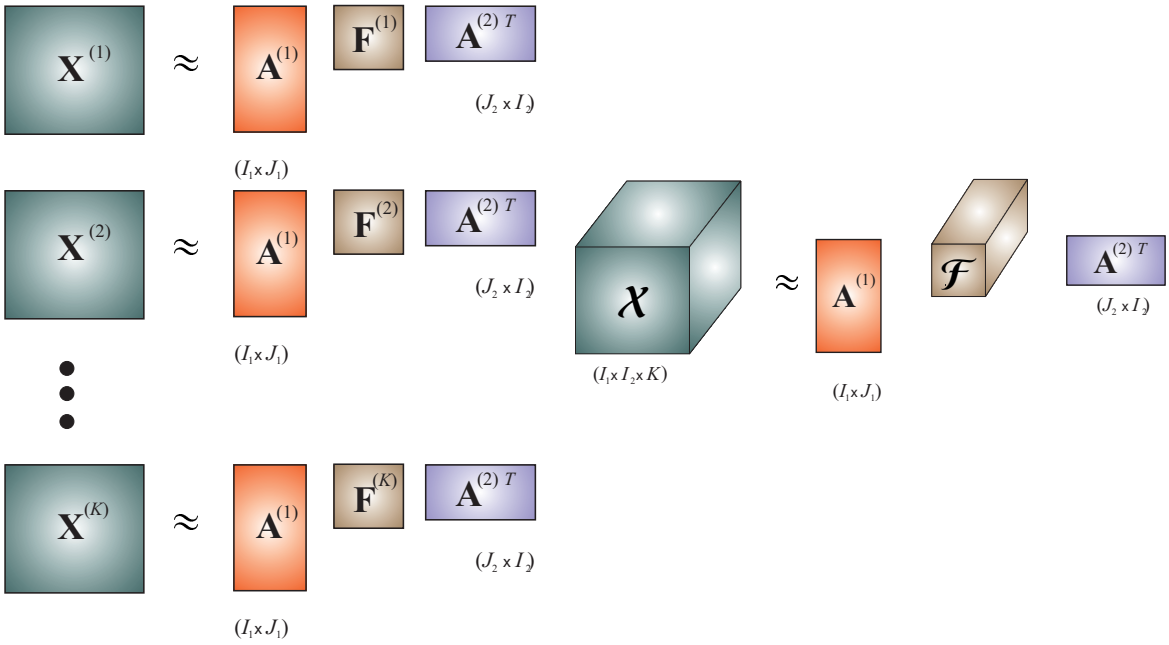
$$\mathcal{X} \approx \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)}. \tag{7.4}$$

In a particular case where $\mathbf{F}^{(k)}$ are $R \times R$ diagonal matrices, that is, $\mathbf{F}^{(k)} = \text{diag}\{\boldsymbol{f}^{(k)}\}$ (for $R_1 = R_2 = R$), the matricization of the concatenation tensor $\mathcal{X}$ is given by

$$\begin{aligned} \mathbf{X}_{(1)} &= \left[\mathbf{X}^{(1)} \cdots \mathbf{X}^{(K)}\right] \approx \mathbf{A}^{(1)} \left[\text{diag}\{\boldsymbol{f}^{(1)}\}\mathbf{A}^{(2)T} \cdots \text{diag}\{\boldsymbol{f}^{(K)}\}\mathbf{A}^{(2)T}\right] \\ &= \mathbf{A}^{(1)} \left[\left(\boldsymbol{f}^{(1)T} \odot \mathbf{A}^{(2)}\right)^T \cdots \left(\boldsymbol{f}^{(K)T} \odot \mathbf{A}^{(2)}\right)^T\right] = \mathbf{A}^{(1)} \left(\mathbf{F} \odot \mathbf{A}^{(2)}\right)^T, \end{aligned} \tag{7.5}$$

where $\mathbf{F} = \left[\boldsymbol{f}^{(1)}, \boldsymbol{f}^{(2)}, \ldots, \boldsymbol{f}^{(K)}\right]^T \in \mathbb{R}^{K \times R}$. This result enables us to rewrite Problem 7.1 as a factorization of the concatenation tensor $\mathcal{X}$ into three factors $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}$, and $\mathbf{F}$

$$\mathcal{X} \approx \mathcal{I} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \mathbf{F}, \tag{7.6}$$



(a) Simultaneous approximative matrix factorizations of 2-D samples.

(b) Equivalent 3D tensor decomposition: TUCKER-2 model.

Figure 7.1: Simultaneous matrix factorizations are equivalent to a TUCKER-2 decomposition of a 3-D tensor into a core tensor $\mathcal{F}$ (representing features) and two basis factors $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$.

where $\mathcal{I} \in \mathbb{R}^{R \times R \times R}$ is an identity tensor. The approximation (7.6) is referred to as the CP model[82]. It should be emphasized that such tensor decomposition and equivalent simultaneous matrix factorizations are quite flexible, and we can impose various constraints.

- If the feature matrices $\mathbf{F}^{(k)}$ are positive definite diagonal matrices and factors $\mathbf{A}^{(n)}$ are orthogonal, then, the model corresponds to HOSVD or multi-way PCA[58].
- If the factors $\mathbf{A}^{(n)}$ are orthogonal and the feature matrices $\mathbf{F}^{(k)}$ are dense, approximation (7.1) corresponds to a model called DEDICOM (Decomposition into Directional COMponents)[14].
- If the factors $\mathbf{A}^{(n)}$ are nonnegative, then, (7.1) corresponds Tri Nonnegative Matrix Factorization of data $\mathbf{X}^{(k)}$. Such a problem arises, for example, in bio-informatics if we combine gene expression and transcription factor regulation[12;32].
- It is important to note that if $\mathbf{X}^{(k)}$ are positive-definite covariance or cumulant matrices, Problem 7.1 becomes closely related to Joint Diagonalization often arising in ICA, where $\mathbf{A}^{(1)} = \mathbf{A}^{(2)} = \mathbf{A}$ corresponds to mixing matrix of ICA model. This leads to a new approach and algorithm for approximative Joint Diagonalization via a symmetric CP with orthogonal factors, given by

$$\mathcal{X} \approx \mathcal{I} \times_1 \mathbf{A} \times_2 \mathbf{A} \times_3 \mathbf{F}. \tag{7.7}$$

Although this model is related to the model given in (7.6) and (7.4), the topics of ICA and multi-way ICA are out of the scope of this chapter.

Note that generally, dimensionality reduction or feature extraction of set of matrices can be effectively and elegantly solved by tensor decompositions, especially using TUCKER-2, TUCKER-3, or CP models. By exploiting existing algorithms for tensor decompositions, it is relatively straightforward to retrieve common factors within the whole data. In the sequence, our aim is to generalize Problem 7.1 to make it applicable to higher dimensional data, and to develop new algorithms for finding reduced features and hidden basis factors.

7.3 General Model for High Dimensional Classification

Assume a set of multidimensional tensors $\mathcal{X}^{(k)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ for $k = 1, 2, \ldots, K$, representing training data coming from C classes. Each training sample $\mathcal{X}^{(k)}$ is given a label c_k indicating the category (class) to which it belongs. We shall formulate the following classification problem (every step will be addressed in a separate section).

Problem 7.2 (Classification for multidimensional datasets)
Consider a set of K training samples $\mathcal{X}^{(k)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}, (k = 1, 2, \ldots, K)$ *corresponding to C categories, and a set of test data* $\mathring{\mathcal{X}}^{(t)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}, (t = 1, 2, \ldots, T)$. *The challenge is to find appropriate labels for the test data. The classification paradigm can be generally performed in the following steps*

1. *Find the set of basis matrices and corresponding features for the* ***training data*** $\mathcal{X}^{(k)}$*;*
2. *Perform* ***feature extraction*** *for test samples* $\mathring{\mathcal{X}}^{(t)}$ *using the basis factors found for the training data (using a suitably designed projected filter);*
3. *Perform* ***classification*** *by comparing the test features with the training features.*

In general, a sample (object) is explained by N basis matrices $\mathbf{A}^{(n)} = [\boldsymbol{a}_1^{(n)}, \boldsymbol{a}_2^{(n)}, \ldots, \boldsymbol{a}_{R_n}^{(n)}] \in \mathbb{R}^{I_n \times R_n}, (n = 1, 2, \ldots, N)$ giving features represented by core tensors. We can assume that $\mathbf{A}^{(n)}$ contains R_n components. The relation of a sample $\mathcal{X}^{(k)}$ and N basis factors $\mathbf{A}^{(n)}$ can be expressed as

$$\mathcal{X}^{(k)} = \mathcal{G}^{(k)} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)} + \mathcal{E}, \qquad (k = 1, 2, \ldots, K), \tag{7.8}$$

where the compressed core tensor $\mathcal{G}^{(k)} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ representing features is of a much lower dimension than the raw data tensor $\mathcal{X}^{(k)}$. In other words, the reduced core tensor $\mathcal{G}^{(k)}$ consists of features of the sample $\mathcal{X}^{(k)}$ in the subspace of $\mathbf{A}^{(n)}$. Each entry $g^{(k)}_{r_1, r_2, \ldots, r_N}$ of the core tensor $\mathcal{G}^{(k)}$ is an individual feature, and expresses the strength of interaction among basis components $\boldsymbol{a}^{(1)}_{r_1}, \boldsymbol{a}^{(2)}_{r_2}, \ldots, \boldsymbol{a}^{(N)}_{r_N}$ in different factors. We call this the interactive bases.

In a particular case for the CP model, the core tensor $\mathcal{G}^{(k)}$ simplifies into a diagonal tensor. In this case, a component of factor $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$ has only one combination with components in the same order as in the other factors. We call it the non-interactive bases. Sample $\mathcal{X}^{(k)}$ is reduced to $R = R_1 = \cdots = R_N$ features which are super-diagonal entries of $\mathcal{G}^{(k)}$.

7.3.1 Estimation of Bases and Corresponding Features

Consider a training dataset containing K data samples $\mathcal{X}^{(k)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. The purpose of the first training step is to find a set of N basis factors (matrices) $\mathbf{A}^{(n)}, (n = 1, 2, \ldots, N)$ which explain the training data along their corresponding dimensions, and feature core tensors $\mathcal{G}^{(k)}$. This problem is illustrated in Figure 7.2(a), and is formulated as follows:

Problem 7.3 (Estimation of basis matrices and corresponding features)
Find N common factors $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}, (n = 1, 2, \ldots, N)$ *from K simultaneous decompositions of K sample tensors* $\mathcal{X}^{(k)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$

$$\mathcal{X}^{(k)} \approx \mathcal{G}^{(k)} \times_1 \mathbf{A}^{(1)} \cdots \times_N \mathbf{A}^{(N)}, \qquad (k = 1, 2, \ldots, K), \tag{7.9}$$

where $R_n < I_n$ *are the number of components (columns) of the factors* $\mathbf{A}^{(n)}$*, and* $\mathcal{G}^{(k)} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ *consist of features of the data tensors* $\mathcal{X}^{(k)}$.

From (7.9), it is clear that tensor decompositions perform sample reduction by projecting the tensors $\mathcal{X}^{(k)}$ to smaller dimension core tensors $\mathcal{G}^{(k)}$, where entries $g_{r_1, r_2, \ldots, r_N}$ of the core tensor $\mathcal{G}^{(k)}$ are features of the training data $\mathcal{X}^{(k)}$ in the feature space spanned by factors $\mathbf{A}^{(n)}$. In total, we have $L = R_1 \times R_2 \times \cdots \times R_N$ features which are vectorization of the core tensors $\mathcal{G}^{(k)}$.

To solve Problem 7.3, we can design cost functions for all K simultaneous decompositions (7.9); one such example is

$$\arg \min_{\{\mathbf{A}^{(1)}, \cdots, \mathbf{A}^{(N)}\}} \sum_{k=1}^{K} \| \mathcal{X}^{(k)} - \mathcal{G}^{(k)} \times_1 \mathbf{A}^{(1)} \cdots \times_N \mathbf{A}^{(N)} \|_F^2, \tag{7.10}$$

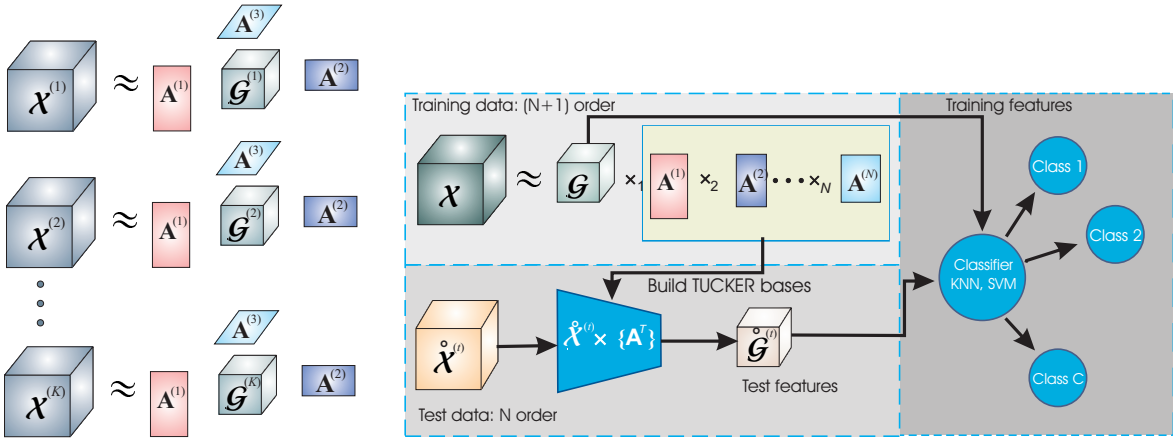
whereas, in principle, this method allows to find the common factors $\mathbf{A}^{(n)}$ and corresponding features, but it is quite complicated. We can considerably simplify Problem 7.3 by concatenating all the training data $\mathcal{X}^{(k)}$ and converting the problem into that of a single tensor decomposition, possibly with some constraints imposed on factor matrices.

Since the projection in (7.9) is a TUCKER decomposition of $\mathcal{X}^{(k)}$, its vectorized version becomes

$$\text{vec}\left(\mathcal{X}^{(k)}\right) \approx \left(\mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(2)} \otimes \mathbf{A}^{(1)}\right) \text{vec}\left(\mathcal{G}^{(k)}\right) = \left(\{\mathbf{A}\}^{\otimes}\right) \text{vec}\left(\mathcal{G}^{(k)}\right). \tag{7.11}$$

By concatenating all $\text{vec}\left(\mathcal{X}^{(k)}\right)$ for $k = 1, 2, \ldots, K$, we obtain a matrix factorization given by

$$\left[\text{vec}\left(\mathcal{X}^{(1)}\right) \text{vec}\left(\mathcal{X}^{(2)}\right) \cdots \text{vec}\left(\mathcal{X}^{(K)}\right)\right]^T = \left[\text{vec}\left(\mathcal{G}^{(1)}\right) \text{vec}\left(\mathcal{G}^{(2)}\right) \cdots \text{vec}\left(\mathcal{G}^{(K)}\right)\right]^T \left(\{\mathbf{A}\}^{\otimes}\right)^T \tag{7.12}$$



(a) Simultaneous decomposition of training data tensors with common basis factors.

(b) Conceptual classification diagram based on TUCKER decomposition and Projection Filter.

Figure 7.2: (a) Illustration of feature extraction from multi-way samples, and (b) the conceptual diagram illustrating a classification procedure based on the TUCKER decomposition of the concatenated tensor of all sampling training data. Reduced features are obtained by projecting the data tensor onto the feature subspace spanned by basis factors (bases).

For simplicity, denote the left side of Eq. (7.12) by a matrix $\mathbf{X}_{(N+1)} \in \mathbb{R}^{K \times (I_1 I_2 \cdots I_N)}$; then, we can write

$$\mathbf{X}_{(N+1)} = \left[\begin{array}{cccc} \text{vec}\left(\mathcal{X}^{(1)}\right) & \text{vec}\left(\mathcal{X}^{(2)}\right) & \cdots & \text{vec}\left(\mathcal{X}^{(K)}\right) \end{array} \right]^T. \tag{7.13}$$

Now, it is easy to prove that the matrix $\mathbf{X}_{(N+1)}$ is the mode-$(N+1)$ matricized version of an $(N+1)$-way concatenated tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots I_N \times K}$ obtained by concatenating all the data tensors $\mathcal{X}^{(k)}$ along the mode $(N+1)$. This can be formulated as

$$\mathcal{X} = \texttt{cat}(N+1, \mathcal{X}^{(1)}, \mathcal{X}^{(2)}, \ldots, \mathcal{X}^{(K)}), \tag{7.14}$$

where the sub-tensors are obtained by fixing the $(N+1)$-th index at a value k

$$\mathcal{X}(\underset{1}{:}, \underset{2}{:}, \ldots, \underset{N}{:}, \underset{N+1}{k}) = \mathcal{X}^{(k)}, \tag{7.15}$$

or alternatively it can be expressed as $\mathcal{X}^{(k)} = \mathcal{X}_{i_{N+1}=k} = \mathcal{X}_k$.

Similarly, the concatenation matrix $[\text{vec}\left(\mathcal{G}^{(k)}\right)]_{k=1}^{K} = [\text{vec}\left(\mathcal{G}^{(1)}\right), \text{vec}\left(\mathcal{G}^{(2)}\right), \ldots, \text{vec}\left(\mathcal{G}^{(K)}\right)]$ represents a matricization of an $(N+1)$ order core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N \times K}$ along the mode $(N+1)$ with its k-th sub-tensor, i.e., $\mathcal{G}(\underset{1}{:}, \underset{2}{:}, \ldots, \underset{N}{:}, \underset{N+1}{k}) = \mathcal{G}^{(k)}$. Thus, Eq. (7.12) can be rewritten in a compact matrix form

$$\mathbf{X}_{(N+1)} \approx \mathbf{G}_{(N+1)} \left(\{\mathbf{A}\}^{\otimes}\right)^T, \tag{7.16}$$

or equivalently in the form of tensor products

$$\mathcal{X} \approx \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)}, \tag{7.17}$$

which illustrates that the approximative simultaneous decomposition of a set of the training data tensors (7.9) is equivalent to decomposing the $(N+1)$-order concatenated tensor $\mathcal{X}$ via the TUCKER-N model. This provides a simple and elegant way to convert Problem 7.3 to the problem of decomposition of the concatenated data tensor $\mathcal{X}$ consisting of all data samples.

Problem 7.4 (Global TUCKER decomposition)
The N common bases of K samples $\mathcal{X}^{(k)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N} (k = 1, 2, \ldots, K)$ in Problem 7.3 are exactly the factors $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$, for $n = 1, \ldots, N$ in the TUCKER-N decomposition of the concatenation tensor along the mode-(N+1), that is

$$\mathcal{X} \approx \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)}, \tag{7.18}$$

where $\mathcal{X} = \mathtt{cat}(N+1, \mathcal{X}^{(1)}, \mathcal{X}^{(2)}, \ldots, \mathcal{X}^{(K)})$ and the core tensor $\mathcal{G}$ represents extracted features for the training samples.

Note that the features of a specific training data $\mathcal{X}^{(k)}$ are represented by the k-th row of the mode-$(N+1)$ matricized version of $\mathcal{G}$. Problem 7.4 is illustrated as the training step in Figure 7.2(b).

7.3.2 Orthogonal Interactive Bases

Interactive bases are estimated as factors of the TUCKER decomposition of the concatenation tensor $\mathcal{X}$. In order to avoid any confusion, orthogonal basis factors are denoted by $\mathbf{U}^{(n)}$. To develop algorithm, we first assume that the matrices $\mathbf{U}^{(n)}$ are known or have been estimated at a given step. So, the core tensor can be obtained as[58;60]

$$\mathcal{G} = \mathcal{X} \times_1 \mathbf{U}^{(1)\,T} \times_2 \mathbf{U}^{(2)\,T} \cdots \times_N \mathbf{U}^{(N)\,T}. \tag{7.19}$$

Therefore, we can maximize the cost function[58;60;62] to find factors $\mathbf{U}^{(n)}$ $(n = 1, 2, \ldots, N)$

$$J(\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}) = \left\| \mathcal{X} \times_1 \mathbf{U}^{(1)\,T} \times_2 \mathbf{U}^{(2)\,T} \cdots \times_N \mathbf{U}^{(N)\,T} \right\|_F^2, \tag{7.20}$$

where only the orthogonal basis matrices $\mathbf{U}^{(n)}$ are unknown. With $\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(n-1)}, \mathbf{U}^{(n+1)}, \ldots, \mathbf{U}^{(N)}$ fixed, we can project tensor $\mathcal{X}$ onto the subspace defined as

$$\mathcal{W}^{(-n)} = \mathcal{X} \times_1 \mathbf{U}^{(1)\,T} \cdots \times_{n-1} \mathbf{U}^{(n-1)\,T} \times_{n+1} \mathbf{U}^{(n+1)\,T} \cdots \times_N \mathbf{U}^{(N)\,T} = \mathcal{X} \times_{-(n,N+1)} \{\mathbf{U}^T\}, \tag{7.21}$$

and then the orthogonal matrix $\mathbf{U}^{(n)}$ can be estimated as R_n leading left singular vectors of the mode-n matricized version $\mathbf{W}_{(n)}^{(-n)}$. This leads to the Higher Order Orthogonal Iteration (HOOI) algorithm introduced by De Lathauwer, De Moor, and Vandewalle[60]. The pseudo-code of the algorithm for estimating N common bases is described in detail in Algorithm 7.1. In this algorithm, `svds` refers to as the Matlab SVD function which computes a few leading singular values and vectors.

Algorithm 7.1: HOOI Algorithm (Orthogonal TUCKER) for Features Extraction

Input: $\mathcal{X}$: concatenation tensor of all training samples $I_1 \times I_2 \times \cdots \times I_N \times K$,
$R_1, R_2, \ldots, R_N$: number of basis components for factors
Output: N orthogonal factors $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ and a core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N \times K}$.

begin
  HOSVD or random initialization for all factors $\mathbf{U}^{(n)}$
  repeat
    for $n = 1$ *to* N **do**
      $\mathcal{W}^{(-n)} = \mathcal{X} \times_{-(n,N+1)} \{\mathbf{U}^T\}$
      $[\mathbf{U}^{(n)}, \mathbf{\Sigma}^{(n)}, \mathbf{V}^{(n)}] = \texttt{svds}(\mathbf{W}^{(-n)}_{(n)}, R_n, \text{'LM'})$ // $\mathbf{W}^{(-n)}_{(n)} \approx \mathbf{U}^{(n)}\mathbf{\Sigma}^{(n)}\mathbf{V}^{(n)\,T}$
    end
  until *a stopping criterion is met*
  $\mathcal{G} = \mathcal{W}^{(-N)} \times_N \mathbf{U}^{(N)T}$
end

7.3.3 Nonnegative Interactive Bases

A nonnegative object can be expressed as a linear combination of its sparse parts which are considered as basis components. The weights of this combination can be used as features of this object in a reduced dimension subspace. Therefore, for nonnegative datasets, NTD can be applied to find basis factors and to extract features, and Problem 7.4 becomes the NTD with N factors.

7.4 Discriminant Analysis Approach for Multi-way Features

The training features obtained by non-interactive bases (CP) or by interactive bases (Tucker model) can be directly used for classification. However, they do not contain any category (label of a class) information which is often useful to model the difference between classes of data. To exploit such information, we should find discriminant bases to project the training features $\mathcal{G}^{(k)}$ onto the discriminant subspaces. Since entries $g^{(k)}_{r_1,r_2,\ldots,r_N}$ can be considered as independent features, and metrics comparing two multidimensional samples are the same as when evaluating of their vectorizations (e.g. Euclidean distance, Kullback-Leibler divergence), we can vectorize feature tensors $\mathcal{G}^{(k)}$, and apply any 1-D discriminant methods for the training features.

An alternative approach is that discriminant projections are directly searched for the raw dataset, and the feature tensors $\mathcal{G}^{(k)}$ are coordinate values of these projections. The basis factors are derived either from the Fisher discriminant criterion, or from the cost functions which are incorporated discriminant constraints.

Recently, a number of algorithms have been proposed for discriminant analysis with high dimensional representations. He *et al.*[85] have first proposed algorithm to find the discriminant bases for

2-D samples. Yan *et al.*[213] proposed a discriminant analysis for tensor representation based on Fisher score. Zhang *et al.*[218] estimated discriminant bases via Laplacian score. Feiping *et al.*[135] found bases exploiting local scatter by defining local weight matrices. However, those methods are different and are not directly related to TUCKER decompositions. In our models and algorithms, the bases can be easily estimated from tensor decompositions, especially the TUCKER decomposition. Moreover, the proposed algorithms can flexibly switch between the orthonormal and discriminant bases with a regularization parameter. The HOOI algorithm once again has been confirmed as a "work-horse" algorithm in dimensionality reduction, feature extraction and classification.

7.4.1 Discriminant Analysis of Features

Consider L features obtained by a multiway decomposition. For interactive bases (TUCKER model), $L = R_1 R_2 \cdots R_N$ is the dimension of the core feature tensors $\mathcal{G}^{(k)}$. The training features of the k-th sample are denoted by $\boldsymbol{g}^{(k)} = [g_l^{(k)}] = \text{vec}\left(\mathcal{G}^{(k)}\right) \in \mathbb{R}^L$ and $g_l^{(k)}$, for $l = 1, 2, \ldots, L$ is the l-th entry (feature) of the vectorized version of the core tensor $\mathcal{G}^{(k)}$.

This section will present a simple LDA method to find the discriminant projection matrix $\boldsymbol{\Psi} \in \mathbb{R}^{L \times F}, (F \ll L)$ for the features $\boldsymbol{g}^{(k)}$, that is (in fact, we can apply any LDA method[28;29])

$$\boldsymbol{f}^{(k)} = \boldsymbol{\Psi}^T \text{vec}\left(\mathcal{G}^{(k)}\right), \tag{7.22}$$

where $\boldsymbol{f}^{(k)} \in \mathbb{R}^F$ are the discriminant features. We shall denote the average vectors for each class by $\bar{\boldsymbol{g}}^{(c)} (c = 1, 2, \ldots, C)$ and the corresponding average for the whole set of samples by $\bar{\bar{\boldsymbol{g}}}$, that is

$$\bar{\boldsymbol{g}}^{(c)} = \frac{1}{K_c} \sum_{k \in \mathcal{I}_c} \boldsymbol{g}^{(k)}, \qquad \bar{\bar{\boldsymbol{g}}} = \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{g}^{(k)}, \tag{7.23}$$

where $\mathcal{I}_c$ is the subset of indices k which indicates the samples k belong to class c, and K_c is the number of training samples in the c-th class.

The average core tensor corresponding to the average features $\bar{\boldsymbol{g}}^{(c)}$ is denoted by $\bar{\mathcal{G}}^{(c)}$, and given by

$$\bar{\mathcal{G}}^{(c)} = \left(\sum_{k \in \mathcal{I}_c} \mathcal{G}^{(k)} \right) / K_c \qquad (c = 1, 2, \ldots, C). \tag{7.24}$$

By removing the averages $\bar{\mathcal{G}}^{(c_k)}$ for all the samples $\bar{\mathcal{G}}^{(k)}$, a new set of centered tensors $\tilde{\mathcal{G}}^{(k)}$ is defined as

$$\tilde{\mathcal{G}}^{(k)} = \mathcal{G}^{(k)} - \bar{\mathcal{G}}^{(c_k)}. \tag{7.25}$$

Concatenation of all the core tensors $\tilde{\mathcal{G}}^{(k)}$ forms an $(N+1)$-D tensor $\tilde{\mathcal{G}}$ so that: $\tilde{\mathcal{G}}_k = \tilde{\mathcal{G}}^{(k)}$. To avoid any confusion in notation regarding the concatenation tensor of average tensors $\bar{\mathcal{G}}^{(c)}$, the average tensor for all the data tensor is denoted by $\bar{\bar{\mathcal{G}}}$ with its vectorization form given by $\bar{\bar{\boldsymbol{g}}} = \text{vec}\left(\bar{\bar{\mathcal{G}}}\right)$

$$\bar{\bar{\mathcal{G}}} = \left(\sum_{k=1}^{K} \mathcal{G}^{(k)} \right) / K. \tag{7.26}$$

We can also remove the average part for all the samples $\bar{\mathcal{G}}^{(c)}$ to form a new set of tensors $\breve{\mathcal{G}}^{(c)}$

$$\breve{\mathcal{G}}^{(c)} \;=\; \sqrt{K_c}\left(\bar{\mathcal{G}}^{(c)} - \bar{\bar{\mathcal{G}}}\right), \tag{7.27}$$

which are parts of the concatenation tensor $\breve{\mathcal{G}}$, i.e: $\breve{\mathcal{G}}_c = \breve{\mathcal{G}}^{(c)}$.

The corresponding discriminant features are therefore given by

$$\bar{\boldsymbol{f}}^{(c)} = \boldsymbol{\Psi}^T \bar{\boldsymbol{g}}^{(c)}, \qquad \bar{\bar{\boldsymbol{f}}} = \boldsymbol{\Psi}^T \bar{\bar{\boldsymbol{g}}}, \tag{7.28}$$

whereas the discriminant projection matrix $\boldsymbol{\Psi}$ can be found by maximizing the Fisher discriminant criterion[2;125] defined as

$$\boldsymbol{\Psi}=\arg\max_{\boldsymbol{\Psi}} \varphi(\boldsymbol{\Psi}) = \arg\max_{\boldsymbol{\Psi}} \frac{\sum_{c=1}^{C} K_c \,\|\bar{\boldsymbol{f}}^{(c)} - \bar{\bar{\boldsymbol{f}}}\|_2^2}{\sum_{k=1}^{K} \|\boldsymbol{f}^{(k)} - \bar{\boldsymbol{f}}^{(c_k)}\|_2^2} = \arg\max_{\boldsymbol{\Psi}} \frac{\sum_{c=1}^{C} K_c \,\|\boldsymbol{\Psi}^T\bar{\boldsymbol{g}}^{(c)} - \boldsymbol{\Psi}^T\bar{\bar{\boldsymbol{g}}}\|_2^2}{\sum_{k=1}^{K} \|\boldsymbol{\Psi}^T\boldsymbol{g}^{(k)} - \boldsymbol{\Psi}^T\bar{\boldsymbol{g}}^{(c_k)}\|_2^2}, \tag{7.29}$$

where c_k indicates the category of sample k.

By defining the within-class scatter matrix $\mathbf{S}_w$ and the between-class scatter matrix $\mathbf{S}_b$ for the features $\boldsymbol{g}^{(k)}$ as[2;125]

$$\begin{aligned}
\mathbf{S}_w &= \sum_{k=1}^{K} (\boldsymbol{g}^{(k)} - \bar{\boldsymbol{g}}^{(c_k)})(\boldsymbol{g}^{(k)} - \bar{\boldsymbol{g}}^{(c_k)})^T = \sum_{k=1}^{K} \tilde{\boldsymbol{g}}^{(k)}\,\tilde{\boldsymbol{g}}^{(k)\,T} \\
&= \tilde{\mathbf{G}}_{(N+1)}^T\,\tilde{\mathbf{G}}_{(N+1)} = \langle \tilde{\mathcal{G}}, \tilde{\mathcal{G}} \rangle_{-(N+1)}, \qquad (7.30) \\
\mathbf{S}_b &= \sum_{c=1}^{C} K_c\,(\bar{\boldsymbol{g}}^{(c)} - \bar{\bar{\boldsymbol{g}}})(\bar{\boldsymbol{g}}^{(c)} - \bar{\bar{\boldsymbol{g}}})^T = \sum_{c=1}^{C} \breve{\boldsymbol{g}}^{(c)}\,\breve{\boldsymbol{g}}^{(c)\,T} \\
&= \langle \breve{\mathcal{G}}, \breve{\mathcal{G}} \rangle_{-(N+1)}, \qquad (7.31)
\end{aligned}$$

it can be shown that expression (7.29) is equivalent to the trace ratio problem[2;125]

$$\boldsymbol{\Psi} = \arg\max_{\boldsymbol{\Psi}} \varphi(\boldsymbol{\Psi}) = \arg\max_{\boldsymbol{\Psi}} \frac{\operatorname{tr}\left[\boldsymbol{\Psi}^T\,\mathbf{S}_b\,\boldsymbol{\Psi}\right]}{\operatorname{tr}\left[\boldsymbol{\Psi}^T\,\mathbf{S}_w\,\boldsymbol{\Psi}\right]}, \tag{7.32}$$

or the simpler inexact problem

$$\boldsymbol{\Psi} = \arg\max_{\boldsymbol{\Psi}} \operatorname{tr}\left[\boldsymbol{\Psi}^T\,\mathbf{S}_w^{-1}\,\mathbf{S}_b\,\boldsymbol{\Psi}\right], \tag{7.33}$$

which can be solved by the generalized eigenvalue decomposition (GEVD)

$$\mathbf{S}_b\,\boldsymbol{\psi} = \tau\,\mathbf{S}_w\,\boldsymbol{\psi}. \tag{7.34}$$

The projection matrix $\boldsymbol{\Psi}$ is composed by the leading eigenvectors $\boldsymbol{\psi}$ of (7.34).

Algorithm 7.2: Discriminant Algorithm for Reduced Multidimensional Features

input : $\mathcal{G}$: tensor of K training features $(R_1 \times \cdots \times R_N \times K)$
$\mathcal{I}$: set of indices or labels indicating categories of samples.
output: $\mathbf{\Psi}$: discriminant projection matrix $(L \times F)$
$\mathbf{F}$: matrix of discriminant features $(F \times K)$

begin
  $\bar{\bar{\mathcal{G}}} = \left(\sum_{k=1}^{K} \mathcal{G}^{(k)}\right)/K$ // average tensor for all training features
  foreach *class* c **do**
    $\bar{\mathcal{G}}^{(c)} = \left(\sum_{k \in \mathcal{I}_c} \mathcal{G}^{(k)}\right)/K_c$ // average feature tensor for class c
    foreach $k \in \mathcal{I}_c$ **do** $\tilde{\mathcal{G}}^{(k)} \leftarrow \mathcal{G}^{(k)} - \bar{\mathcal{G}}^{(c)}$ // centralize
    $\bar{\mathcal{G}}^{(c)} \leftarrow \sqrt{K_c}\left(\bar{\mathcal{G}}^{(c)} - \bar{\bar{\mathcal{G}}}\right)$
  end
  $\mathbf{S}_w = \langle \tilde{\mathcal{G}}, \tilde{\mathcal{G}} \rangle_{-(N+1)}$ // within-class scatter matrix
  $\mathbf{S}_b = \langle \bar{\mathcal{G}}, \bar{\mathcal{G}} \rangle_{-(N+1)}$ // between-class scatter matrix
  $[\mathbf{\Psi}, \mathbf{\Lambda}] = \texttt{eigs}(\mathbf{S}_b, \mathbf{S}_w, F, \text{'LM'})$ // initialize
  repeat
    $\varphi = \dfrac{\texttt{trace}\left(\mathbf{\Psi}^T \mathbf{S}_b \mathbf{\Psi}\right)}{\texttt{trace}\left(\mathbf{\Psi}^T \mathbf{S}_w \mathbf{\Psi}\right)}$
    $[\mathbf{\Psi}, \mathbf{\Lambda}] = \texttt{eigs}(\mathbf{S}_b - \varphi \mathbf{S}_w, F, \text{'LM'})$ // or compute GEVD (7.34)
    $[\mathbf{\Psi}, \mathbf{\Lambda}] = \texttt{eigs}(\mathbf{\Psi}\mathbf{\Psi}^T \mathbf{S}_w \mathbf{\Psi}\mathbf{\Psi}^T, F, \text{'LM'})$
  until *a criterion is met*
  $\mathbf{F} = \mathbf{\Psi}^T \mathbf{G}_{(N+1)}^T$
end

An efficient method to solve the trace ratio problem (7.32) is to iteratively solve a trace difference problem[209]

$$\mathbf{\Psi} = \arg\max_{\mathbf{\Psi}} \operatorname{tr}\left[\mathbf{\Psi}^T \left(\mathbf{S}_b - \varphi \mathbf{S}_w\right) \mathbf{\Psi}\right]. \tag{7.35}$$

The pseudocode of this method is summarized in Algorithm 7.2.

Note that the feature vectors are obtained by the linear transformation

$$\boldsymbol{f}_{(k)} = \mathbf{\Psi}^T \operatorname{vec}\left(\mathcal{G}^{(k)}\right), \tag{7.36}$$

or equivalently, the matrix of training features $\mathbf{F} = [\boldsymbol{f}_k] \in \mathbb{R}^{F \times K}$ is given by $\mathbf{F} = \mathbf{\Psi}^T \mathbf{G}_{(N+1)}^T$.

7.4.2 High Order Discriminant Analysis using Orthogonal Tucker Decomposition

An alternative approach to exploit the discriminant information for TUCKER features is that the core feature tensors $\mathcal{G}^{(k)}$ are directly projected on the discriminant bases.

In general, we can maximize the Fisher ratio between the core tensors $\mathcal{G}^{(k)}$ to find the orthogonal basis factors $\mathbf{U}^{(n)}$

$$\varphi = \arg \max_{\mathbf{U}^{(1)},\ldots,\mathbf{U}^{(N)}} \frac{\sum_{c=1}^{C} K_c \left\| \bar{\mathcal{G}}^{(c)} - \bar{\bar{\mathcal{G}}} \right\|_F^2}{\sum_{k}^{K} \left\| \mathcal{G}^{(k)} - \bar{\mathcal{G}}^{(c_k)} \right\|_F^2}, \tag{7.37}$$

where $\bar{\mathcal{G}}^{(c)}$ is the mean tensor of the c-th class consisting of K_c training samples, c_k denotes the class to which the k-th training sample $\mathcal{X}^{(k)}$ belongs, and $\bar{\bar{\mathcal{G}}}$ is the mean tensor of the whole training features. This technique provides a generalization of the 1-D Linear Discriminant Analysis to multilinear one.

In a similar way to the HOOI algorithm presented in Section 7.3.2, the learning rule for the factor $\mathbf{U}^{(n)}$ is derived with the assumption that all the other factors are fixed. Taking into account that the basis factors are orthogonal, we can express the denominator of (7.37) via the trace of the within class scatter matrix:

$$\begin{aligned}
\sum_{k=1}^{K} \left\| \mathcal{G}^{(k)} - \bar{\mathcal{G}}^{(c_k)} \right\|_F^2 &= \sum_{k=1}^{K} \left\| \left(\mathcal{X}^{(k)} \times_{-n} \{\mathbf{U}^T\} \right) \times_n \mathbf{U}^{(n)\,T} - \left(\bar{\mathcal{X}}^{(c_k)} \times_{-n} \{\mathbf{U}^T\} \right) \times_n \mathbf{U}^{(n)\,T} \right\|_F^2 \\
&= \sum_{k=1}^{K} \left\| \left(\mathcal{X}^{(k)} - \bar{\mathcal{X}}^{(c_k)} \right) \times_{-n} \{\mathbf{U}^T\} \times_n \mathbf{U}^{(n)\,T} \right\|_F^2 = \sum_{k=1}^{K} \left\| \tilde{\mathcal{Z}}^{(k)}_{-n} \times_n \mathbf{U}^{(n)\,T} \right\|_F^2 \\
&= \sum_{k=1}^{K} \mathrm{tr} \left[\mathbf{U}^{(n)\,T} \langle \tilde{\mathcal{Z}}^{(k)}_{-n}, \tilde{\mathcal{Z}}^{(k)}_{-n} \rangle_{-n} \mathbf{U}^{(n)} \right] = \mathrm{tr} \left[\mathbf{U}^{(n)\,T} \left(\sum_{k=1}^{K} \langle \tilde{\mathcal{Z}}^{(k)}_{-n}, \tilde{\mathcal{Z}}^{(k)}_{-n} \rangle_{-n} \right) \mathbf{U}^{(n)} \right] \\
&= \mathrm{tr} \left[\mathbf{U}^{(n)\,T} \mathbf{S}_w^{-n} \mathbf{U}^{(n)} \right],
\end{aligned} \tag{7.38}$$

where $\tilde{\mathcal{Z}}^{(k)}_{-n} = \tilde{\mathcal{X}}^{(k)} \times_{-n} \{\mathbf{U}^T\}$ are the k-part of the concatenation tensor $\tilde{\mathcal{Z}}^{-n}$, i.e. $\tilde{\mathcal{Z}}^{-n}_k = \tilde{\mathcal{Z}}^{(k)}_{-n}$, and

$$\tilde{\mathcal{X}}^{(k)} = \mathcal{X}^{(k)} - \bar{\mathcal{X}}^{(c_k)}, \quad \tilde{\mathcal{Z}}^{-n} = \tilde{\mathcal{X}} \times_{-(n,N+1)} \{\mathbf{U}^T\}, \tag{7.39}$$

and the within-class scatter matrix $\mathbf{S}_w^{-n}$ is defined as

$$\mathbf{S}_w^{-n} = \sum_{k=1}^{K} \langle \tilde{\mathcal{Z}}^{(k)}_{-n}, \tilde{\mathcal{Z}}^{(k)}_{-n} \rangle_{-n} = \langle \tilde{\mathcal{Z}}^{-n}, \tilde{\mathcal{Z}}^{-n} \rangle_{-n}. \tag{7.40}$$

Similarly, the between-class scatter is expressed as trace of the between class scatter matrix:

$$\begin{aligned}
\sum_{c=1}^{C} K_c \left\| \bar{\mathcal{G}}^{(c)} - \bar{\bar{\mathcal{G}}} \right\|_F^2 &= \sum_{c=1}^{C} K_c \left\| \left(\bar{\mathcal{X}}^{(c)} \times_{-n} \{\mathbf{U}^T\} \right) \times_n \mathbf{U}^{(n)\,T} - \left(\bar{\bar{\mathcal{X}}} \times_{-n} \{\mathbf{U}^T\} \right) \times_n \mathbf{U}^{(n)\,T} \right\|_F^2 \\
&= \sum_{c=1}^{C} K_c \left\| \left(\bar{\mathcal{X}}^{(c)} - \bar{\bar{\mathcal{X}}} \right) \times_{-n} \{\mathbf{U}^T\} \times_n \mathbf{U}^{(n)\,T} \right\|_F^2 = \sum_{c=1}^{C} \left\| \check{\mathcal{Z}}^{(c)}_{-n} \times_n \mathbf{U}^{(n)\,T} \right\|_F^2 \\
&= \sum_{c=1}^{C} \mathrm{tr} \left[\mathbf{U}^{(n)\,T} \langle \check{\mathcal{Z}}^{(c)}_{-n}, \check{\mathcal{Z}}^{(c)}_{-n} \rangle_{-n} \mathbf{U}^{(n)} \right] = \mathrm{tr} \left[\mathbf{U}^{(n)\,T} \left(\sum_{c=1}^{C} \langle \check{\mathcal{Z}}^{(c)}_{-n}, \check{\mathcal{Z}}^{(c)}_{-n} \rangle_{-n} \right) \mathbf{U}^{(n)} \right] \\
&= \mathrm{tr} \left[\mathbf{U}^{(n)\,T} \mathbf{S}_b^{-n} \mathbf{U}^{(n)} \right],
\end{aligned} \tag{7.41}$$

Algorithm 7.3: HODA Algorithm for Feature Extraction

input : $\mathcal{X}$: Concanated tensor of K training samples $I_1 \times I_2 \times \cdots \times I_N \times K$
output: $\mathbf{U}^{(n)}$: N orthogonal basis factors $I_n \times R_n$ $(n = 1, 2, \ldots, N)$
$\mathcal{G}$: Training feature tensors $R_1 \times R_2 \times \cdots \times R_N \times K$.

begin
  Initialize $\mathbf{U}^{(n)}$
  Calculate $\tilde{\mathcal{X}}$, and $\check{\mathcal{X}}$ according to (7.39) and (7.42)
  repeat
    for $n = 1$ *to* N **do**
      $\tilde{\mathcal{Z}} = \tilde{\mathcal{X}} \times_{-(n,N+1)} \{\mathbf{U}^T\}$
      $\mathbf{S}_w^{-n} = \langle \tilde{\mathcal{Z}}, \tilde{\mathcal{Z}} \rangle_{-n}$ `// within-class scatter matrix`
      $\check{\mathcal{Z}} = \check{\mathcal{X}} \times_{-(n,N+1)} \{\mathbf{U}^T\}$
      $\mathbf{S}_b^{-n} = \langle \check{\mathcal{Z}}, \check{\mathcal{Z}} \rangle_{-n}$ `// between-class scatter matrix`
      $\varphi = \frac{\mathtt{trace}\left(\mathbf{U}^{(n)\,T} \mathbf{S}_b^{-n} \mathbf{U}^{(n)}\right)}{\mathtt{trace}\left(\mathbf{U}^{(n)\,T} \mathbf{S}_w^{-n} \mathbf{U}^{(n)}\right)}$
      $[\mathbf{U}^{(n)}, \mathbf{\Lambda}] = \mathtt{eigs}(\mathbf{S}_b^{-n} - \varphi \mathbf{S}_w^{-n}, R_n, \text{'LM'})$ `// or`
      $[\mathbf{U}^{(n)}, \mathbf{\Lambda}] = \mathtt{eigs}(\mathbf{S}_b^{-n}, \mathbf{S}_w^{-n}, R_n, \text{'LM'})$
      $[\mathbf{U}^{(n)}, \mathbf{\Lambda}] = \mathtt{eigs}(\mathbf{U}^{(n)} \mathbf{U}^{(n)\,T} \langle \mathcal{X}, \mathcal{X} \rangle_{-n} \mathbf{U}^{(n)} \mathbf{U}^{(n)\,T}, R_n, \text{'LM'})$
    end
  until *a criterion is met*
  $\mathcal{G} = \mathcal{X} \times_{-(N+1)} \{\mathbf{U}\}^T$
end

where $\check{\mathcal{Z}}_{-n}^{(c)} = \check{\mathcal{X}}^{(c)} \times_{-n} \{\mathbf{U}^T\}$ are the c-part of the concatenation tensor $\check{\mathcal{Z}}^{-n}$, i.e., $\check{\mathcal{Z}}_c^{-n} = \check{\mathcal{Z}}_{-n}^{(c)}$,

$$\check{\mathcal{X}}^{(c)} = \sqrt{K_c} \left(\bar{\mathcal{X}}^{(c)} - \bar{\bar{\mathcal{X}}} \right), \tag{7.42}$$

$$\check{\mathcal{Z}}^{-n} = \check{\mathcal{X}} \times_{-(n,N+1)} \{\mathbf{U}^T\}, \tag{7.43}$$

and the between-class scatter matrix $\mathbf{S}_b^{-n}$ is defined as

$$\mathbf{S}_b^{-n} = \sum_{c=1}^{C} \langle \check{\mathcal{Z}}_{-n}^{(c)}, \check{\mathcal{Z}}_{-n}^{(c)} \rangle_{-n} = \langle \check{\mathcal{Z}}^{-n}, \check{\mathcal{Z}}^{-n} \rangle_{-n}. \tag{7.44}$$

By substituting (7.38) and (7.41) into the cost function (7.37), the discriminant factor $\mathbf{U}^{(n)}$ can be found via maximizing the following trace ratio

$$\varphi = \arg\max_{\mathbf{U}^{(n)}} \frac{\mathrm{tr}\left[\mathbf{U}^{(n)\,T} \mathbf{S}_b^{-n} \mathbf{U}^{(n)}\right]}{\mathrm{tr}\left[\mathbf{U}^{(n)\,T} \mathbf{S}_w^{-n} \mathbf{U}^{(n)}\right]} = \arg\max_{\mathbf{U}^{(n)}} \frac{\mathrm{tr}\left[\mathbf{S}_b\right]}{\mathrm{tr}\left[\mathbf{S}_w\right]} \quad , (n = 1, 2, \ldots, N). \tag{7.45}$$

This can be used to solve problem (7.32), where the factors $\mathbf{U}^{(n)}$ can be found as R_n leading left generalized eigenvectors of the generalized eigenvalue decomposition $\mathbf{S}_w \mathbf{U}^{(n)} = \lambda \mathbf{S}_b \mathbf{U}^{(n)}$ or R_n leading eigenvector of matrix $(\mathbf{S}_b - \varphi \mathbf{S}_w)$. Alternating estimation of factors $\mathbf{U}^{(n)}$ gives us the High Order

Discriminant Analysis algorithm (HODA). The pseudocode of this algorithm is given in Algorithm 7.3. We note that although generalized eigenvectors $\mathbf{U}^{(n)}$ from the decomposition $\mathbf{S}_w\mathbf{U}^{(n)} = \lambda\mathbf{S}_b\,\mathbf{U}^{(n)}$ are not orthogonal, features can also be extracted using the approximate projection given in (7.19).

Since $\mathbf{S}_w$ can be very ill-conditioned, especially in early updates, the system (7.45) may have no solution or can have infinite solutions (the linear equations system is underdetermined)[74]. Thus, to avoid the breakdown of iterations some sort of regularization is essential, for instance,

$$\varphi = \arg\max_{\mathbf{U}^{(n)}} \frac{\operatorname{tr}\left[\mathbf{S}_b\right]}{\operatorname{tr}\left[\mathbf{S}_w + \alpha\,\mathbf{I}\right]}, \tag{7.46}$$

where $\mathbf{I}$ is the identity matrix and $\alpha \geqslant 0$ is the regularization parameter.

We note that seeking the optimal projective orthogonal bases $\mathbf{U}^{(n)}$ in the feature space is equivalent to solving the following optimization problem

$$\varphi = \arg\max_{\mathbf{U}^{(n)}} \frac{\operatorname{tr}\left[\mathbf{S}_b\right]}{\operatorname{tr}\left[\mathbf{S}_w\right]} = \arg\max_{\mathbf{U}^{(n)}} \frac{\operatorname{tr}\left[\mathbf{S}_t\right]}{\operatorname{tr}\left[\mathbf{S}_w\right]}, \tag{7.47}$$

where the total scatter matrix $\mathbf{S}_t = \mathbf{S}_b + \mathbf{S}_w$ and trace of this matrix is given by

$$\operatorname{tr}\left[\mathbf{S}_t\right] = \operatorname{tr}\left[\mathbf{S}_b + \mathbf{S}_w\right] = \sum_{k=1}^{K} \left\| \boldsymbol{\mathcal{G}}^{(k)} - \bar{\bar{\boldsymbol{\mathcal{G}}}} \right\|_F^2. \tag{7.48}$$

Hence, an alternative regularization form can be established as

$$\varphi = \arg\max_{\mathbf{U}^{(n)}} \frac{\operatorname{tr}\left[\mathbf{S}_t\right]}{\operatorname{tr}\left[\alpha\,\mathbf{S}_w + (1-\alpha)\,\mathbf{I}\right]}, \tag{7.49}$$

where $0 \leqslant \alpha \leqslant 1$. The basis factors $\mathbf{U}^{(n)}$ are R_n leading left eigenvectors of matrices $((\alpha\,\mathbf{S}_w + (1-\alpha)\,\mathbf{I})^{-1}\mathbf{S}_b)$ or $(\mathbf{S}_b - \varphi\,(\alpha\,\mathbf{S}_w + (1-\alpha)\,\mathbf{I}))$. The choice of parameter α can be crucial to yield a good performance. We note that the typical value of α is 0 or 1.

- For $\alpha = 1$, the optimization problem (7.49) simplifies into the problem given in (7.45), i.e. we obtain the HODA algorithm without regularization.
- For $\alpha = 0$, the optimization problem (7.49) simplifies the maximization of (7.48). In fact, this problem is equivalent to the TUCKER decomposition of the training samples after centering (7.20) with orthogonality constraints and solved by the HOOI algorithm (see Section 7.3.2).

7.4.3 Discriminant Analysis for NTD

This section discusses ways of finding discriminant basis factors for nonnegative TUCKER decomposition. The method is based on simultaneously solving two optimization problems:

1. Minimize the Frobenius norm (2.51) of the raw data and its approximation to find interpretable basis factors;

2. Maximize the Fisher score of the projected features (7.37) onto the subspace of the estimated factors.

For this purpose, a new global cost function with penalty terms is designed as

$$D_F\left(\mathcal{X}\|\mathcal{G},\{\mathbf{A}\}\right)=\frac{1}{2}\|\mathcal{X}-\mathcal{G}\times_{-(N+1)}\{\mathbf{A}\}\|_F^2+\frac{1}{2}\lambda_o\sum_{n=1}^{N}\sum_{r\neq p}\boldsymbol{a}_r^{(n)T}\boldsymbol{a}_p^{(n)}+\frac{1}{2}\lambda_w\,\mathrm{tr}\left[\mathbf{S}_w\right]-\frac{1}{2}\lambda_b\,\mathrm{tr}\left[\mathbf{S}_b\right]. \quad (7.50)$$

The second regularization term enforces (as much as possible) the orthogonality of basis components $\boldsymbol{a}_r^{(n)}$, that is, components $\boldsymbol{a}_r^{(n)}$ should be as sparse as possible. The within-class and between-class scatter matrices $\mathbf{S}_w$ and $\mathbf{S}_b$ are defined in (7.38) and (7.41). The last two regularization terms require $\mathrm{tr}\left[\mathbf{S}_w\right]$ to be as small as possible, while $\mathrm{tr}\left[\mathbf{S}_b\right]$ should be as large as possible.

In order to estimate the nonnegative basis factors, we need to compute the gradients of regularization terms with respect to $\mathbf{A}^{(n)}$ given by

$$\frac{\partial\left(\sum_{n=1}^{N}\sum_{r\neq p}\boldsymbol{a}_r^{(n)T}\boldsymbol{a}_p^{(n)}\right)}{\partial\mathbf{A}^{(n)}}=\mathbf{A}^{(n)}\left(\mathbf{1}\mathbf{1}^T-\mathbf{I}\right). \quad (7.51)$$

Using a gradient descent approach[47;103;131] we can derive a multiplicative learning rule for $\mathbf{A}^{(n)}$ as

$$\mathbf{A}^{(n)}\leftarrow\mathbf{A}^{(n)}\circledast\langle\mathcal{X}\times_{-(n,N+1)}\{\mathbf{A}^T\},\mathcal{G}\rangle_{-n}\oslash\left(\mathbf{A}^{(n)}\langle\mathcal{G},\mathcal{G}\times_{-(n,N+1)}\{\mathbf{A}^T\mathbf{A}\}\rangle_{-n}+\lambda_o\mathbf{A}^{(n)}\left(\mathbf{1}\mathbf{1}^T-\mathbf{I}\right)\right). \quad (7.52)$$

To derive the learning rule for the core tensor $\mathcal{G}$ we shall assume that all the factors $\mathbf{A}^{(n)}$ are fixed. Then, the gradients of the 3-rd and 4-th regularization terms in (7.50) with respect to $\mathcal{G}^{(k)}$ are given by

$$\frac{\partial\mathrm{tr}\left[\mathbf{S}_w\right]}{\partial\mathcal{G}^{(k)}}=\frac{\partial\sum_{k=1}^{K}\left\|\mathcal{G}^{(k)}-\bar{\mathcal{G}}^{(c_k)}\right\|_F^2}{\partial\mathcal{G}^{(k)}}=2\left(\mathcal{G}^{(k)}-\bar{\mathcal{G}}^{(c_k)}\right), \quad (7.53)$$

$$\frac{\partial\mathrm{tr}\left[\mathbf{S}_b\right]}{\partial\mathcal{G}^{(k)}}=\frac{\partial\sum_{c=1}^{C}K_c\left\|\bar{\mathcal{G}}^{(c)}-\bar{\bar{\mathcal{G}}}\right\|_F^2}{\partial\mathcal{G}^{(k)}}=2\left(\bar{\mathcal{G}}^{(c_k)}-\bar{\bar{\mathcal{G}}}\right), \quad (7.54)$$

and the multiplicative update rule for $\mathcal{G}$[47;103;131;157] is modified to update rule a sample $\mathcal{G}^{(k)}$ as

$$\mathcal{G}^{(k)}\leftarrow\mathcal{G}^{(k)}\circledast\left(\mathcal{X}^{(k)}\times\{\mathbf{A}^T\}+(\lambda_w+\lambda_b)\,\bar{\mathcal{G}}^{(c_k)}\right)\oslash\left(\mathcal{G}^{(k)}\times\{\mathbf{A}^T\mathbf{A}\}+\lambda_w\,\mathcal{G}^{(k)}+\lambda_b\,\bar{\bar{\mathcal{G}}}\right). \quad (7.55)$$

This update rule can be rewritten for the whole concatenated core tensor of features as

$$\mathcal{G}\leftarrow\mathcal{G}\circledast\left(\mathcal{X}\times_{-(N+1)}\{\mathbf{A}^T\}+(\lambda_w+\lambda_b)\,\bar{\mathcal{G}}^{*}\right)\oslash\left(\mathcal{G}\times_{-(N+1)}\{\mathbf{A}^T\mathbf{A}\}+\lambda_w\,\mathcal{G}+\lambda_b\,\bar{\bar{\mathcal{G}}}^{*}\right), \quad (7.56)$$

where $\bar{\mathcal{G}}^{*}$ is an $(N+1)$-way tensor whose each part $\bar{\mathcal{G}}_k^{*}$ is the average tensor corresponding to class c_k

$$\bar{\mathcal{G}}_k^{*}=\bar{\mathcal{G}}^{(c_k)} \quad (7.57)$$

Algorithm 7.4: Multiplicative Discriminant Analysis for NTD

input : $\mathcal{X}$: tensor of K training samples $I_1 \times I_2 \times \cdots \times I_N \times K$
λ_o, λ_w and λ_b: regularization parameters.
output: $\mathbf{A}^{(n)}$: N nonnegative factors $I_n \times R_n$
$\mathcal{G}$: nonnegative training feature core tensors $R_1 \times R_2 \times \cdots \times R_N \times K$.

begin
  Initialize $\mathbf{A}^{(n)}$
  repeat
    for $n = 1$ *to* N **do**
      $\mathbf{A}^{(n)} \leftarrow \mathbf{A}^{(n)} \circledast \langle \mathcal{X} \times_{-(n,N+1)} \{\mathbf{A}^T\}, \mathcal{G} \rangle_{-n} \oslash \left(\mathbf{A}^{(n)} \langle \mathcal{G}, \mathcal{G} \times_{-(n,N+1)} \{\mathbf{A}^T \mathbf{A}\} \rangle_{-n} + \lambda_o \mathbf{A}^{(n)} \left(\mathbf{1}\mathbf{1}^T - \mathbf{I} \right) \right)$,
    end
    $\bar{\bar{\mathcal{G}}} = \frac{1}{K} \sum_k \mathcal{G}^{(k)}$
    foreach $c = 1$ *to* C **do** $\bar{\mathcal{G}}^{(c)} = \frac{1}{K_c} \sum_{k \in \mathcal{I}_c} \mathcal{G}^{(k)}$
    $\mathcal{G} \leftarrow \mathcal{G} \circledast \left(\mathcal{X} \times_{-(N+1)} \{\mathbf{A}^T\} + (\lambda_w + \lambda_b) \bar{\mathcal{G}}^* \right) \oslash \left(\mathcal{G} \times_{-(N+1)} \{\mathbf{A}^T \mathbf{A}\} + \lambda_w \mathcal{G} + \lambda_b \bar{\bar{\mathcal{G}}}^* \right)$
  until *a criterion is met*
end

and the tensor $\bar{\bar{\mathcal{G}}}^*$ is a replication of the average tensor $\bar{\bar{\mathcal{G}}}$ along the mode $N+1$, that means $\bar{\bar{\mathcal{G}}}^*_k = \bar{\bar{\mathcal{G}}}$.

Combination of the learning rules (7.52) and (7.56) gives us the new multiplicative algorithm for finding discriminant basis factors $\mathbf{A}^{(n)}$ and features $\mathcal{G}$; the pseudo-code of this algorithm is given in Algorithm 7.4. Note that, factors $\mathbf{A}^{(n)}$ always need to be normalized to unit-length vectors, but is not explicitly listed in this algorithm.

An alternative approach to find the discriminant basis factors is to regularize factors $\mathbf{A}^{(n)}$ by the between-class and within-class scatter matrices. It is interesting to note that the features to classify objects can be obtained by a simple projection:

$$\mathcal{F}^{(k)} = \mathcal{X}^{(k)} \times_1 \mathbf{A}^{(1)T} \times_2 \mathbf{A}^{(2)T} \cdots \times_N \mathbf{A}^{(N)T} = \mathcal{X}^{(k)} \times \{\mathbf{A}^T\}. \tag{7.58}$$

In the case of orthogonal bases (with HOOI or HODA algorithms), feature tensors $\mathcal{F}^{(k)}$ are exactly the core tensors $\mathcal{G}^{(k)}$. However, this kind of projection can also be applied for nonnegative bases.

For such case, the two regularization terms for the discriminant in the cost function (7.50) are now computed based on $\mathcal{F}^{(k)}$ instead of $\mathcal{G}^{(k)}$. By taking into consideration that the scatter matrices $\mathbf{S}_w$ and $\mathbf{S}_b$ are not constants with respect to factors $\mathbf{A}^{(n)}$, and their relation to factors $\mathbf{A}^{(n)}$ is given by (7.45), their partial derivatives are given by

$$\frac{\partial \mathrm{tr}\left[\mathbf{S}_w\right]}{\partial \mathbf{A}^{(n)}} = \frac{\partial \mathrm{tr}\left[\mathbf{A}^{(n)\,T} \mathbf{S}_w^{-n} \mathbf{A}^{(n)}\right]}{\partial \mathbf{A}^{(n)}} = 2\, \mathbf{S}_w^{-n} \mathbf{A}^{(n)}, \tag{7.59}$$

$$\frac{\partial \mathrm{tr}\left[\mathbf{S}_b\right]}{\partial \mathbf{A}^{(n)}} = \frac{\partial \mathrm{tr}\left[\mathbf{A}^{(n)\,T} \mathbf{S}_b^{-n} \mathbf{A}^{(n)}\right]}{\partial \mathbf{A}^{(n)}} = 2\, \mathbf{S}_b^{-n} \mathbf{A}^{(n)}, \tag{7.60}$$

where the symmetric scatter matrices $\mathbf{S}_w^{-n}$ and $\mathbf{S}_b^{-n}$ are expressed via tensor contracted products

$$\mathbf{S}_w^{-n} = \langle \tilde{\boldsymbol{\mathcal{X}}} \times_{-(n,N+1)} \{\mathbf{A}^T\}, \tilde{\boldsymbol{\mathcal{X}}} \times_{-(n,N+1)} \{\mathbf{A}^T\} \rangle_{-n}, \tag{7.61}$$

$$\mathbf{S}_b^{-n} = \langle \check{\boldsymbol{\mathcal{X}}} \times_{-(n,N+1)} \{\mathbf{A}^T\}, \check{\boldsymbol{\mathcal{X}}} \times_{-(n,N+1)} \{\mathbf{A}^T\} \rangle_{-n}. \tag{7.62}$$

With the tensors $\tilde{\boldsymbol{\mathcal{X}}}$ and $\check{\boldsymbol{\mathcal{X}}}$ given in (7.39) and (7.42), based on the multiplicative learning rule for $\mathbf{A}^{(n)}$ [47;103;131;157] a new learning rule for $\mathbf{A}^{(n)}$ can be derived as

$$\begin{aligned} \mathbf{A}^{(n)} \leftarrow\ & \mathbf{A}^{(n)} \circledast \left(\langle \boldsymbol{\mathcal{X}} \times_{-(n,N+1)} \{\mathbf{A}^T\}, \boldsymbol{\mathcal{G}} \rangle_{-n} + \lambda_b \, \mathbf{S}_b^{-n} \, \mathbf{A}^{(n)} \right) \oslash \\ & \left(\mathbf{A}^{(n)} \langle \boldsymbol{\mathcal{G}}, \boldsymbol{\mathcal{G}} \times_{-(n,N+1)} \{\mathbf{A}^T \mathbf{A}\} \rangle_{-n} + \lambda_o \, \mathbf{A}^{(n)} \left(\mathbf{1}\mathbf{1}^T - \mathbf{I} \right) + \lambda_w \, \mathbf{S}_w^{-n} \, \mathbf{A}^{(n)} \right). \end{aligned} \tag{7.63}$$

To reduce the computational complexity, the scatter matrices $\mathbf{S}_w^{-n}$ and $\mathbf{S}_b^{-n}$ should be derived from the tensor contraction $\boldsymbol{\mathcal{X}} \times_{-(n,N+1)} \{\mathbf{A}^T\}$. The core tensor $\boldsymbol{\mathcal{G}}$ is updated using the multiplicative rule (7.56).

7.5 Feature Ranking and Selection

The number of features L of a multiway sample in a classification problem strongly depends on the number of basis components R_n of the factors $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ in tensor decomposition. The smaller the values R_n, the smaller the number of features. The number of components R_n for factor $\mathbf{A}^{(n)}$ can be defined by the number of dominant eigenvalues of the contracted product $\mathbf{X}_{(n)}\mathbf{X}_{(n)}^T = \mathbf{U}\,\boldsymbol{\Lambda}\,\mathbf{U}^T \in \mathbb{R}^{I_n \times I_n}$, where $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{I_N})$, and $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_{R_n} \geqslant \cdots \geqslant \lambda_{I_N}$ are eigenvalues. The factors should explain the whole training data at least above the threshold fitness θ (typical value $\theta = 95\%$)

$$\arg\min_{R_n} \frac{\sum_{r=1}^{R_n} \lambda_r}{\sum_{r=1}^{I_n} \lambda_r} > \theta. \tag{7.64}$$

For the TUCKER decomposition, we can find R_n by using the heuristic rule in (7.64), for each factor in each mode (see Algorithm 7.5).

For interactive bases methods, the total number of features is $L = R_1 \times R_2 \times \cdots \times R_N$. Although the number of features is reduced and is much smaller than the number of samples of the raw data, its value is still large and dramatically increases with the data dimension. For example, for images of size of 100×100 pixels, which are compressed by two interactive factors of 10 columns, their reduced versions have size of 10×10, and hence have still 100 features.

In practice, we do not need to use all the features from the core tensors but only some significant features, without sacrificing the accuracy via some information ranking criteria such as correlation score, minimum-redundancy-maximum-relevance selection, Fisher score, Laplacian score, and entropy[97]. Information indices for all the features are calculated, and then sorted in a descending order. Significant features corresponding to the largest indices should be chosen first. Features with small score indices can be neglected without affecting the performance.

Algorithm 7.5: Initialization for Basis Factors

input : $\mathcal{X}$: tensor of K training samples $I_1 \times I_2 \times \cdots \times I_N \times K$
output: $\mathbf{A}^{(n)}$: N factors $I_n \times R_n$

begin
  parfor $n = 1$ *to* N **do** // parallel loop
    $[\mathbf{A}^{(n)}, \mathbf{\Lambda}] = \texttt{eig}\,(\langle \mathcal{X}, \mathcal{X} \rangle_{-n})$
    $[\boldsymbol{\lambda}, \boldsymbol{\zeta}] = \texttt{sort}(\text{diag}(\mathbf{\Lambda}))$ // sort $\boldsymbol{\lambda}$ in descending order
    $R_n = \arg\min \frac{\sum_{r=1}^{R_n} \lambda_r}{\sum_{r=1}^{I_n} \lambda_r} > \theta$
    $\mathbf{A}^{(n)} = \mathbf{A}^{(n)}_{\zeta_{1:R_n}}$
  endfor
end

Using validation data, we can analyze the effect of the number of features on the achieved accuracy. The number of dominant features can be found so that the desired accuracy changes according to an acceptance tolerance during the validation. As a consequence, this step removes redundant features. For TUCKER decompositions, we note that a major feature $g_{r_1,r_2,\ldots,r_N}$ is the coordinate value of the tensor data explained by the base vectors $\boldsymbol{a}^{(1)}_{r_1}, \boldsymbol{a}^{(2)}_{r_2}, \ldots, \boldsymbol{a}^{(N)}_{r_N}$. Thus, the components which are not involved in any major features can be ignored in order to to reduce the factor dimensions.

A convenient method to rank a feature is based on the Fisher ratios (scores) of features defined as

$$\varphi(i) = \frac{\sum_{c=1}^{C} K_c \, (\overline{g}_i^{(c)} - \overline{\overline{g}}_i)^2}{\sum_{k=1}^{K} (g_i^{(k)} - \overline{g}_i^{(c_k)})^2}, \qquad (i = 1, 2, \ldots, L), \tag{7.65}$$

where $g_i^{(k)}$ is the i-th entry (feature) of the vectorized version of the core tensor $\mathcal{G}^{(k)}$, $c_k = 1, 2, \ldots, C$ denotes the class to which the training sample $\mathcal{X}^{(k)}$ belongs, and K_c is the number of training samples in the c-th class. The c-th class mean sample of the t-th feature $\overline{g}_i^{(c)}$, $i = 1, \ldots, L, c = 1, \ldots, C$, and the total mean feature $\overline{\overline{g}}_i$ are respectively defined as

$$\overline{g}_i^{(c)} = \frac{1}{K_c} \sum_{k \in \mathcal{I}_1} g_i^{(k)}, \qquad \overline{\overline{g}}_i = \frac{1}{K} \sum_{k=1}^{N} g_i^{(k)}. \tag{7.66}$$

After ranking the features in a descending order of their Fisher scores, significant features should be chosen to classify the sample.

7.6 Feature Extraction

Feature extraction corresponds to projecting the data sample $\mathring{\mathcal{X}}^{(t)} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}$ onto a feature subspace spanned by an available set of basis factors $\mathbf{A}^{(n)}$ or $\mathbf{U}^{(n)}$. In a general case, this problem is stated as follows.

Problem 7.5 (Feature extraction)
Feature extraction of a tensor $\mathring{\mathcal{X}}^{(t)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ *with a set of given bases* $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ *is to find the core tensor* $\mathring{\mathcal{G}}^{(t)} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ *in the TUCKER decomposition*

$$\mathring{\mathcal{X}}^{(t)} = \mathring{\mathcal{G}}^{(t)} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)}. \tag{7.67}$$

For general bases $\mathbf{A}^{(n)}$, Problem (7.5) can be explicitly solved by products of matrix inverses

$$\mathring{\mathcal{G}}^{(t)} = \mathring{\mathcal{X}}^{(t)} \times_1 \mathbf{A}^{(1)\dagger} \times_2 \mathbf{A}^{(2)\dagger} \cdots \times_N \mathbf{A}^{(N)\dagger}. \tag{7.68}$$

For orthogonal bases $\mathbf{A}^{(n)} = \mathbf{U}^{(n)}$, with $\mathbf{U}^{(n)\,T}\mathbf{U}^{(n)} = \mathbf{I}$, a core tensor $\mathring{\mathcal{G}}^{(t)}$ is easily obtained

$$\mathring{\mathcal{G}}^{(t)} = \mathring{\mathcal{X}}^{(t)} \times_1 \mathbf{A}^{(1)\,T} \times_2 \mathbf{A}^{(2)\,T} \cdots \times_N \mathbf{A}^{(N)\,T}. \tag{7.69}$$

For nonnegative bases $\mathbf{A}^{(n)}$, the core tensor $\mathring{\mathcal{G}}^{(t)}$ can be estimated by applying iterative (multiplicative) learning rule for the core tensor. Such iterations often converge quickly after few iterations.

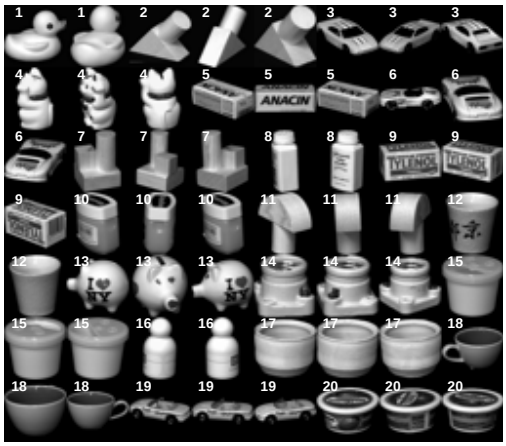
Although these methods are quite different, both approaches (7.68) and (7.69) in practice can also be used to retrieve the features for nonnegative bases. Of course, the features of the training data and the test data must be extracted by the same approach.

7.7 Image Classification - Dataset COIL20

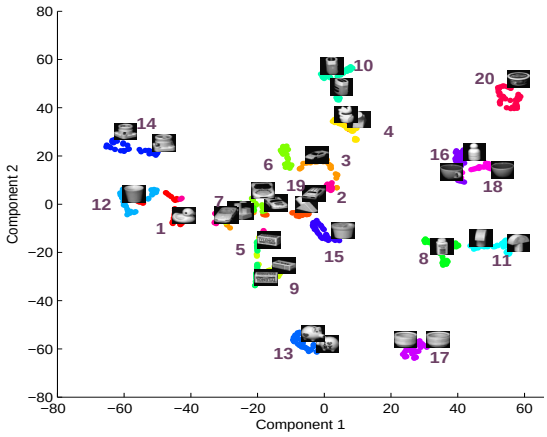
In the first set of simulations, we considered the Columbia University Image Library (COIL-20) dataset[134] consists of 1,420 grayscale images of 20 objects (72 images per object) with a wide variety of complex geometric and reflectance characteristics. Each image was downsampled to 32×32 grayscale (0-255). Figure 7.3(a) shows some sample images of this dataset. The dataset was randomly divided into two separate sets with 720 training and 700 test images. The results were averaged over 100 trials. The training data was constructed as a 3-D tensor of size $32 \times 32 \times 720$ images. We applied three methods to find basis factors with orthogonality (Algorithm 7.1), discriminant (Algorithm 7.3) and nonnegativity (Algorithm 7.4) constraints from the training tensor. For all the methods, $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$ were fixed to 10 components, hence, there were totally $F = 100$ features for each sample.

To classify the data, we trained an SVM classifier using the Gaussian Radial Basis Function kernel[31]. With the same hold/out ratio of 50%, NTD, Orthogonal TUCKER-2 and HODA-2 had almost perfect performance as shown in Table 7.1. The discriminant factors were estimated by solving the trace difference problem[209]. Orthonormal factors achieved the highest accuracy of 99.96%.

The hold/out ratio was next verified at 3 additional levels of 80%, 90% and 95%. As the hold/out ratio was 90%, there were 160 samples for training: 8 samples per class, and 1260 samples for test. Classification with nonnegative factors obtained an average accuracy of 94.78%. HODA with trace-ratio method (HODA-T) achieved 96.26% accuracy, whereas by solving the ratio trace problem with



(a) Randomly selected samples of the COIL-20 dataset.



(b) Visualization of some selected objects from the COIL-20.

Figure 7.3: Visualization of the COIL-20 dataset and distribution of its features by t-SNE components. Digits represent different classes of objects

Table 7.1: Classification Performance for The COIL-20 Dataset. Comparison of accuracy of methods using the SVM classifier for different hold/out ratios. Feature core tensors had size of 10×10.

Method	50%		80%		90%		95%	
	Test	N_f	Test	N_f	Test	N_f	Test	N_f
NTD-2	99.94±0.22	36	98.27±0.81	68	94.78±1.38	68		
Orth. TUCKER-2	**99.96±0.13**	20	**99.03±0.65**	32	**97.18+1.09**	32	**91.76±1.96**	36
HODA-T	99.90±0.19	21	98.65±0.69	24	96.26±1.09	24	89.77±2.33	24
HODA-G (GEVD, $\alpha = 0.001$)					97.12±1.18	36	91.41±2.03	32
HODA-G (GEVD, $\alpha = 1$)					82.05±2.11	46	74.35±3.11	40

GEVD this algorithm (HODA-G) achieved only 82.05% accuracy. The regularized HODA algorithm (7.49) with $\alpha = 0.001$ achieved much better accuracy of 97.12%. We note that for the regularization parameter $\alpha = 0$, HODA simplifies to HOOI, which gave the highest average accuracy of 97.18%. The detailed results are listed in Table 7.1.

The nonnegative components explain the data as common parts over the samples, the orthonormal factors try to explain the data at a highest fitness, whereas the discriminant factors focus on differences between the samples. Due to different physical meanings of decomposition, the number of necessary significant features for the three approaches are quite different. It is obvious that classification with nonnegative factors requires more components than those of the other methods. For the same accuracy level, the discriminant factors often need less significant features than those of others. For the hold/out ratio of 50%, classification with nonnegative factors required 36 significant features to achieve the highest performance of 99.94%, while that with orthogonal factors needed 20 significant features. For the hold/out ratio of 80%, the number of selected significant features for nonnegative, orthogonal, and discriminant bases are 68, 32 and 24, respectively.

From the extracted features, the dataset was visualized via two t-SNE components[207] shown in Figure 7.3(b) illustrating good separation. Objects that have a high similarity are located in close proximity to each other in the scatter plot, such as classes 5 and 9, classes 3, 6 and 19, whereas objects that have low similarity are located far from each other, for example classes 13, 14, 17, 20.

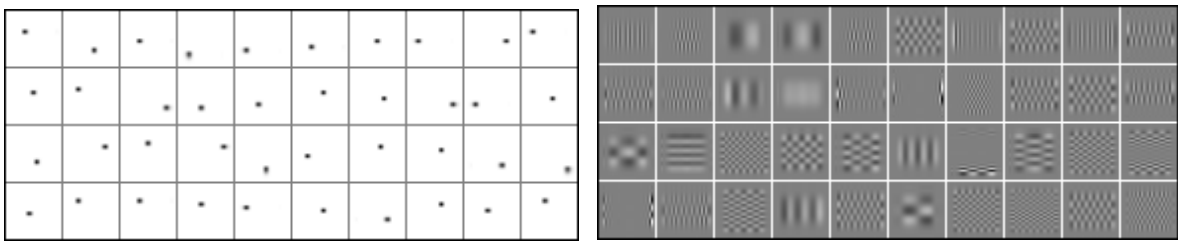
7.8 Classification of Handwritten Digits

In the second set of simulations, we factorized and classified the MNIST data set of images of handwritten digits (0-9)[112]. This data set is composed 60,000 training images and 10,000 testing images. Each image is a 28 $\times$ 28 grayscale (0-255) labeled representation of an individual digit. In Figure 7.5(a), we present 100 randomly selected handwritten digits. Some efficient methods can process this large data such as Jiang *et al.*[99], Horio and Yamakawa[93]. In this example, we selected a small subset of digits to illustrate our models and algorithms.

In the training stage, we decomposed the data to find two TUCKER basis factors. Orthonormal and discriminant factors were set to explain 99.9% of the training tensor, whereas both nonnegative factors had 10 components. Nonnegative factors gave a classification accuracy of 97.66%, with 78 significant features. Those features were coefficients of 48 compositions of basis components. Each composition of two components formed a basis image, that is, a digit image was considered as a summation of basis images. Figure 7.4(a) displayed the first 40 nonnegative basis images. A basis image expresses a part of a digit in which grey pixel denotes positive value and zero by bright one.

In Figure 7.4(b), we displayed 40 basis images generated by discriminant components. With those bases, the dataset was classified with 98.39% accuracy using the SVM classifier. Discriminant basis images are rank-one matrices with particular structures. Grey parts correspond to zeros, whereas negative and positive values for dark and bright elements respectively.

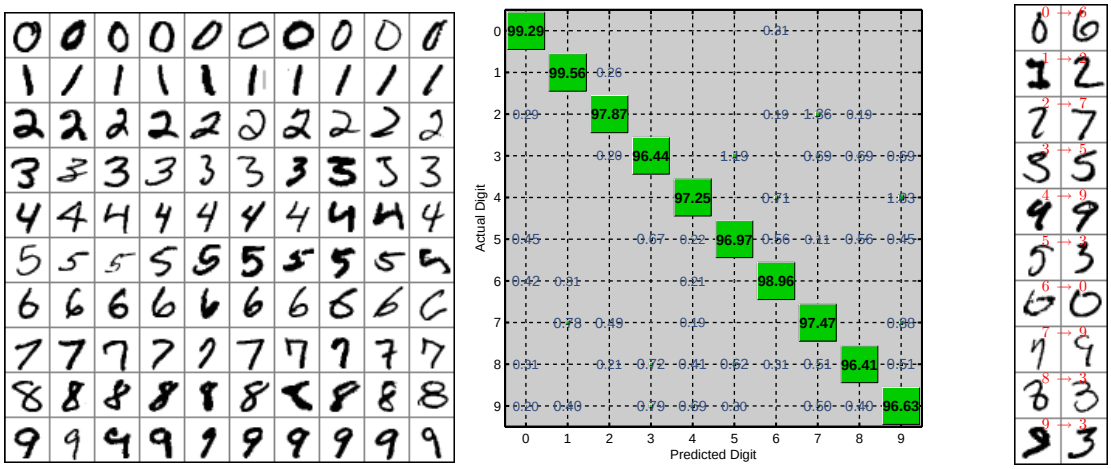
Table 7.2(a) shows the performance of our methods. Orthonormal factors achieved a 97.32% accuracy with 30 significant features. All performances were verified with the KNN-3 classifier, except the last method using SVM. There was not much difference between these two classifiers. In order to illustrate more clearly the classification rates of digits, in Figure 7.5(b), we show the Hinton graph of the confusion matrix using the KNN-3 classifier. The volume of box is proportional to the intensity of a corresponding prediction rate. A diagonal coefficient indicates the classification accuracy for each digit. Whereas other entries express the misclassification (error) rates. For example, digits 0 and 1 were classified with high accuracies ($>$ 99%). A digit 3 may be potentially misclassified as one of digits 5, 7, 8, 9. In Figure 7.5(c), we show test samples 418, 1232, 2151, 3248, 4389, 5549, 6067, 7059, 8197, 9390 misclassified as other ones using the KNN-3 classifier.



(a) Forty dominant basis images composed of nonnegative factors.

(b) Basis images of discriminant factors for 200 digit images.

Figure 7.4: Visualization of basis images for the hand-written digit images.



(a) Randomly selected handwritten digits from the MNIST dataset.

(b) Confusion matrix for the classification with ten digit categories.

(c) Some misclassified digits.

Figure 7.5: Visualization of classification of ten digit classes: (a) 100 handwritten digits randomly selected from the dataset; (b) Hinton graph of the confusion matrix using the KNN-3 classifier; (c) ten digits for ten classes were misclassified as other digits using the KNN-3 classifier.

An alternative efficient approach for this problem is to classify the Gabor features of digit images. We computed 24 Gabor features for each image consisting of 8 orientations at 3 scales. That means 2-D samples (images) were augmented dimensionality to become 3-D tensors. Gabor features were down-sampled to $16 \times 16 \times 24$ dimensional sample tensors $\boldsymbol{\mathcal{X}}^{(k)}$. Hence, both the training and test data were 4-D tensors. To illustrate the performance of this approach, we chose only 20 first samples for each digit for both training and test data. For classification of the raw data (images), both orthonormal and discriminant factors provide 89.50% accuracy with 24 and 14 significant features, respectively (see in Table 7.2(b)). For the Gabor tensor, we decomposed the training tensor into 3 factors with sizes 16×10, 16×10, and 24×23. Classification of a set of 41 significant features on the subspace of orthonormal factors (TUCKER-3) with the SVM classifier achieved an accuracy of 93%. The same procedure with 32 discriminant significant features provided an accuracy of 94.50%; the results are given in Table 7.2(b). The values corresponding to highest performance are given in brackets.

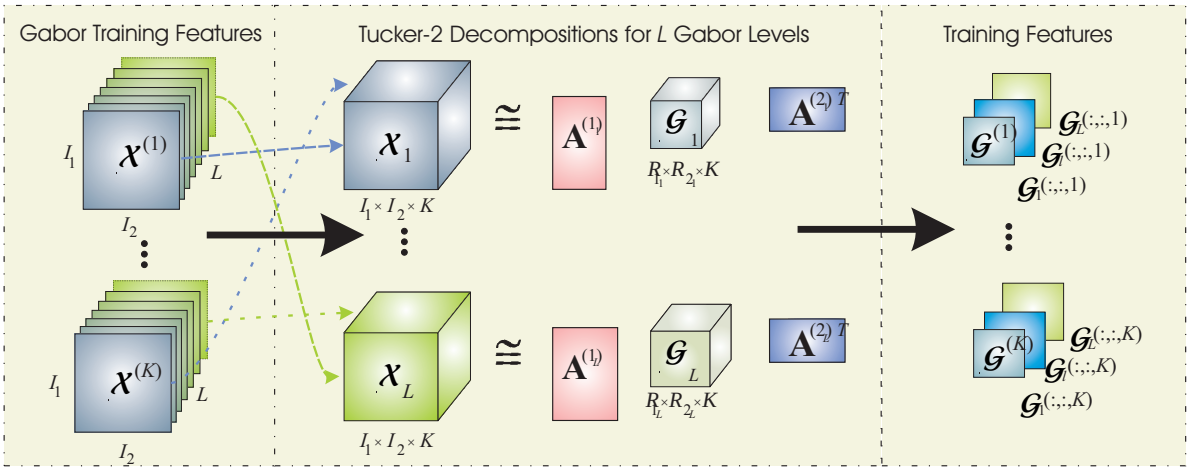


Figure 7.6: Conceptual model illustrating feature extractions for handwritten digit images using Gabor filters and multiple projection filters based on simultaneous TUCKER-2 decompositions.

The classification accuracy can be considerably improved by taking into account that there is low correlation among them, or rare common parts between Gabor features which are not in the same levels (orientations and scales). Hence, instead of decomposition of the Gabor training tensors along all the three modes, we should find common bases only for the two first dimensions. Due to this reason, we split the training tensor $\mathcal{X}$ into 24 3-D sub-tensors $\mathcal{X}_l = \mathcal{X}(:,:,l,:) \in \mathbb{R}^{16\times 16\times K}$ $(l = 1,2,\ldots,24)$ which contain the l-th frontal slices of tensors $\mathcal{X}^{(k)}$ for $k = 1,2,\ldots,K$ with $K = 200$ training samples. For each specific Gabor level (orientation and scale) $(l = 1,2,\ldots,24)$, we found two basis factors $\mathbf{A}^{(1_l)} \in \mathbb{R}^{16\times R_{1l}}$, and $\mathbf{A}^{(2_l)} \in \mathbb{R}^{16\times R_{2l}}$ via a TUCKER-2 decomposition

$$\mathcal{X}_l \approx \mathcal{G}_l \times_1 \mathbf{A}^{(1_l)} \times_2 \mathbf{A}^{(2_l)}, \quad (l = 1,2,\ldots,L) \tag{7.70}$$

in which $\mathcal{G}_l$ is an $R_{1l} \times R_{2l} \times K$ dimensional tensor whose k-th frontal slices represent compressed features of the samples $\mathcal{X}^{(k)}$ at level l.

From $L = 24$ decompositions (7.70) for all the levels, we obtained $L = 24$ sets of bases $\mathbf{A}^{(1_l)}$ and $\mathbf{A}^{(2_l)}$, and $L = 24$ core tensors $\mathcal{G}_l$, $l = 1,2,\ldots,L$. Therefore, features of the multiway training sample $\mathcal{X}^{(k)}$ are represented by 32 k-th frontal slices of the core tensors $\mathcal{G}_l$ that can be expressed as a concanated vector $\boldsymbol{g}_k$ of $\sum_{l=1}^{L=32} R_{1l} R_{2l}$ entries

$$\boldsymbol{g}^{(k)} = [\text{vec}(\mathbf{G}_1(:,:,k)) ; \text{vec}(\mathbf{G}_2(:,:,k)) ; \ldots ; \text{vec}(\mathbf{G}_L(:,:,k))] . \tag{7.71}$$

The whole training procedure is illustrated in Figure 7.6. Features of a test sample $\mathring{\mathcal{X}}^{(t)} \in \mathbb{R}^{16\times 16\times 24}$ $(t = 1,2,\ldots,200)$ can be obtained by projecting each frontal slice $\mathring{\mathbf{X}}_l^{(t)}$ of this tensor onto the corresponding feature subspace spanned by bases $\mathbf{A}^{(1_l)}$ and $\mathbf{A}^{(2_l)}$, for $l = 1,2,\ldots,L$, described as

$$\mathring{\mathbf{G}}_l^{(t)} = \mathring{\mathbf{X}}_l^{(t)} \times_1 \mathbf{A}^{(1_l)\,T} \times_2 \mathbf{A}^{(2_l)\,T} = \mathbf{A}^{(1_l)\,T} \mathring{\mathbf{X}}_l^{(t)} \mathbf{A}^{(2_l)}, \quad (l = 1,2,\ldots,L) . \tag{7.72}$$

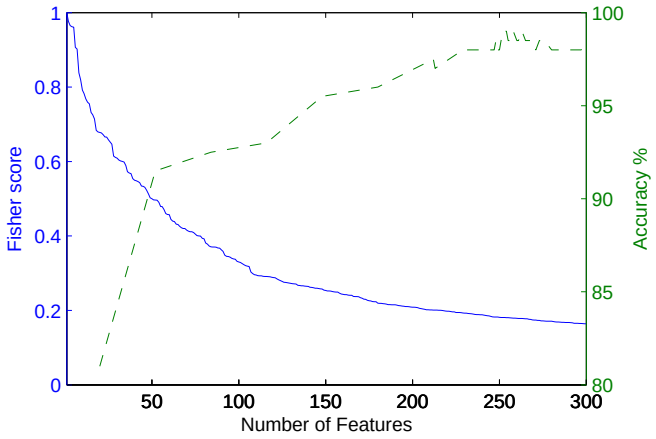


Figure 7.7: Classification accuracy for the hand-written digit dataset achieved with multiple projection filters for Gabor levels.

Table 7.2: Comparison of classification performance for the hand-written digit dataset. For Gabor samples, only first 20 samples for each digit were taken for both training and test data.

(a) Samples are raw images.

Method	Accuracy	N_f
NTD-2	97.66	78
Orth. TUCKER-2	97.32	30
HODA-2	97.71	52
HODA-2	**98.39** (SVM)	40

(b) Samples are 3-D Gabor features.

Method	Raw data		Gabor features	
	Accuracy	N_f	Accuracy	N_f
NTD	90	50		
TUCKER	89.50	24	93	41
HODA	89.50	14	**94.50**	32 (26)

Those features can also be expressed in the vector form as

$$\mathring{\boldsymbol{g}}^{(t)} = \left[\text{vec}\left(\mathring{\mathbf{G}}_1^{(t)}\right); \text{vec}\left(\mathring{\mathbf{G}}_2^{(t)}\right); \ldots; \text{vec}\left(\mathring{\mathbf{G}}_{24}^{(t)}\right)\right] . \tag{7.73}$$

For the model of multiple projection filters, we employed the HODA algorithm, and set the number of basis components to 12. The classification accuracy achieved **98.5**% for 252 significant features. Accuracies and Fisher scores of 300 significant features are shown in Figure 7.8.

The paradigm given in Figure 7.6 can be applied for classification of other image datasets. For example, for the ORL face database[121], we constructed Gabor feature tensors of 8 orientations at 4 scales $\boldsymbol{\mathcal{X}}^{(k)} \in \mathbb{R}^{16\times16\times32}$. With a hold/out ratio of 50%, classification of such tensors achieved an average accuracy of 99.32%, 99.29%, and 99.27% over 100 runs using the HODA-G, HODA-T, and HOOI algorithms, respectively.

7.9 Scenes Classification

Recognition of the scene implies providing information about the semantic category and the function of the environment. This problem has an important role in modern digital cameras, and robot vision. For

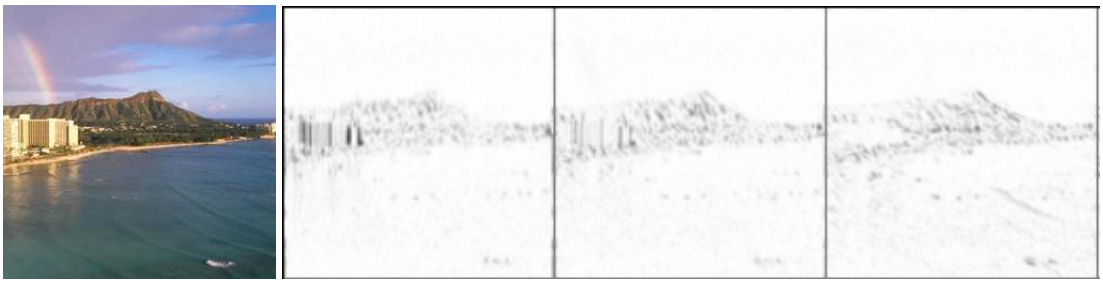
automatic imaging system, scene information helps to achieve more accurate autofocus, auto exposure and auto white balance control prior to capture. The Scene Recognition System integrated in the Nikon D3 and D300 digital SLR cameras[138] uses information from the 1,005-pixel RGB sensor to recognize a subject or scene, that enables highly precise exposure control utilizing color information. Intelligent Scene Recognition that features on SONY Cyber-shot models takes the guesswork out of adjusting digital camera's settings for beautiful results in a range of common shooting situations.

In computational vision, some experimental studies have suggested that recognition of real world scenes may be initiated from the encoding of the global configuration, ignoring most of the details and object information[20;21]. The primary semantic representation appears to be built on a low resolution spatial configuration. In this direction, Oliva and Torralba[141] proposed Spatial Envelope to express "shape of a scene" using a few perceptual dimensions . Scene structure can be characterized by global features which are often constructed from Gabor features in multiple orientations and scales. A set of features in specific orientations and scales indeed represent a matrix. Hence, Gabor features for an image establish a 3-D tensor.

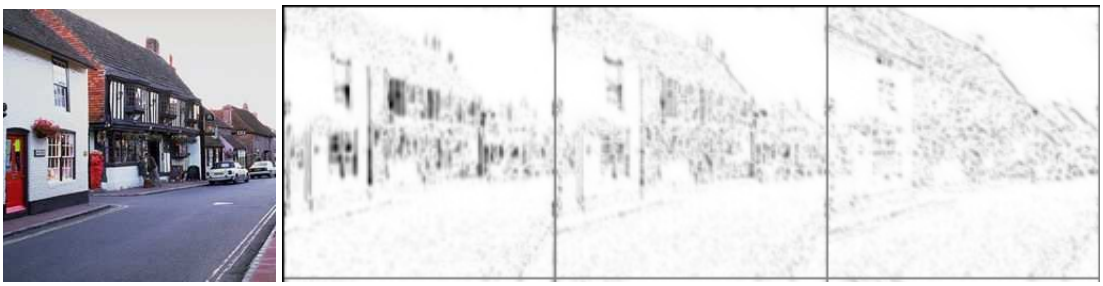
We use the same dataset analyzed by Oliva and Torralba[141], available at website http://people.csail.mit.edu/torralba/code/spatialenvelope. There are 2.600 color images of 256x256 pixels classified into 8 outdoor scene categories: coast, mountain, forest, open country, street, inside city, tall buildings and highways. Oliva and Torralba[141] proposed to use global features to classify this dataset. In fact, we disregard chrominance channels, employ only the luminance channel of images. Global features are constructed from Gabor features consisting of 8 orientations at 4 scales. That means 2-D samples (images) are augmented dimensionality to become 3-D tensors whose each frontal slice consists of Gabor features at a specific orientation and level. Figure 7.8 illustrates Gabor features for some sample images. Gabor features are down-sampled to $16 \times 16 \times 32$ dimensional tensors before extracting discriminant features. Training data consists of 100 samples per class randomly selected from the whole data. Test data are the rest samples. We denote the 4-D training data by $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times K}$, $I_1 = 16, I_2 = 16, I_3 = 32$ which consists of $K = 800$ 3-D samples $\mathcal{X}^{(k)} \in \mathbb{R}^{16 \times 16 \times 32}$ for $k = 1, 2, \ldots, 800$.

A common method to deal with high dimensional data is to treat them as 1-D samples. In this direction, Oliva and Torralba[141] vectorized all global features $\mathcal{X}^{(k)}$, and converted data tensors to a matrix $\mathbf{X}_{(4)}$ of 800 *scences* $\times$ 8,192 *features*. Linear Discriminant Analysis can be applied to seek discriminant projection for this 2-D data.

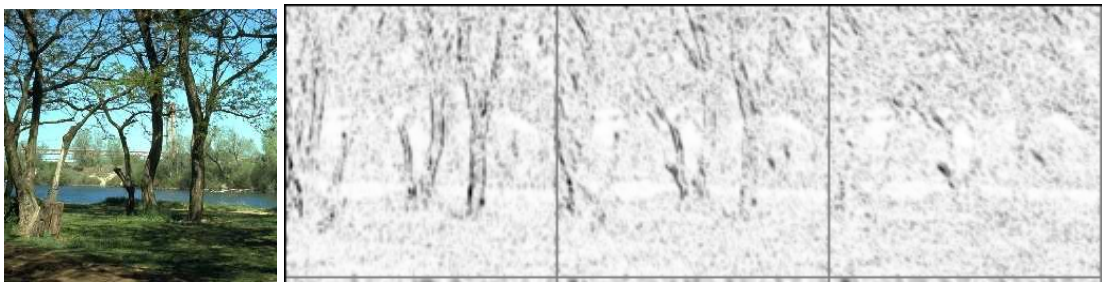
We employ the similar diagram in the previous example for this dataset. Instead of decomposition along all the three modes, we find common bases only for the first two dimensions. We split the training tensor $\mathcal{X}$ into 32 three dimensional sub-tensors $\mathcal{X}_l = \mathcal{X}(:,:,l,:) \in \mathbb{R}^{I_1 \times I_2 \times K}$ $(l = 1, 2, \ldots, 32)$ which consist of l-th frontal slices of tensors $\mathcal{X}^{(k)}$ for $k = 1, 2, \ldots, K$, and extract features as in (7.70),



(a) Coast scene



(b) Urban space



(c) Forest scene

Figure 7.8: Visualization of some randomly chosen scenes from the dataset with their Gabor features in the 3 first orientations.

(7.71), (7.73)[163].

The classification paradigm can be generally described in Figure 7.6. We verified the classification performance with feature extraction using the Linear Discriminant Analysis for the vectorized Gabor training features, and using the multiway decomposition. All the nonnegative factors were set to have 6 components. The results were averaged over 100 trials, and are given in Table 7.9(b). All the methods used the SVM classifier with the Gaussian Radial Basis Function kernel[31]. The multiway-based approaches achieved an average accuracy of 85.06% with discriminant bases, and 84.92% with nonnegative bases. Those results were improved by at least 2.4% compared with the LDA approach.

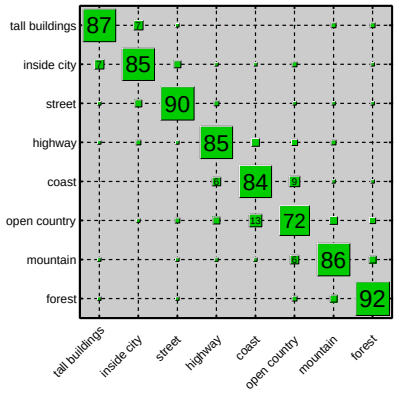
The confusion matrix shows the average classification results for the HODA algorithm in Table 7.9(a). The classification accuracy is also illustrated using Hinton diagram of the confusion matrix in Figure 7.9. Each blob at a specified position (m, n), for $1 \leqslant m, n \leqslant 8$ in the Hinton diagram has its

Category	1	2	3	4	5	6	7	8
1. Tall buildings	**87.05**	7.09	1.48	0.17	0.38	0.36	1.75	1.72
2. Inside city	6.73	**84.78**	4.05	1.00	1.06	1.57	0.21	0.60
3. Street	1.24	4.37	**89.55**	2.37	0.01	0.73	1.05	0.68
4. Highway	0.59	2.28	1.39	**85.24**	5.20	2.78	2.26	0.26
5. Coast	0.03	0.06	0.01	5.65	**83.77**	8.65	1.16	0.67
6. Open country	0.46	1.00	1.70	4.07	12.96	**71.84**	4.81	3.15
7. Mountain	1.03	0.11	0.75	1.05	1.36	5.92	**86.13**	3.65
8. Forest	0.86	0.23	0.70	0.18	0.07	1.60	4.21	**92.14**

(a) Confusion Matrix for HODA method

Method	LDA	NTD-2	HODA-2
Accuracy (%)	82.64 $\pm$ 0.80	84.92 $\pm$ 0.71	85.06 $\pm$ 0.75

(b) Comparison of the classification methods.



(c) Hinton diagram of the confusion matrix.

Figure 7.9: Confusion Matrix and its Hinton diagram show the accuracy of scene classification.

size proportional to the rate in which class m is classified into class n. Dominant entries which are mostly on the diagonal of the confusion matrix indicate the validity of the classification.

7.10 BCI Motor Imagery Classification

In the next set of simulations, we considered the classification and single trial recognition for BCI EEG data involving left/right motor imagery (MI) movements. Exemplary process shown in Figure 7.10 illustrates how to organize brain wave into tensor and decompose multidimensional training data tensor into factor matrices and core tensor representing reduced features and the process of feature extraction for test data[40;157;165]. EEG signals should be first passed over preprocesing stages such as artifact removal, bandpass filters, then transformed into spectral tensors uses bank of band pass filters, wavelets transforms. The tranforming stage could ultisize two or more different wavelets or alternative time frequency transforms. The training data tensor with a priori knowledge is constructed and tensor decomposition is preformed using constrained Tucker decomposition. In the next step projected filter is constructed and test data are projected to estimate reduced features. By comparison these features with labeled features (for training data) classification is performed. We performed experiments on three different BCI datasets, and compare performances of our model to that of the Common Spatial Pattern method (CSP)[149]. Examples in this section are provided in the NFEA toolbox[150].

7.10.1 Single Trial Recognition

The BCI EEG dataset analyzed in this section was recorded from 62 channels (with sampling frequency 500 Hz) with duration of 2 seconds with a 4 second break between the trials. The dataset[68] was recorded for 2 subjects and has 840 trials.

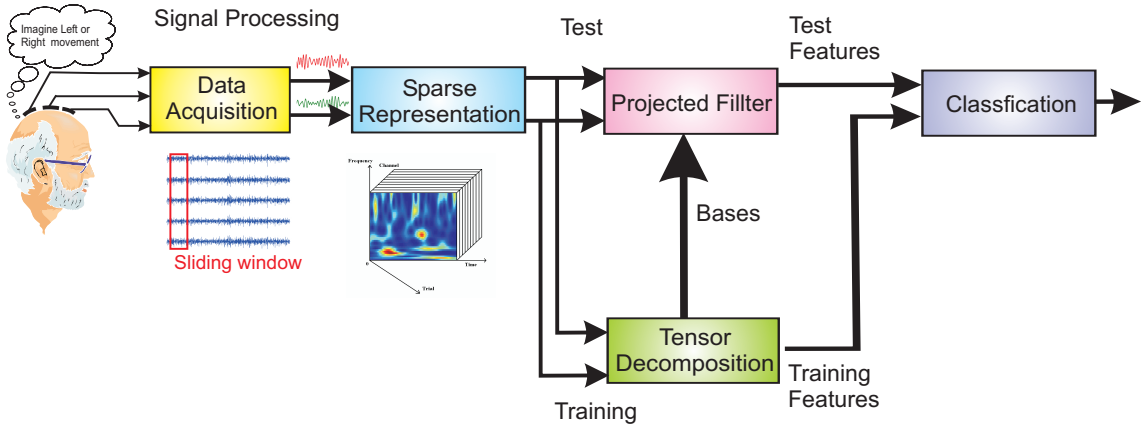


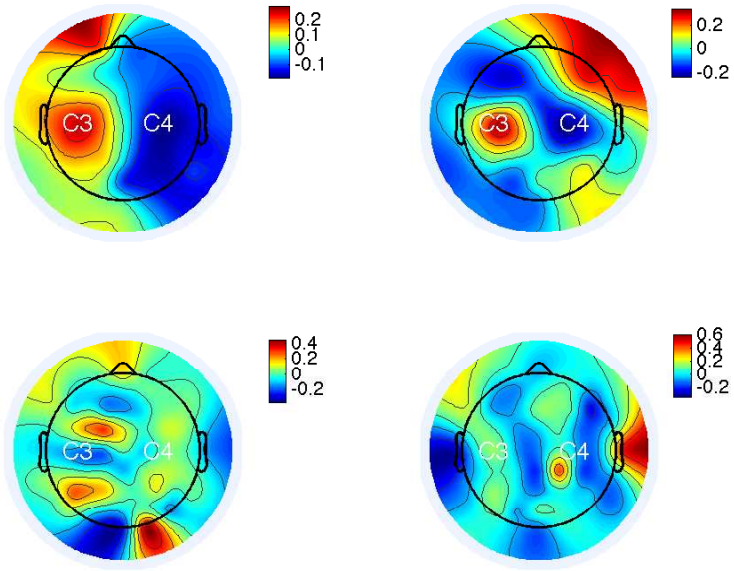
Figure 7.10: Conceptual diagram for BCI recognition based on multiway feature extraction[47].

For each subject, the data were collected over two sessions with a 15 minute break in between. The first session was conducted without feedback, and the 60 trials (30 trials for each class) obtained in this session were used for training and analysis. The second session consisted of 140 trials (70 trials for each class) as testing data to give online feedbacks. In the data collection stage, each subject was asked to sit in an armchair, keeping arms on the chair arms with two hands relaxing, and looking at a computer monitor at approximately 1m in front of the subject at eyes level. EEG signals were sampled at 500 Hz and preprocessed by a bandpass filter with cutoff frequencies of 8 Hz and 30 Hz.

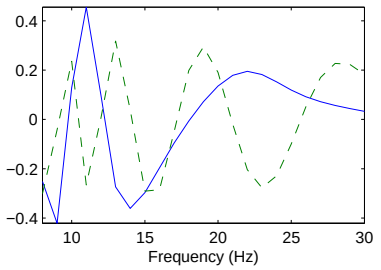
In the time domain each trial can be represented as a matrix of 62 channels $\times$ 1000 samples. For each subject the first 30 training trials belong to the left category and the rest training trials are for the right class. Similarly, the first half of test trials is designed for the left category, and the rest is assigned to the right class. The purpose was to find the labels corresponding to left or right hand imagery movements for all the test trials.

In preparation and imagination of movement the mu and beta rhythms are desynchronized over the contralateral primary sensorimotor area[147]. This phenomenon is known as Event-Related Desynchronization (ERD). In some subjects in addition to the contralateral ERD an ipsilateral Event-Related Synchronization (ERS) or a contralateral beta ERS following the beta ERD is found[147;148]. By convention, an ERD corresponds to a power decrease and an ERS to a power increase. For the right hand imagery movement, an ERD distributes over the left hemisphere and an ERS over the right hemisphere. On the contrary, for the left hand imagery movement the ERS/ERD phenomena occur on the left and right hemisphere, respectively.

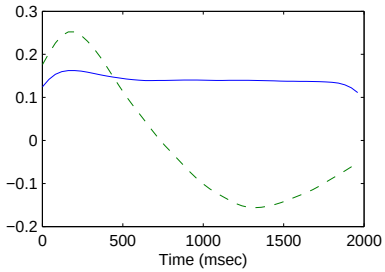
A popular classification method for such kind of dataset is the common spatial pattern method (CSP) with suitable preprocessing[149]. The performance obtained by using CSP achieved 82.86% and 90% for subject 1 and subject 2, respectively. In this section, we will present methods which improve



(a) Topographic map of the spatial orthogonal factor $\mathbf{U}^{(1)}$.



(b) Spectral orthogonal factor $\mathbf{U}^{(2)}$.



(c) Temporal orthogonal factor $\mathbf{U}^{(3)}$.

Figure 7.11: Visualization of some leading basis components obtained by orthogonal TUCKER-3 (HOOI algorithm) Example 7.10.1. (a) Topographic map built on 4 spatial components $\mathbf{U}^{(1)}$. The first two components (top) show the ERS/ERD cover strongly the motor cortex area indicated by blue and red regions; the next two components (bottom) did not express clearly the ERS/ERD but can improve the classification accuracy; (b) Spectral components indicate the most differences between two classes concentrate on the mu band (8-12 Hz) and the beta band (14-30 Hz); (c) Oscillations of the spectral component (b) were expressed by the temporal components $\mathbf{U}^{(3)}$.

dramatically the classification accuracy via tensor decompositions. Moreover, our approaches allow us to interpret ERD/ERS by some dominant components.

The key point of the enhancement methods is that the samples (trials) are augmented to become 3-D or 4-D tensors with additional modes. Transformation of EEG signals into the time-frequency domain is a standard technique to augment dimensionality. All the EEG signals were transformed into the time-frequency domain using the complex Morlet wavelets CMOR6-1 with the bandwidth parameter $f_b = 6$ Hz, and the wavelet center frequency $f_c = 1$ Hz. The data for each trial formed a 3-D spectral tensor with modes 62 *channels* $\times$ 23 *frequency bins* (8-30 Hz) $\times$ 50 *time frames*. That means the training data $\mathcal{X}$ is a 4-dimensional tensor of 120 3-D sub-tensors for two subjects and two classes: $62 \times 23 \times 50 \times 120$. The first 60 sub-tensors are for subject 1 and the next 60 sub-tensors are for subject 2. The test data were also organized in a similar way and consisted of 240 3-D sub-tensors (70 tensors/class/subject): 62 *channels* $\times$ 23 *frequency bins* $\times$ 50 *time frames* $\times$ 280 *trials*.

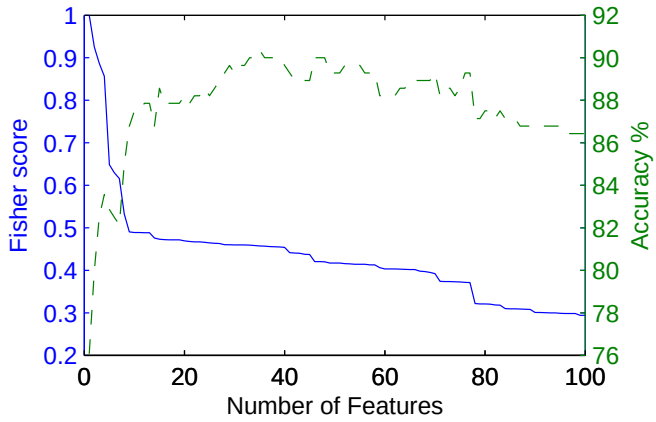


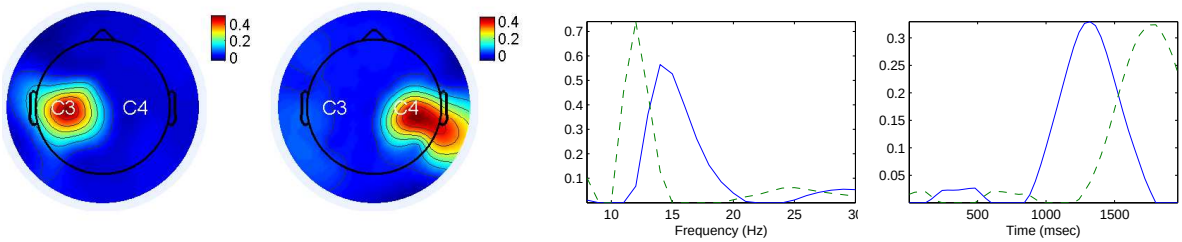
Figure 7.12: Fisher scores of orthogonal TUCKER-3 features and the classification accuracy of 90.36% with 35 significant features , 89.29% for subject 1 and 91.43% for subject 2.

7.10.1.1 Orthogonal Factors

Classification of the 3-D spectral tensors can be performed by the Orthogonal TUCKER-3 decomposition as described in Section 7.3.2. The HOOI algorithm was used to estimate 3 orthonormal factors $\mathbf{U}^{(n)}$, for $n = 1, 2, 3$ with the number of components set to explain 99% of the training data tensor. The decomposition resulted in the estimation of 3 factors $\mathbf{U}^{(n)}$ with sizes of 62×25, 23×11, and 50×12, respectively. Hence, there were in total $F = 25 \times 11 \times 12 = 3{,}300$ features compressed from 71,300 samples of the spectral tensors. This is a quite large number for classification of two categories.

In order to select a set of significant features, the features were ranked in a descending order of Fisher scores. Figure 7.10.1 shows 100 significant features with their scores (solid line) normalized so that the largest score became unity. We trained an SVM classifier using the Gaussian Radial Basis Function kernel[31]. The method achieved an average accuracy of 90.36% for 35 significant features (89.29% for subject 1, and 91.43 % for subject 2). This means we improved performance by 3.92% compared with that of CSP. Figure 7.10.1 also illustrated the classification accuracy verified for different numbers of significant features (from 1 to 100) on the right axes. The accuracy increased with the number significant features, then, decreased when using excessive number of features.

In Figure 7.11, we illustrated some dominant components that correspond to significant features. The 4 leading spatial components shown in Figure 7.11(a) indicate distributions of ERS/ERD phenomena over channels C3 and C4. The distributions gradually decreased for features with low scores. The two leading spectral components are shown in Figure 7.11(b). The first spectral component (solid line) indicates the major rhythm in the frequency ranges of the mu rhythm [8-13] Hz, and the beta rhythm [14-30] Hz.



(a) Topographic map of 2 spatial components $\mathbf{A}^{(1)}$. (b) Spectral components $\mathbf{A}^{(2)}$. (c) Temporal components $\mathbf{A}^{(3)}$.

Figure 7.13: Visualization of the dominant nonnegative components for Example 7.10.1: (a) two spatial components separately express the power distributions of EEG signals over channels C3 and C4. An event with high feature for the first component (channel C3) and low feature for the second component (channel C4) relates to the right hand imagery movement. On the contrary, this event relates to the left hand imagery movement; (b)-(c) Major rhythms are reflected by spectral components in the frequency range of [12-20] Hz and [10-15] Hz, and by the temporal components $\mathbf{A}^{(3)}$.

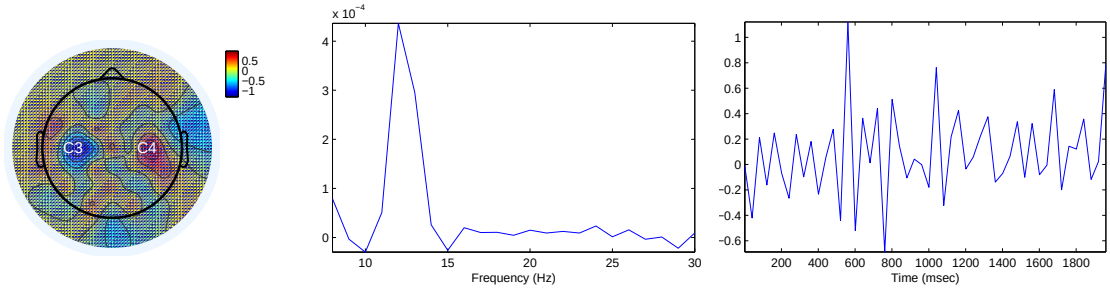
7.10.1.2 Nonnegative Factors

Since the EEG spectral tensor consists of nonnegative data, the decomposition of the spectral data into nonnegative common parts can often help in the classification and interpretation. This method is described in Sections 7.3.3 and 7.4.3. In this experiment, we estimated three nonnegative factors $\mathbf{A}^{(n)}$ for the 4-D training tensor. For this decomposition, the factors were set to have $R_1 = 10$ spatial components, $R_2 = 5$ spectral components and $R_3 = 5$ temporal components.

Classification of a set of 5 significant features achieved an average accuracy of 87.50% (85% for subject 1 and 90% for subject 2). In Figure 7.13, we illustrate basis components which related to the 5 significant features. Two spatial components in Figure 7.13(a) indicated that EEG power distributions over two channels C3 and C4 were separately decomposed. An event with high intensity for the first leading component (reflecting the channel C3), and low intensity for the second leading component (channel C4) relates to the right hand imagery movement. On the contrary, this event relates to the left hand imagery movement. The two leading spectral components in Figure 7.13(b) reflected main rhythm on the channels C3 and C4 and in the frequency range of 12-20 Hz. We can also discriminate the activations of these rhythms via temporal components $\mathbf{A}^{(3)}$ shown in Figure 7.13(c).

7.10.1.3 Discriminant Factors

In this section, we illustrate the classification using the discriminant factors approach. We note that the training data is a 4-D tensor, therefore, the HODA algorithm was set to find 3 discriminant basis factors $\mathbf{U}^{(n)}$, for $n = 1, 2, 3$. To solve the trace ratio problem (7.45), we used the general EVD approach. The number of components of factors set to explain 99% of the raw data returned the factors $\mathbf{U}^{(n)}(n = 1, 2, 3)$ with sizes of 62×25, 23×11, 50×12, respectively.



(a) Topographic map of $\mathbf{U}^{(1)}$. (b) Spectral factor $\mathbf{U}^{(2)}$. (c) Temporal factor $\mathbf{U}^{(3)}$.

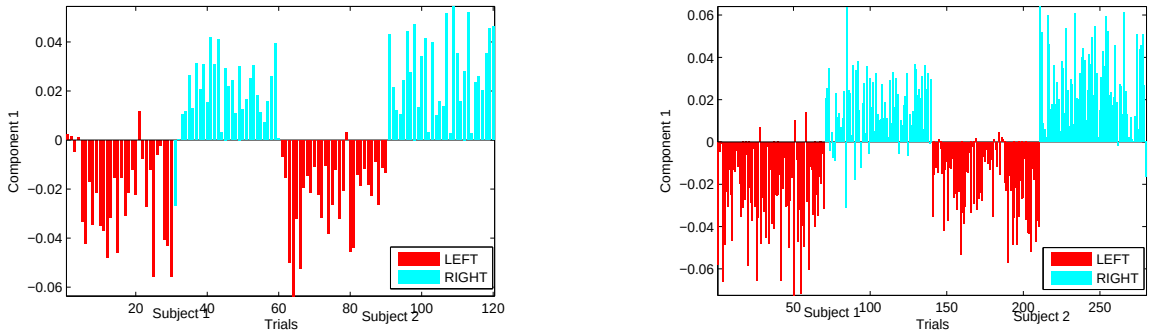
Figure 7.14: Visualization of the reduced discriminant basis factors for the BCI motor-imagery dataset. Factors had only one component after a proper selection. (a) Topographic map built on the spatial factor $\mathbf{U}^{(1)}$ shows the BCI motor-imagery data cover strongly the motor cortex area indicated by blue and red regions; (b) Spectral factor indicates the most differences between two classes on a frequency range of 10-15 Hz; (c) Oscillations of the spectral component (b) expressed by the temporal component $\mathbf{U}^{(3)}$.

The first leading feature corresponds to the first three basis components $\boldsymbol{u}_1^{(1)}, \boldsymbol{u}_1^{(2)}, \boldsymbol{u}_1^{(3)}$. With only one leading feature, we obtained an average accuracy for single trial recognition of 93.57%, (90.71% for subject 1, and 96.43% for subject 2). Both the k-nearest neighbor classifier ($k = 3$) and the SVM classifier gave the same performance.

The three dominant basis components are illustrated in Figure 7.14(a) for the topographic map of the spatial component, in Figure 7.14(b) for the spectral component, and in Figure 7.14(c) for the temporal component. The main differences between the two classes were characterized by oscillations in the frequency range $[10 - 15]$ Hz that strongly cover the motor cortex areas indicated by blue and red regions (channels C3 and C4) in Figure 7.14(a).

With only one basis component for each factor, each tensor sample can be expressed by only one feature. Hence, the training features form a vector of 120 entries, and the test features form a vector of 280 entries. Illustrations of two feature vectors are given in Figure 7.15(a) - 7.15(b). The EEG power for negative features was high for channel C3, and low for channel C4, hence, negative features were assigned to the left class. Similarly, positive features whose EEG powers were low for channel C3 and high for channel C4 should be assigned to the right class.

For this experiment, we need only one basis component for each factor to project the raw sample onto the feature space. However, the basis factors $\mathbf{U}^{(n)}$ cannot be forced to be a vector $R_n = 1$, for $n = 1, 2, 3$, or to explain the training data with a low threshold fitness θ defined in (7.64). For example, at $\theta = 80\%$, the numbers of components for three factors were 1, 2, 1 respectively, the classification performance achieved only a 68.93% accuracy. A factorization of this spectral tensor with a rank-one tensor achieved an accuracy of 66.43%. In Table 7.3, we analyzed the accuracy of the single trial recognition with threshold θ varying in the range of [80%, 99%]. The number of components increases as the fitness rate θ increases. The accuracy was increased by high threshold fitness θ.



(a) Discriminant features of 120 training samples.

(b) Discriminant features of 280 test samples.

Figure 7.15: Discriminant features of the BCI dataset using only one basis component for each factor.

7.10.1.4 Augmentation of Dimensionality with Multi-dictionaries

In the previous sections, we illustrated that our methods improved the performance by at least 4% and up to 10.71% compared with the CSP method[149]. Although the nonnegative bases provided a slightly lower accuracy, their nonnegative components can easily interpret ERD/ERD. In this section, we shall introduce a technique to further improve the accuracy.

We note that a transformation of data with a dictionary aims to decorrelate the raw data and express them in a sparse domain. Different dictionaries (transforms) allow to obtain different sparse representations with various sparsity profiles. The short-time Fourier transform (STFT) is often used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time. Gabor filters represent data with different frequencies and orientations. For neuronal oscillations, the continuous Morlet wavelet transform are usually used to optimally identify stimulus-induced amplitude modulations of oscillatory activities. By exploiting the differences of categories in different domains, we can improve the classification accuracy.

For this dataset, we selected the continuous Morlet wavelet transforms with two different bandwidth parameters $f_b = 1$ Hz and $f_b = 6$ Hz, but the same center frequency $f_c = 1$ Hz. This gave us two dictionaries CMOR1-1 and CMOR6-1. Each dictionary formed a 4-D tensor including modes *channels* $\times$ *frequency bins* $\times$ *time frames* $\times$ *trials* for both training and test data.

As a result, a trial became a 4-D tensor with 4 modes, and the training tensor had a size of 62 *channels* $\times$ 23 *frequency bins* $\times$ 50 *time frames* $\times$ 2 *dictionaries* $\times$ 120 *trials*. Training a 5-D tensor needs 4 factors. We used the HODA algorithm to estimate the discriminant bases. Four significant features were selected to classify the data, and returned an average accuracy of 95.71%, (94.29% for subject 1 and 97.14% for subject 2).

To summarize, tensor decompositions with nonnegative, orthonormal, or discriminant bases im-

Table 7.3: Analysis of the classification accuracy of discriminant factors with different training parameters for the BCI Motor Imagery Dataset. (a) the accuracy could be improved with significant features, but decreased when using excessive number of features. (b) decomposition of the training data with small core tensor could in general reduce the accuracy. (c) Comparison of performance of methods.

(a) ACC for different number of features.

No.	Accuracy (%)	
Features	Training	Test
1	95.83	**93.57**
2	95.83	91.79
10	98.33	89.64
30	99.17	88.57
50	**100**	84.64

(b) ACC for different sizes of core tensor.

θ (%)	Core's size	Accuracy (%)
80	$1\times2\times1$	68.93
85	$2\times2\times2$	82.86
90	$4\times4\times4$	88.21
95	$6\times7\times6$	90.71
97	$10\times8\times8$	91.07
99	$25\times11\times12$	**93.57**

(c) Comparison of performance of methods.

Method	Accuracy (%)		
	Subject 1	Subject 2	Average
CSP	82.86	90.00	86.43
NTD-3	85.00	90.00	87.50
HOOI	89.29	91.43	90.36
HODA (4-D)	90.71	96.43	93.57
HODA (5-D)	**94.29**	**97.14**	**95.71**

proved the classification accuracy for the BCI dataset by almost 10%. A comparison of all the methods is given in Table 7.3. Augmentation of dimensionality for samples with additional modes improved the performance. An efficient augmentation approach exploits multiple dictionaries that explain data with different sparsity profiles.

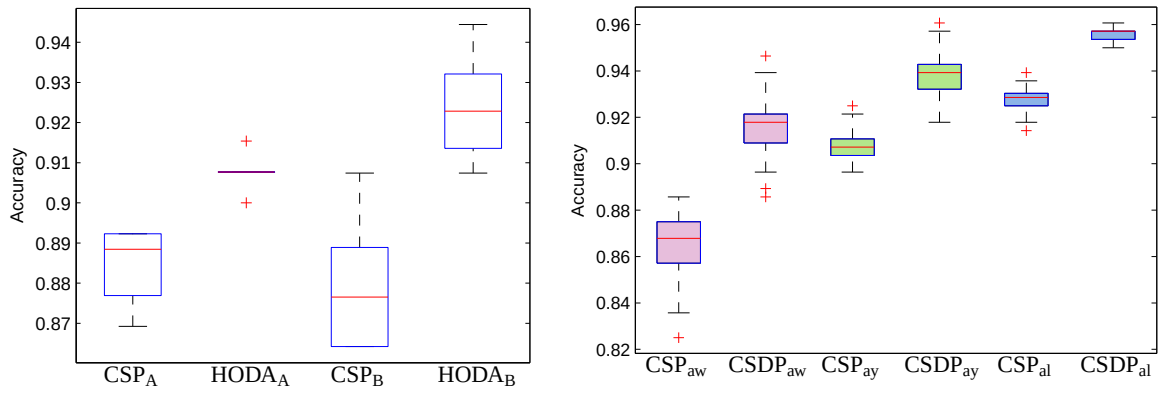
7.10.2 Crossvalidation for ABSP BCI Dataset

This example considers the BCI EEG dataset[48]. We compares our models with CSP methods for two subjects: A and B.

Dataset *"subA_6chan_2LR_s1"*[48] consists of 130 trials of BCI EEG motor imagery for subject A. All EEG signals were recorded in a duration of 3 seconds at a sampling frequency of 256 Hz over 6 channels by a gTec amplifier. Each trial was assigned a label according to left or right hand motor imagery. EEG signals first go through a bandpass filter (8-30 Hz). The 10-fold cross-validation is employed for 10 runs to evaluate classification accuracy. For CSP, we estimated two projected (spatial) filters for each class to extract spatial features and trained an LDA classifier to classify the test data. That means there are four spatial filters for left- and righ-hand motor imageries to give 4 features for EEG signals in a trial. The classification accuracy achieved by CSP is $88.46 \pm 0.96\%$.

For the same signals, we extracted multiway features from spectral tensors which represent EEG signals in each trial in the time-frequency domain using the complex Morlet wavelets CMOR 6-1, and have size of 23 frequency bins (8-30 Hz) $\times$ 77 time frames (in 3 seconds) $\times$ 6 channels. The whole data is a 4-D tensor of size $23 \times 77 \times 6 \times 130$. For the same dataset and the same indices for 10-fold crossvalidation, and LDA classifier, the result obtained in this example is $90.77 \pm 0.36\%$ accuracy, and improved 2%. The p-value associated with the significance testing is 1.25e-6.

Dataset *"subB_6chan_2LR"*[48] consists of 162 trials of BCI EEG motor imagery for subject B. All EEG signals were recorded in a duration of 4 seconds at a sampling frequency of 250 Hz over



(a) Dataset ABSP[48] (b) BCI III motor imagery dataset[66].

Figure 7.16: Comparison of performances for BCI datasets in Examples 7.10.2 and 7.10.3 between CSP and HODA methods.

6 channels by a Neuroscan amplifier. By running 10-fold crossvalidation and using the CSP method to extract 4 spatial features for EEG signals in a trial, we obtained an accuracy of 87.90 $\pm$ 1.4 %. We employed tensor discriminant analysis on 3-D CMOR1-1 spectral tensors of 23 frequency bins (8-30 Hz) $\times$ 100 time frames (in 4 seconds) $\times$ 6 channels. The whole data is a 4-D tensor of size 23 $\times$ 100 $\times$ 6 $\times$ 162. For the same 10-fold crossvalidation indices, and the LDA classifier, our model with one leading Fisher feature achieved 92.41 $\pm$ 1.27% accuracy, and improved 5%. The p-value associated with the significance testing is 7.84e-07. Distribution of accuracies for two subjects shown in Figure 7.16(a) reveals that classification using features extracted from spectral tensors is much more stable and high performance than those by CSP.

7.10.3 Crossvalidation for BCI III Motor Imagery Dataset (Dataset IVb)

This section demonstrates classification of two classes for the BCI III motor imagery competition benchmark[66]. We selected only 7 channels (51-57) from 118 channels of the full data to illustrate the classification performance. EEG signals in trials were extracted from the continuous EEG signals, and organzied into 3-D arrays of modes: 7 channels $\times$ 350 samples (in 3.5 seconds) $\times$ 280 trials[211]. For multiway-features, EEG signals are first transformed into time-frequency domain using the complex MORLET wavelets to form spectral tensors of size 23 frequency bins $\times$ 350 time frames $\times$ 7 channels $\times$ 280 trials. Classification accuracy was evaluated by 5-fold crossvalidation. That means there were 224 3-D tensors for training and 56 3-D tensors for test. The tensor has temporal modes with large number of time frames. A simple technique to deal with this problem is that the data is tensorized to have additional modes. For example, folding the temporal mode of 350 time frames as 5 $\times$ 2 $\times$ 5 $\times$ 7 dimensional tensor can yield a 6-D data tensor.

Using CSP features, we obtained accuracies of 86.63 $\pm$ 1.31%, 90.87 $\pm$ 0.5% and 92.7 $\pm$ 0.49% for subjects *aw, ay al*, respectively. Whereas combination of CSP features and discriminant features from spectral tensors (Common Spatial and Discriminant Projections) helped to achieve accuracies of 91.61 $\pm$ 1.15%, 93.81$\pm$ 0.9%, 95.6 $\pm$ 0.21%. The performances were improved at least 3% accuracy. Classification accuracies for methods are shown in Figure 7.16(b).

7.11 Summary

In this chapter, we have proposed a general approach for model reduction, feature extraction and classification problems of high dimensional dataset. Revisiting the TUCKER models, we have developed robust algorithms within a general framework, which generalizes or extends some existing approaches. A family of flexible algorithms has been developed to find bases with different constraints such as orthogonality, nonnegativity, and discriminant projection. All of them have been verified by extensive numerical experiments for real-world datasets. Through examples, factors with orthogonal components often achieved highest performance (recognition rate) with an acceptable number of significant features. Especially, such bases can be relatively quickly estimated using the HOOI algorithm[58;60] without any category information.

Employing category information to find discriminant bases (Algorithm 7.3) has been shown not only to (slightly) improve the performance, but also to reduce the number of desired selected features. However, complexity of a such algorithm increases and the fitness of approximations decreases.

For all the used datasets, although nonnegative bases did not usually provide the best performance, their components could often help us to physically interpret the data, for example, the BCI EEG datasets. A supervised training paradigm with discriminant criterion incorporated in the cost function has also been presented to find nonnegative factors. However, choosing optimal regularization parameters is still an open problem. Furthermore, multiplicative learning rules for the estimation of nonnegative bases are characterized by rather slow convergence, and frequent convergence to spurious local minima. Therefore, in practice, the HOOI algorithm should be run first to give orthogonal bases which can be then used as initialization for nonnegative bases.

Features can be extracted based on CP decompositions. In this case, samples are also organized in the same way as Tucker decomposition, and the obtained features are rows of the last factors. Recently, we received promising results in seeking discriminant features from EEG signals for healthy children and children with attention deficit using Event-Related Potential[52;53;54].

Finally, our methods and algorithms have shown to be effective for many practical problems. The presented techniques are very perspective and useful in applications like model reduction, pattern recognition, vision, classification, and multi-way clustering.

Chapter 8

Conclusions

The main objective of this thesis is to propose robust algorithms for CP and Tucker decompositions. Rank-one update algorithms (HALS) have lower computational cost than the ALS algorithms, but are compatible with this algorithm in the term of performance. The proposed algorithms can combine multiple regularization terms such as smoothness, orthogonality, nonnegativity and discriminant information.

We also investigated the NQP problem and proposed a recursive approach to solve this problem. Based on the proposed technique, a family of QALS algorithms for nonnegative CP and Tucker decompositions is derived and confirmed as appropriate ALS algorithms for tensor decompositions. We proposed a robust algorithm rK-QALS which can update arbitrary number of components $1 \leqslant K \leqslant R$ instead of one component or all the components. The algorithm allows flexible control of tradeoff between computational cost and performance of the QALS algorithm. That is the rK-QALS algorithm should converge faster than the HALS algorithm, but has low computational cost than that of the QALS algorithm.

All-at-once algorithms based on the damped Gauss-Newton iteration are proposed with low complexity to build up the approximate Hessian and gradient, and also to inverse the approximate Hessian. Especially for CP and NTF, the fast dGN (LM) algorithm has been derived to not only bypass computation of Jacobian, Hessian and gradients but also inverse of the approximate Hessian. The LM algorithms have been experimentally confirmed as the best algorithm for all experiments including difficult benchmarks and real-world applications.

For large-scale tensor factorization, we propose the grid (block) model in which subtensors are first factorized, then the factors for the whole data tensor are approximated from subfactors. Algorithms for large-scale CP and NTF have been derived and confirmed by synthetic and real-wolrd data and applications including EEG analysis and estimation of impulse responses in MIMO systems. The model can be extended to Tucker decomposition.

Finally, we present the model for feature extraction for multiway data based on Tucker and CP decomposition. Applications for BCI, object classification have been verified and confirmed high performance of our model. Algorithms and paradigms in the thesis are demonstrated in the Matlab NFEA toolbox[150].

Bibliography

[1] Karim M. Abadir and Jan R. Magnus. *Matrix Algebra (Econometric Exercises)*. Cambridge University Press, August 2005. (Cited on page 42.)

[2] H. Abdi. Discriminant correspondence analysis. *N.J. Salkind (Ed.): Encyclopedia of Measurement and Statistics*, pages 270–275, 2007. (Cited on page 144.)

[3] E. Acar, C.A. Bingol, H. Bingol, R. Bro, and B. Yener. Computational analysis of epileptic focus localization. In *Proc. of The Fourth IASTED International Conference on Biomedical Engineering BioMED2006*, pages 317–322, Innsbruck, Austria, 15–17 February 2006. (Cited on page 88.)

[4] E. Acar, D. M. Dunlavy, and T. G. Kolda. A scalable optimization approach for fitting canonical tensor decompositions. *Journal of Chemometrics*, 25(2):67–86, February 2011. (Cited on pages 7, 51, 97, 98 and 99.)

[5] E. Acar and B. Yener. Unsupervised multiway data analysis: A literature survey. *IEEE Transactions on Knowledge and Data Engineering*, 21:6–20, 2008. (Cited on pages 5, 6 and 15.)

[6] R. Acar and C. Vogel. Analysis of bounded variation penalty methods for ill-posed problems. *IEEE Transactions on Image Processing*, 10:1217–1229, 1994. (Cited on page 23.)

[7] R. Albright, J. Cox, D. Duling, A. N. Langville, and C. D. Meyer. Algorithms, initializations, and convergence for the nonnegative matrix factorization. Technical report, NCSU Technical Report Math 81706, 2006. (Cited on pages 7, 17, 28 and 39.)

[8] S. Amari. *Differential-Geometrical Methods in Statistics*. Springer Verlag, 1985. (Cited on page 24.)

[9] C.A. Andersson and R. Bro. Improving the speed of multi-way algorithms: Part I. Tucker3. *Chemometrics Intell. Lab. Systems*, 42:93–103, 1998. (Cited on page 87.)

[10] C.A. Andersson and R. Bro. The N-way toolbox for MATLAB. *Chemometrics Intell. Lab. Systems*, 52(1):1–4, 2000. (Cited on pages 5, 17, 18, 19 and 28.)

[11] C.J. Appellof and E.R. Davidson. Strategies for analyzing data from video fluoromatric monitoring of liquid chromatographic effluents. *Analytical Chemistry*, 53:2053–2056, 1981. (Cited on page 5.)

[12] L. Badea. Extracting gene expression profiles common to Colon and Pancreatic Adenocarcinoma using simultaneous nonnegative matrix factorization. In *Proceedings of Pacific Symposium on Biocomputing PSB-2008*, pages 267–278, World Scientific, 2008. (Cited on page 137.)

[13] B.W. Bader, R. Harshman, and T.G. Kolda. Pattern analysis of directed graphs using DEDICOM: An application to Enron Email. Technical Report SAND2006-7744, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, December 2006. (Cited on pages 6 and 16.)

[14] B.W. Bader, R. Harshman, and T.G. Kolda. Temporal analysis of social networks using three-way DEDICOM. Technical Report SAND2006-2161, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, April 2006. (Cited on pages 6, 16 and 137.)

[15] B.W. Bader, R.A. Harshman, and T.G. Kolda. Temporal analysis of semantic graphs using ASALSAN. In *ICDM 2007: Proceedings of the 7th IEEE International Conference on Data Mining*, pages 33–42, October 2007. (Cited on pages 6, 15 and 16.)

[16] B.W. Bader and T.G. Kolda. Efficient MATLAB computations with sparse and factored tensors. *SIAM Journal on Scientific Computing*, 30, 2007. (Cited on page 99.)

[17] B.W. Bader and T.G. Kolda. MATLAB tensor toolbox version 2.4. http://csmr.ca.sandia.gov/∼tgkolda/TensorToolbox/, January 2010. (Cited on pages 51 and 99.)

[18] M.S. Bartlett, J.R. Movellan, and T.J. Sejnowski. Face recognition by independent component analysis. *IEEE Trans. Neural Networks*, 13(6):1450–1464, 2002. (Cited on page 124.)

[19] M. Berry, M. Browne, A. Langville, P. Pauca, and R. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics and Data Analysis*, 52(1):155–173, 2007. (Cited on pages 6, 17 and 18.)

[20] I. Biederman. Recognition-by-components: A theory of human image interpretation. *Psychological Review*, 94:115–148, 1987. (Cited on page 159.)

[21] I. Biederman. Aspects and extension of a theory of human image understanding. *Computational Process in Human Vision: An Interdisciplinary Perspective*, 1988. (Cited on page 159.)

[22] D. A. Bini and P. Boito. A fast algorithm for approximate polynomial GCD based on structured matrix computations. *Operator Theory: Advances and Applications*, 199, 2010. (Cited on page 52.)

[23] C. Boutsidis and E. Gallopoulos. SVD based initialization: A head start for nonnegative matrix factorization. *Pattern Recognition*, 41:1350–1362, 2008. (Cited on page 17.)

[24] R. Bro. PARAFAC. Tutorial and applications. In *Special Issue 2nd Internet Conf. in Chemometrics (INCINC'96)*, volume 38, pages 149–171. Chemom. Intell. Lab. Syst, 1997. (Cited on pages 6, 17 and 18.)

[25] R. Bro. *Multi-way Analysis in the Food Industry - Models, Algorithms, and Applications*. PhD thesis, University of Amsterdam, Holland, 1998. (Cited on pages 5, 6, 18, 19, 23, 51, 87 and 99.)

[26] R. Bro and C.A. Andersson. Improving the speed of multiway algorithms - Part II: Compression. *Chemometrics and Intelligent Laboratory Systems*, 42:105–113, 1998. (Cited on page 18.)

[27] C. Brunner, R. Leeb, G. R. Muller-Putz, and A. Schlogl. BCI competition 2008 - Graz data set A. 2009. (Cited on page 93.)

[28] D. Cai, X. He, and J. Han. Efficient kernel discriminant analysis via spectral regression. In *Proc. Int. Conf. on Data Mining (ICDM'07)*, 2007. (Cited on page 143.)

[29] D. Cai, X. He, and J. Han. SRDA: An efficient algorithm for large-scale discriminant analysis. *IEEE Trans. on Knowl. and Data Eng.*, 20(1):1–12, 2008. (Cited on page 143.)

[30] C. F. Caiafa and A. Cichocki. Generalizing the column-row matrix decomposition to multi-way arrays. *Linear Algebra and its Applications*, 433:557–573, 2010. (Cited on page 36.)

[31] S. Canu, Y. Grandvalet, V. Guigue, and A. Rakotomamonjy. SVM and kernel methods –Matlab toolbox, 2005. (Cited on pages 153, 160 and 164.)

[32] J.-F. Cardoso and A. Souloumiac. Jacobi angles for simultaneous diagonalization. *SIAM J. Matrix Anal. Appl.*, 17(1):161–164, January 1996. (Cited on page 137.)

[33] J.D. Carroll and J.J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of Eckart–Young decomposition. *Psychometrika*, 35(3):283–319, 1970. (Cited on pages 6, 12, 17 and 76.)

[34] R.B. Cattell. Parallel proportional profiles and other principles for determining the choice of factors by rotation. *Psychometrika*, 9:267–283, 1944. (Cited on page 6.)

[35] T. Chan, H.M. Zhou, and R.H. Chan. A continuation method for total variation denoising problems, 1995. (Cited on page 23.)

[36] A. Cichocki, S. Amari, R. Zdunek, R. Kompass, G. Hori, and Z. He. Extended SMART algorithms for non-negative matrix factorization. *Springer, LNAI-4029*, 4029:548–562, 2006. (Cited on pages 24, 25 and 26.)

[37] A. Cichocki and A.-H. Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE Transactions*, 92-A(3):708–721, 2009. (Cited on pages 19, 20, 24, 26, 27, 39, 50, 82, 94, 97 and 131.)

[38] A. Cichocki, A.-H. Phan, and C. Caiafa. Flexible HALS algorithms for sparse non-negative matrix/tensor factorization. In *Proc. of 18-th IEEE workshops on Machine Learning for Signal Processing*, Cancun, Mexico, 16–19, October 2008. (Cited on pages 19 and 24.)

[39] A. Cichocki, A.-H. Phan, R. Zdunek, and L.-Q. Zhang. Flexible component analysis for sparse, smooth, nonnegative coding or representation. In *Lecture Notes in Computer Science, LNCS-4984*, volume 4984, pages 811–820. Springer, 2008. (Cited on page 19.)

[40] A. Cichocki, Y. Washizawa, T. Rutkowski, H. Bakardjian, A.-H. Phan, S. Choi, H. Lee, Q. Zhao, L. Zhang, and Y. Li. Noninvasive BCIs: Multiway signal-processing array decompositions. *Computer*, 41(10):34–42, 2008. (Cited on page 161.)

[41] A. Cichocki and R. Zdunek. Multilayer nonnegative matrix factorization. *Electronics Letters*, 42(16):947–948, 2006. (Cited on pages 7, 18 and 39.)

[42] A. Cichocki and R. Zdunek. Regularized alternating least squares algorithms for non-negative matrix/tensor factorizations. *Springer, LNCS-4493*, 4493:793–802, June 3–7 2007. (Cited on pages 18, 24 and 26.)

[43] A. Cichocki, R. Zdunek, and S. Amari. Csiszar's divergences for non-negative matrix factorization: Family of new algorithms. *Springer, LNCS-3889*, 3889:32–39, 2006. (Cited on page 6.)

[44] A. Cichocki, R. Zdunek, and S. Amari. Hierarchical ALS algorithms for nonnegative matrix and 3D tensor factorization. In *Lecture Notes in Computer Science, LNCS-4666*, pages 169–176, 2007. (Cited on page 19.)

[45] A. Cichocki, R. Zdunek, S. Choi, R. Plemmons, and S. Amari. Nonnegative tensor factorization using Alpha and Beta divergencies. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP07)*, volume III, pages 1393–1396, Honolulu, Hawaii, USA, April 15–20 2007. (Cited on pages 24, 26 and 28.)

[46] A. Cichocki, R. Zdunek, S. Choi, R. Plemmons, and S.-I. Amari. Novel multi-layer nonnegative tensor factorization with sparsity constraints. *Springer, LNCS-4432*, 4432:271–280, April 11–14 2007. (Cited on pages 24 and 26.)

[47] A. Cichocki, R. Zdunek, A.-H. Phan, and S. Amari. *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. Wiley, Chichester, 2009. (Cited on pages 5, 6, 7, 8, 15, 17, 18, 19, 21, 24, 28, 39, 40, 63, 68, 82, 93, 94, 98, 135, 149, 151 and 162.)

[48] A. Cichocki and Q. Zhao. EEG motor imagery dataset. Technical report, Laboratory for Advanced Brain Signal Processing, BSI, RIKEN, Saitama, Japan, 2011. (Cited on pages 168 and 169.)

[49] P. Comon. Tensor diagonalization, a useful tool in signal processing. In *10th International Federation of Automatic*

Control Symposium on System Identification, pages 77–82, 1994. (Cited on page 5.)

[50] P. Comon. Tensor package, enhanced line search. http://www.i3s.unice.fr/∼pcomon/TensorPackage.html, May, 2010. (Cited on pages 6 and 99.)

[51] P. Comon, X. Luciani, and A. L. F. de Almeida. Tensor decompositions, alternating least squares and other tales. *Jour. Chemometrics*, 23, 2009. (Cited on pages 6, 18, 19, 51 and 58.)

[52] F. Cong, I. Kalyakin, A.-H. Phan, A. Cichocki, T. Huttunen-Scott, H. Lyytinen, and T. Ristaniemi. Extract mismatch negativity and p3a through two-dimensional nonnegative decomposition on time-frequency represented event-related potentials. In *ISNN, 2010*, pages 385–391, 2010. (Cited on page 170.)

[53] F. Cong, A.-H. Phan, A. Cichocki, H. Lyytinen, and T. Ristaniemi. Identical fits of nonnegative matrix/tensor factorization may correspond to different extracted event-related potentials. In *International Joint Conference on Neural Networks 2010*, pages 2260–2264, 2010. (Cited on page 170.)

[54] F. Cong, A.-H. Phan, H. Lyytinen, T. Ristaniemi, and A. Cichocki. Classifying healthy children and children with attention deficit through features derived from sparse and nonnegative tensor factorization using event-related potential. In *LVA/ICA 2010*, pages 620–628, 2010. (Cited on page 170.)

[55] M.E. Daube-Witherspoon and G. Muehllehner. An iterative image space reconstruction algorthm suitable for volume ECT. *IEEE Transactions on Medical Imaging*, 5:61–66, 1986. (Cited on pages 30 and 84.)

[56] L. De Lathauwer. A link between the canonical decomposition in multilinear algebra and simultaneous matrix diagonalization. *SIAM J. Matrix Anal. Appl.*, 28:642–666, 2006. (Cited on page 52.)

[57] L. De Lathauwer. Decompositions of a higher-order tensor in block terms – Part I: Lemmas for partitioned matrices. *SIAM J. Matrix Anal. Appl.*, 30(3):1022–1032, 2008. Special Issue on Tensor Decompositions and Applications. (Cited on page 28.)

[58] L. De Lathauwer, B. de Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM Journal of Matrix Analysis and Applications*, 21:1253–1278, 2001. (Cited on pages 15, 28, 97, 98, 120, 137, 141 and 170.)

[59] L. De Lathauwer, B. De Moor, and J. Vandewalle. A Multilinear Singular Value Decomposition. *SIAM J. Matrix Anal. Appl.*, 21(4):1253–1278, 2000. (Cited on page 132.)

[60] L. De Lathauwer, B. De Moor, and J. Vandewalle. On the best rank-1 and rank-(R1,R2,...,RN) approximation of higher-order tensors. *SIAM J. Matrix Anal. Appl.*, 21(4):1324–1342, 2000. (Cited on pages 15, 28, 63, 98, 132, 141 and 170.)

[61] L. De Lathauwer and D. Nion. Decompositions of a higher-order tensor in block terms – Part III: Alternating least squares algorithms. *SIAM J. Matrix Anal. Appl.*, 30(3):1067–1083, 2008. Special Issue Tensor Decompositions and Applications. (Cited on page 28.)

[62] L. De Lathauwer and J. Vandewalle. Dimensionality reduction in higher-order signal processing and rank-$(R_1, R_2, \ldots, R_n)$ reduction in multilinear algebra. *Linear Algebra Applications*, 391:31–55, November 2004. (Cited on page 141.)

[63] A.R. De Pierro. On the relation between the ISRA and the EM algorithm for positron emission tomography. *IEEE Transactions on Medial Imaging*, 12(2):328–333, June 1993. (Cited on pages 30 and 84.)

[64] I. Dhillon and S. Sra. Generalized nonnegative matrix approximations with Bregman divergences. In *Neural Information Proc. Systems*, pages 283–290, Vancouver, Canada, December 2005. (Cited on page 28.)

[65] C. Ding, T. Li, W. Peng, and H. Park. Orthogonal nonnegative matrix tri-factorizations for clustering. In *KDD06*, pages 126–135, New York, NY, USA, 2006. ACM Press. (Cited on page 66.)

[66] G. Dornhege, B. Blankertz, G. Curio, and K.-R. Müller. Boosting bit rates in non-invasive EEG single-trial classifications by feature combination and multi-class paradigms. *IEEE Trans. Biomed. Eng.*, 51, 2004. (Cited on page 169.)

[67] P.P.B. Eggermont and V.N. LaRiccia. Maximum smoothed likelihood density estimation for inverse problems. *Ann. Statist.*, 23(1):199–220, 1995. (Cited on pages 30 and 84.)

[68] Center for Brain-Like Computing and Shanghai Jiao Tong University Machine Intelligence. Data set for single trial EEG classification in BCI. (Cited on page 161.)

[69] G. E. Forsythe and C. B. Moler. *Computer Solution of Linear Algebraic Systems, Chapter 19*. Prentice-Hall, 1967. (Cited on pages 108 and 116.)

[70] A. Franc. *Etude algebrique des multitableaux: apports de l'algebrique tensorielle*. PhD thesis, Université Montpellier II, 1992. (Cited on page 19.)

[71] J. H. Friedman. Exploratory projection pursuit. *Journal of the American Statistical Association*, 82(397):249–266, 1987. (Cited on page 135.)

[72] J. H. Friedman. Regularized discriminant analysis. *Journal of the American Statistical Association*, 84(405):165–175, 1989. (Cited on page 135.)

[73] N. Gillis and F. Glineur. Nonnegative factorization and the maximum edge biclique problem. CORE Discussion Papers 2008064, 2008. (Cited on page 39.)

[74] G.H. Golub and C.F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, London, 1996. (Cited on page 148.)

[75] S.A. Goreinov, E.E. Tyrtyshnikov, and N.L. Zamarashkin. A theory of pseudo-skeleton approximations. *Linear Alegebra and Applications*, 261:1–21, 1997. (Cited on page 36.)

[76] P. Guillaume and R. Pintelon. A Gauss-Newton-like optimization algorithm for weighted nonlinear nonlinear least squares problems. *IEEE Trans. Signal Processing*, 44, 1996. (Cited on page 66.)

[77] X. Guo, S. Miron, D. Brie, and A. Stegeman. Uni-mode and partial uniqueness conditions for CANDECOMP/PARAFAC of three-way arrays with linearly dependent loadings. *SIAM J. Matrix Anal. Appl.*, page in press, 2011. (Cited on page 51.)

[78] T.M. Hancewicz and J.-H. Wang. Discriminant image resolution: a novel multivariate image analysis method utilizing a spatial classification constraint in addition to bilinear nonnegativity. *Chemometrics and Intelligent Laboratory Systems*, 77:18–31, 2005. (Cited on pages 7, 18 and 39.)

[79] R. A. Harshman. Models for analysis of asymmetrical relationships among n objects or stimuli. In *Paper presented at the First Joint Meeting of the Psychometric Society and The Society for Mathematical Psychology*. Hamilton, 1978. (Cited on page 6.)

[80] R. A. Harshman and M. E Lundy. Three-way DEDICOM: Analyzing multiple matrices of asymmetric relationships. In *the Annual Meeting of the North American Psychometric Society*, Columbus, Ohio, 1992. (Cited on pages 6, 15 and 16.)

[81] R.A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an explanatory multimodal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970. (Cited on pages 6, 12, 15, 17, 18 and 19.)

[82] R.A. Harshman. PARAFAC2: Mathematical and technical notes. *UCLA Working Papers in Phonetics*, 22:30–44, 1972. (Cited on pages 6 and 137.)

[83] R.A. Harshman and M.E. Lundy. *Research Methods for Multimode Data Analysis*. Praeger, New York, USA, 1984. (Cited on page 6.)

[84] T. Hazan, S. Polak, and A. Shashua. Sparse image coding using a 3D non-negative tensor factorization. In *Proc. Int. Conference on Computer Vision (ICCV)*, pages 50–57, 2005. (Cited on page 6.)

[85] Z. He and A. Cichocki. Efficient method for estimating the dimension of Tucker3 model. *Journal of Multivariate Analysis*, 2009. (Cited on page 142.)

[86] M. Heiler and C. Schnoerr. Controlling sparseness in non-negative tensor factorization. *Springer LNCS*, 3951:56–67, 2006. (Cited on page 6.)

[87] D. J. Higham and N. J. Higham. *MATLAB Guide, Second Edition*. SIAM, 2005. (Cited on pages 108 and 116.)

[88] N. J. Higham. Accuracy and stability of numerical algorithms. *SIAM*, 1996. (Cited on pages 108 and 116.)

[89] F.L. Hitchcock. Multiple invariants and generalized rank of a p-way matrix or tensor. *Journal of Mathematics and Physics*, 7:39–79, 1927. (Cited on page 12.)

[90] N.-D. Ho. *Nonnegative Matrix Factorization - Algorithms and Applications*. These/dissertation, Universite Catholique de Louvain, Belgium, FSA/INMA - Departement d'ingenierie mathematique, 2008. (Cited on pages 19 and 23.)

[91] N.-D. Ho, P. Van Dooren, and V.D. Blondel. Descent methods for nonnegative matrix factorization. *Numerical Linear Algebra in Signals, Systems and Control*, 2008. (Cited on page 19.)

[92] K. Horio and T. Yamakawa. Feedback self-organizing map and its application to spatio-temporal pattern classification. *International Journal of Computational Intelligence and Applications*, 1(1):1–18, 2001. (Cited on page 135.)

[93] K. Horio and T. Yamakawa. Handwritten character recognition based on relative position of local features extracted by self-organizing maps. *International Journal of Innovative Computing and Control*, 3(4):789 – 798, 2007-08. (Cited on pages 135 and 155.)

[94] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 1990. (Cited on pages 42, 52, 57 and 65.)

[95] R. A. Horn and C. R. Johnson. *Topics in matrix analysis*. Cambridge University Press, Cambridge, 1991. (Cited on page 72.)

[96] A. Hyvarinen, J. Karhunen, and E. Oja. *Independent Component Analysis*. John Wiley & Sons Ltd, New York, 2001. (Cited on page 135.)

[97] A. K. Jain, R. P.W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:4–37, 2000. (Cited on pages 135 and 151.)

[98] J. Jiang, L. Zhang, and T. Furukawa. A class density approximation neural network for improving the generalization of fisherface. *Neurocomputing*, 71:3230–3246, 2008. (Cited on page 135.)

[99] J. Jiang, L. Zhang, and T. Furukawa. RBFxSOM: An efficient algorithm for large-scale multi-system learning. *IEICE Transactions on Information and Systems*, E92-D(7):1388–1396, 2009. (Cited on pages 135 and 155.)

[100] H.A.L. Kiers. An alternating least squares algorithm for PARAFAC2 and DEDICOM3. *Computational Statistics and*

Data Analysis, 16:103–118, 1993. (Cited on page 15.)

[101] H.A.L. Kiers. A three-step algorithm for CANDECOMP/PARAFAC analysis of large data sets with multicollinearity. *Journal of Chemometrics*, 12(3):155–171, 1998. (Cited on page 18.)

[102] H. Kim and H. Park. Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. *SIAM J. Matrix Anal. Appl.*, 30:713–730, July 2008. (Cited on page 39.)

[103] Y.-D. Kim and S. Choi. Nonnegative Tucker decomposition. In *Proc. of Conf. Computer Vision and Pattern Recognition (CVPR-2007)*, Minneapolis, Minnesota, June 2007. (Cited on pages 15, 28, 40, 68, 149 and 151.)

[104] J. Kivinen and M.K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132, 1997. (Cited on page 40.)

[105] T.G. Kolda and B. Bader. The TOPHITS model for higher-order web link analysis. In *Proceedings of the SIAM Data Mining Conference Workshop on Link Analysis, Counterterrorism and Security*, 2006. (Cited on pages 6, 15 and 16.)

[106] T.G. Kolda and B.W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, September 2009. (Cited on pages 5, 11, 13, 14, 15, 17, 18, 28, 63, 76, 96, 132 and 135.)

[107] T.G. Kolda, B.W. Bader, and J.P. Kenny. Higher-order web link analysis using multilinear algebra. In *ICDM 2005: Proceedings of the 5th IEEE International Conference on Data Mining*, pages 242–249, November 2005. (Cited on page 16.)

[108] Z. Koldovský, P. Tichavský, and A.-H. Phan. Stability analysis and fast damped Gauss-Newton algorithm for INDSCAL tensor decomposition. In *Statistical Signal Processing Workshop (SSP), IEEE*, pages 581–584, 2011. (Cited on pages 97 and 100.)

[109] A. N. Langville, C. D. Meyer, and R. Albright. Initializations for the nonnegative matrix factorization. In *Proc. of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Philadelphia, USA, August 20–23 2006. (Cited on pages 7, 28 and 39.)

[110] H Lantéri, R. Soummmer, and C. Aime. Comparison between ISRA and RLA algorithms: Use of a Wiener filter based stopping criterion. *Astronomy and Astrophysics Supplemantary Series*, 140:235–246., 1999. (Cited on pages 30 and 84.)

[111] L. De Lathauwer and J. Castaing. Blind identification of underdetermined mixtures by simultaneous matrix diagonalization. *IEEE Transactions on Signal Processing*, 56(3):1096–1105, 2008. (Cited on page 52.)

[112] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. (Cited on pages 94 and 155.)

[113] D.D. Lee and H.S. Seung. Learning of the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999. (Cited on pages 6, 31, 40 and 84.)

[114] Z. Liang and P.F. Shi. Uncorrelated discriminant vectors using a kernel method. *Pattern Recognition*, 38(2):307–310, 2005. (Cited on page 124.)

[115] C. J. Lin. Projected gradient methods for non-negative matrix factorization. *Neural Computation*, 19(10):2756–2779, October 2007. (Cited on pages 18, 39 and 40.)

[116] C.-Y. Lin, N. Cao, S. Liu, S. Papadimitriou, J. Sun, and X. Yan. Smallblue: Social network analysis for expertise search and collective intelligence. In *ICDE*, pages 1483–1486, 2009. (Cited on page 5.)

[117] X. Luciani and L. Albera. Semi-algebraic canonical decomposition of multi-way arrays and joint eigenvalue decomposition. In *ICASSP*, pages 4104–4107, 2011. (Cited on page 52.)

[118] M.E. Lundy, R.E. Harshman, and J.B. Kruskal. A two-stage procedure incorporating good features of both trilinear and quadrilinear models. pages 123–130, 1989. (Cited on pages 6 and 15.)

[119] D. Terzopoulos M. A. O. Vasilescu. Multilinear analysis of image ensembles: Tensorfaces. In *7th European Conference on Computer Vision (ECCV'02), Lecture Notes in Computer Science*, volume 2350, pages 447–460, 2002. (Cited on page 5.)

[120] J. R. Magnus and H. Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics, 2nd Edition*. John Wiley & Sons, March 1999. (Cited on page 54.)

[121] A. Martinez and R. Benavente. The AR face database. Technical Report 24, Computer Vision Center (CVC), Barcelona, Spain, June 1998. (Cited on page 158.)

[122] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010. (Cited on pages 108 and 116.)

[123] K. Matsuoka. Noise injection into inputs in back-propagation learning. *IEEE Trans. Syst, Man, Cybern*, 22, 1992. (Cited on page 124.)

[124] K. Matsuoka and M. Kawamoto. A neural network that self-organizes to perform three operations related to principal component analysis. *Neural Networks*, 7(5):753 – 765, 1994. (Cited on page 135.)

[125] G. J. McLachlan. *Discriminant Analysis and Statistical Pattern Recognition*. Wiley Interscience, 2004. (Cited on page 144.)

[126] M. Minami and S. Eguchi. Robust blind source separation by Beta-divergence. *Neural Computation*, 14:1859–1886, 2002. (Cited on page 26.)

[127] B. C. Mitchell and D. S. Burdick. Slowly converging PARAFAC sequences: Swamps and two-factor degeneracies. *Jour. Chemometrics*, 8:155–168, 1994. (Cited on page 51.)

[128] F. Miwakeichi, E. Martnez-Montes, P. Valds-Sosa, N. Nishiyama, H. Mizuhara, and Y. Yamaguchi. Decomposing EEG data into space−time−frequency components using parallel factor analysis. *NeuroImage*, 22(3):1035–1045, 2004. (Cited on page 6.)

[129] M. Mørup, L. K. Hansen, C. S. Herrmann, J. Parnas, and S. M. Arnfred. Parallel factor analysis as an exploratory tool for wavelet transformed event-related EEG. *NeuroImage*, 29(3):938–947, 2006. (Cited on page 6.)

[130] M. Mørup, L.K. Hansen, and S.M. Arnfred. ERPWAVELAB a toolbox for multi-channel analysis of time-frequency transformed event related potentials. *Journal of Neuroscience Methods*, 161(2):361–368, 2006. (Cited on pages 28, 40 and 122.)

[131] M. Mørup, L.K. Hansen, and S.M. Arnfred. Algorithms for sparse nonnegative Tucker decompositions. *Neural Computation*, 20:2112–2131, 2008. (Cited on pages 5, 15, 28, 68, 98, 120, 149 and 151.)

[132] M. Mørup, L.K. Hansen, J. Parnas, and S.M. Arnfred. Decomposing the time-frequency representation of EEG using non-negative matrix and multi-way factorization. Technical report, 2006. (Cited on pages 40, 82, 93, 97 and 122.)

[133] C. Navasca, L. De Lathauwer, and S. Kindermann. Swamp reducing technique for tensor decomposition. In *Proc. 16th European Signal Processing Conference (EUSIPCO 2008)*, 2008. (Cited on page 18.)

[134] S. A. Nene, S. K. Nayar, and H. Murase. Columbia object image library (COIL-20). Technical Report CUCS-005-96, Columbia University, February 1996. (Cited on page 153.)

[135] F. Nie, S. Xiang, Y. Song, and Ch. Zhang. Extracting the optimal dimensionality for local tensor discriminant analysis. *Pattern Recogn.*, 42(1):105–114, 2009. (Cited on page 143.)

[136] H. B. Nielsen. Damping parameter in Marquardt's method. Technical report, 1999. (Cited on page 62.)

[137] M. Nikolova. Minimizers of cost-functions involving nonsmooth data-fidelity terms. application to the processing of outliers. *SIAM Journal on Numerical Analysis*, 40(3):965–994, 2002. (Cited on page 22.)

[138] NIKON. Scene recognition system. http://imaging.nikon.com/products/imaging/technology/scene/19/index.htm. (Cited on page 159.)

[139] D. Nion and L. De Lathauwer. A tensor-based blind DS-CDMA receiver using simultaneous matrix diagonalization. In *Proc. of SPAWC 07, IEEE Workshop on Signal Processing Advances in Wireless Communications, Helsinki, Finland*, June 2007. (Cited on page 88.)

[140] D. Nion and L. De Lathauwer. An enhanced line search scheme for complex-valued tensor decompositions. Application in DS-CDMA. *Signal Processing*, 88(3):749–755, 2008. (Cited on pages 7, 18 and 39.)

[141] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001. (Cited on page 159.)

[142] P. Paatero. Least-squares formulation of robust nonnegative factor analysis. *Chemometrics and Intelligent Laboratory Systems*, 37:23–35, 1997. (Cited on pages 7, 40 and 51.)

[143] P. Paatero. A weighted non-negative least squares algorithm for three-way PARAFAC factor analysis. *Chemometrics Intelligent Laboratory Systems*, 38(2):223–242, 1997. (Cited on pages 7, 40, 51, 52, 54, 55, 59, 63, 65, 69 and 98.)

[144] P. Paatero. The multilinear engine: A table-driven, least squares program for solving multilinear problems, including the n-way parallel factor analysis model. *Journal of Computational and Graphical Statistics*, 8(4):854–888, December 1999. (Cited on pages 7, 51 and 53.)

[145] P. Paatero and P. K. Hopke. Rotational tools for factor analytic models. *Journal of Chemometrics*, 23:91–100, 2009. (Cited on page 18.)

[146] P. Paatero, C. Navasca, and P. Hopke. Fast rotationally enhanced alternating-least-squares. Workshop on Tensor Decompositions and Applications (TDA 2010), SIAM, 2010. (Cited on page 18.)

[147] G. Pfurtscheller and Lopes F. H. da Silva. Event-related EEG/MEG synchronization and desynchronization: basic principles. *Clin Neurophysiol*, 110:1842–1857, 1997. (Cited on page 162.)

[148] G. Pfurtscheller and Lopes F. H. da Silva. EEG event-related desynchronization (ERD) and event-related synchronization (ERS). In E. Niedermeyer and F. L. da Silva, editors, *Electroencephalography: Basic Principles , Clinical Applications, and Related Fields*, volume 5, 2005. (Cited on page 162.)

[149] G. Pfurtscheller and C. Neuper. Motor imagery and direct brain-computer communication. *IEEE*, pages 1123–1134, 2001. (Cited on pages 161, 162 and 167.)

[150] A.-H. Phan. NFEA: Tensor toolbox for feature extraction and applications. `http://www.bsp.brain.riken.jp/phan/nfea.html`, 2011. (Cited on pages 6, 135, 161 and 171.)

[151] A.-H. Phan and A. Cichocki. Fast and efficient algorithms for nonnegative Tucker decomposition. In *Proc. of The Fifth International Symposium on Neural Networks, Springer LNCS-5264*, pages 772–782, Beijing, China, 24–28,

September 2008. (Cited on page 28.)

[152] A.-H. Phan and A. Cichocki. Multi-way nonnegative tensor factorization using fast hierarchical alternating least squares algorithm (HALS). In *Proc. of The 2008 International Symposium on Nonlinear Theory and its Applications*, Budapest, Hungary, 2008. (Cited on pages 19, 39, 82 and 97.)

[153] A.-H. Phan and A. Cichocki. Analysis of interactions among hidden components for tucker model. In *APSIPA Annual Summit and Conference*, 2009. (Cited on page 6.)

[154] A.-H. Phan and A. Cichocki. Block decomposition for very large-scale nonnegative tensor factorization. In *CAMSAP*, pages 316–319, 2009. (Cited on page 82.)

[155] A.-H. Phan and A. Cichocki. Fast nonnegative tensor factorization for very large-scale problems using two-stage procedure. In *CAMSAP*, pages 297 –300, 2009. (Cited on page 82.)

[156] A.-H. Phan and A. Cichocki. Local learning rules for nonnegative Tucker decomposition. In *Neural Information Processing*, volume 5863 of *Lecture Notes in Computer Science*, pages 538–545. Springer Berlin / Heidelberg, 2009. (Cited on pages 20 and 120.)

[157] A.-H. Phan and A. Cichocki. Tensor decompositions for feature extraction and classification of high dimensional datasets. *Nonlinear Theory and Its Applications, IEICE*, 1:37–68 (invited paper), 2010. (Cited on pages 6, 126, 149, 151 and 161.)

[158] A. H. Phan and A. Cichocki. Extended HALS algorithm for nonnegative Tucker decomposition and its applications for multiway analysis and classification. *Neurocomputing*, 74(11):1956 – 1969, 2011. Selected papers from ICONIP 2009. (Cited on pages 20 and 120.)

[159] A.-H. Phan and A. Cichocki. PARAFAC algorithms for large-scale problems. *Neurocomputing*, 74(11):1970 – 1984, 2011. Selected papers from ICONIP 2009. (Cited on page 77.)

[160] A. H. Phan and A. Cichocki. Seeking an appropriate alternative least squares algorithm for nonnegative tensor factorizations. *Neural Computing and Applications*, pages 1–15, 2011. 10.1007/s00521-011-0652-0. (Cited on page 97.)

[161] A. H. Phan, A. Cichocki, K. Matsuoka, and J. Cao. Novel hierarchical ALS algorithm for nonnegative tensor factorization. In *ICASSP*, pages 1984–1987, 2011. (Cited on pages 20 and 97.)

[162] A.-H. Phan, A. Cichocki, and K.S. Nguyen. Simple and efficient algorithm for distributed compressed sensing. In *Machine Learning for Signal Processing*, pages 61 – 66, Cancun, 2008. (Cited on page 21.)

[163] A.-H. Phan, A. Cichocki, and Th. Vu-Dinh. Classification of scenes based on multiway feature extraction. In *Advanced Technologies for Communications (ATC), 2010 International Conference on*, pages 142–145, 2010. (Cited on page 160.)

[164] A.-H. Phan, A. Cichocki, and Th. Vu-Dinh. Nonnegative DEDICOM based on tensor decompositions for social networks exploration. *Australian Journal of Intelligent Information Processing Systems (ICONIP'10)*, 12(1):10–15, 2010. (Cited on pages 6 and 16.)

[165] A.-H. Phan, A. Cichocki, and Th. Vu-Dinh. A tensorial approach to single trial recognition for brain computer interface. In *Advanced Technologies for Communications (ATC), 2010 International Conference on*, pages 138–141, 2010. (Cited on page 161.)

[166] A. H. Phan, A. Cichocki, R. Zdunek, and T. Vu-Dinh. Novel alternating least squares algorithm for nonnegative matrix and tensor factorizations. In Kok Wai Wong, B. Sumudu U. Mendis, and Abdesselam Bouzerdoum, editors, *ICONIP (1)*, volume 6443 of *Lecture Notes in Computer Science*, pages 262–269. Springer, 2010. (Cited on page 97.)

[167] A.-H. Phan, P. Tichavský, and A. Cichocki. Low complexity damped Gauss-Newton algorithms for parallel factor analysis. *SIAM, SIMAX*, 2010,(submit). (Cited on pages 7, 65, 72 and 97.)

[168] A.-H. Phan, P. Tichavský, and A. Cichocki. Damped Gauss-Newton algorithm for nonnegative Tucker decomposition. In *Statistical Signal Processing Workshop (SSP), IEEE*, pages 665 –668, 2011. (Cited on pages 7, 73 and 97.)

[169] A.-H. Phan, P. Tichavský, and A. Cichocki. Fast damped Gauss-Newton algorithm for sparse and nonnegative tensor factorization. In *ICASSP*, pages 1988 – 1991, 2011. (Cited on pages 72, 73 and 97.)

[170] C. E. Priebe, J.M. Conroy, D. J. Marchette, and Y. Park. Scan statistics on Enron graphs. *Comput. Math. Organ. Theory*, 11(3):229–247, 2005. (Cited on page 16.)

[171] M. Rajih, P. Comon, and R. A. Harshman. Enhanced line search: A novel method to accelerate PARAFAC. *SIAM J. Matrix Anal. Appl.*, 30(3):1128–1147, 2008. (Cited on pages 7, 18, 19, 39, 51, 87 and 99.)

[172] W. S. Rayens and B. C. Mitchell. Two-factor degeneracies and a stabilization of PARAFAC. *Chemometrics Intell. Lab. Syst.*, 38:173–181, 1997. (Cited on page 51.)

[173] F. Roemer and M. Haardt. A closed-form solution for multilinear PARAFAC decompositions. In *Proc. 5-th IEEE Sensor Array and Multich. Sig. Proc. Workshop (SAM 2008)*, pages 487 – 491, July 2008. (Cited on page 52.)

[174] M. Rojas and T. Steihaug. An interior-point trust-region-based method for large-scale non-negative regularization. *Inverse Problems*, 18:1291–1307, 2002. (Cited on page 66.)

[175] J.-P. Royer, P. Comon, and N. T. Moreau. Computing the nonnegative 3-way tensor factorization using tikhonov

regularization. In *ICASSP*, pages 2732–2735, 2011. (Cited on page 18.)

[176] J.-P. Royer, P. Comon, and N. T. Moreau. Nonnegative 3-way tensor factorization via conjugate gradient with globally optimal stepsize. In *ICASSP*, pages 4040–4043, 2011. (Cited on page 51.)

[177] P. Sajda, S. Du, T. Brown, L.C. Parra, and R. Stoyanova. Recovery of constituent spectra in 3D chemical shift imaging using nonnegative matrix factorization. In *Proc. of 4th International Symposium on Independent Component Analysis and Blind Signal Separation*, pages 71–76, Nara, Japan, April 2003. (Cited on page 6.)

[178] F. Samaria and A.C. Harter. Parameterisation of a stochastic model for human face identification. In *Proceedings of the Second IEEE Workshop on Applications of Computer Vision*, 1994. (Cited on page 124.)

[179] S. Sanei, A. H. Phan, J-L. Lo, V. Abolghasemi, and A. Cichocki. A compressive sensing approach for progressive transmission of images. In *Digital Signal Processing, 2009*, pages 1–5, 2009. (Cited on page 21.)

[180] B. Savas and L. Eldén. Handwritten digit classification using higher order singular value decomposition. *Pattern Recogn.*, 40(3):993–1003, 2007. (Cited on page 95.)

[181] F. Sha, Y. Lin, L. K. Saul, and D. D. Lee. Multiplicative updates for nonnegative quadratic programming. *Neural Computation*, 19, 2007. (Cited on page 40.)

[182] F. Sha, L. K. Saul, and D. D. Lee. Multiplicative updates for nonnegative quadratic programming in support vector machines. In *in Advances in Neural Information Processing Systems 15*, pages 1041–1048. MIT Press, 2002. (Cited on page 40.)

[183] A. Shashua and T. Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *Proc. of the 22-th International Conference on Machine Learning*, Bonn, Germany, 2005. (Cited on page 40.)

[184] A. Shashua, R. Zass, and T. Hazan. Multi-way clustering using super-symmetric non-negative tensor factorization. In *European Conference on Computer Vision (ECCV)*, Graz, Austria, May 2006. (Cited on page 6.)

[185] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Pittsburgh, PA, USA, 1994. (Cited on page 51.)

[186] N. Sidiropoulos, G. Giannakis, and R. Bro. Blind PARAFAC receivers for DS-CDMA systems. *IEEE Transactions on Signal Processing*, 48(3):810–823, 2000. (Cited on pages 5, 88, 127 and 128.)

[187] N.D. Sidiropoulos and R. Bro. PARAFAC techniques for signal separation. In P. Stoica, G. Giannakis, Y. Hua, and L. Tong, editors, *Signal Processing Advances in Communications*, volume 2, chapter 4. Prentice-Hall, Upper Saddle River, NJ, USA, 2000. (Cited on pages 6, 17 and 18.)

[188] A. Smilde, R. Bro, and P. Geladi. *Multi-way Analysis: Applications in the Chemical Sciences*. John Wiley & Sons Ltd, New York, 2004. (Cited on pages 6 and 17.)

[189] H. W. Sorenson. *Parameter estimation: principles and problems*. Marcel Dekker, NY, USA, 1980. (Cited on page 66.)

[190] J. Sun. *Incremental Pattern Discovery on Streams, Graphs and Tensors*. PhD thesis, CMU-CS-07-149, 2007. (Cited on page 75.)

[191] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proc.of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 374–383, 2006. (Cited on page 75.)

[192] J. Sun, D. Tao, S. Papadimitriou, P. S. Yu, and C. Faloutsos. Incremental tensor analysis: Theory and applications. *TKDD*, 2(3), 2008. (Cited on page 135.)

[193] Y. Takane and H.A.L. Kiers. Latent class DEDICOM. *Journal of Classification*, 14:225–247, 1997. (Cited on page 15.)

[194] C. Tallon-Baudry, O. Bertrand, C. Delpuech, and J. Pernier. Stimulus specificity of phase-locked and non-phase-locked 40 Hz visual responses in human. *Journal of Neuroscience*, 16 (13):4240–4249, 1996. (Cited on page 122.)

[195] D. Tao, X. Li, X. Wu, and S. J. Maybank. General tensor discriminant analysis and Gabor features for gait recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(10):1700–1715, 2007. (Cited on page 135.)

[196] P. Tichavský and Z. Koldovský. Simultaneous search for all modes in multilinear models. pages 4114 – 4117. Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP10), 2010. (Cited on pages 7, 51, 52 and 61.)

[197] P. Tichavský and Z. Koldovský. Stability of CANDECOMP-PARAFAC tensor decomposition. In *ICASSP*, pages 4164–4167, 2011. (Cited on pages 97 and 100.)

[198] P. Tichavský and Z. Koldovský. Weight adjusted tensor method for blind separation of underdetermined mixtures of nonstationary sources. *IEEE Transactions on Signal Processing*, 59(3):1037–1047, 2011. (Cited on pages 97 and 100.)

[199] G. Tomasi. INDAFAC and PARAFAC3W. `http://www.models.kvl/dk/source/indafac/index.asp`, 2003. (Cited on page 98.)

[200] G. Tomasi. *Practical and Computational Aspects in Chemometric Data Analysis*. PhD thesis, Frederiksberg, Denmark, 2006. (Cited on pages 18, 19, 39, 51, 52, 53, 54, 55, 59, 63, 98 and 99.)

[201] G. Tomasi. Recent developments in fast algorithms for fitting the PARAFAC model. Greece, 2006. TRICAP. (Cited on pages 52 and 98.)

[202] G. Tomasi and R. Bro. PARAFAC and missing values. *Chemometrics Intelligent Laboratory Systems*, 75(2):163–180, 2005. (Cited on pages 7, 51, 54 and 63.)

[203] G. Tomasi and R. Bro. A comparison of algorithms for fitting the PARAFAC model. *Computational Statistics and Data Analysis*, 50(7):1700–1734, April 2006. (Cited on pages 7, 51, 60, 76 and 98.)

[204] G. Tomasi and R. Bro. *Comprehensive Chemometrics*, chapter Multilinear Models: Iterative Methods, pages 411–451. Number 22. Elsevier, Oxford, 2009. (Cited on page 18.)

[205] L.R. Tucker. The extension of factor analysis to three-dimensional matrices. In H. Gulliksen and N. Frederiksen, editors, *Contributions to Mathematical Psychology*, pages 110–127. Holt, Rinehart and Winston, New York, 1964. (Cited on page 136.)

[206] L.R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966. (Cited on pages 6, 14 and 136.)

[207] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, November 2008. (Cited on pages 95 and 155.)

[208] P. M. Kroonenberg W. J. Heiser. Dimensionwise fitting in PARAFAC-CANDECOMP with missing data and constrained parameters. Technical Report PRM 97-01, Leiden Psychological Reports, 1997. (Cited on page 19.)

[209] H. Wang, S. Yan, D. Xu, X. Tang, , and T. Huang. Trace ratio vs. ratio trace for dimensionality reduction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR07)*, 2007. (Cited on pages 145 and 153.)

[210] J.H. Wang, P.K. Hopke, T.M. Hancewicz, and S.-L. Zhang. Application of modified alternating least squares regression to spectroscopic image analysis. *Analytica Chimica Acta*, 476:93–109, 2003. (Cited on pages 7, 18 and 39.)

[211] Y. Washizawa, H. Higashi, T. Rutkowski, T. Tanaka, and A. Cichocki. Tensor based simultaneous feature extraction and sample weighting for EEG classification. In *Proceedings of the 17th international conference on Neural information processing: models and applications - Volume Part II*, ICONIP'10, pages 26–33, Berlin, Heidelberg, 2010. Springer-Verlag. (Cited on page 169.)

[212] Stefan M. Wild, James H. Curry, and Anne Dougherty. Improving non-negative matrix factorizations through structured initialization. *Pattern Recognition*, 37(11):2217–2232, November 2004. (Cited on page 17.)

[213] Sh. Yan, D. Xu, Q. Yang, L. Zhang, X. Tang, and H.-J. Zhang. Discriminant analysis with tensor representation. In *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, pages 526–532, 2005. (Cited on page 143.)

[214] Y. Yu and A. P. Petropulu. PARAFAC-based blind estimation of possibly underdetermined convolutive MIMO systems. *IEEE Trans. on Signal Processing*, 56:111–124, 2007. (Cited on pages 5, 88, 130 and 131.)

[215] R. Zdunek and A. Cichocki. Nonnegative matrix factorization with constrained second-order optimization. *Signal Processing*, 87:1904–1916, 2007. (Cited on pages 7, 18 and 39.)

[216] R. Zdunek and A. Cichocki. Nonnegative matrix factorization with quadratic programming. *Neurocomputing*, 71(10–12):2309–2320, 2008. (Cited on page 40.)

[217] R. Zdunek, A.-H. Phan, and A. Cichocki. Damped Newton iterations for nonnegative matrix factorization. *Australian Journal of Intelligent Information Processing Systems (ICONIP'10)*, 12(1):16–22, 2010. (Cited on page 18.)

[218] W. Zhang, Zh. Lin, and X. Tang. Tensor linear Laplacian discrimination (TLLD) for feature extraction. *Pattern Recognition*, 42(9):1941 – 1948, 2009. (Cited on page 143.)