On Dynamic Tensor Decompositions

Shuo Zhou

ORCID: 0000-0002-6475-3293

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE

May 2019

Produced on archival quality paper.

Abstract

Tensors are multi-way arrays that can be used to represent multi-dimensional data such as video clips, time-evolving graphs/networks, and spatio-temporal data like fMRI. In recent years, CANDECOMP/PARAFAC (CP) decomposition, one of the most popular tools for feature extraction, dimensionality reduction and knowledge discovery on multi-way data, has been extensively studied and widely applied in a range of scientific fields and achieved great success.

Today's data are often dynamically changing over time. In such dynamic environments, a data tensor may be expanded, shrunk or modified on any of its dimensions. Since tensor decomposition is usually the first and necessary step for down-streaming data analyzing tasks, it is crucial to always keep the latest decomposition of a dynamic tensor available given its previous decomposition and the new data. However, tracking the CP decomposition for such dynamic tensors is a challenging task, due to the large scale of the tensor and the high velocity of new data arriving. In addition, data sparsity also increases the difficulty of decomposing dynamic tensors, since special considerations have to be given for efficiency purpose. Furthermore, in order to incorporate domain knowledge and to obtain meaningful and interpretable decompositions, constraints such as non-negativity, ℓ_1 and ℓ_2 regularizations are often used on top of ordinary CP formulation, while how to address them in a dynamic setting is still an open question.

Traditional solving algorithms, such as Alternating Least Squares (ALS), are usually static methods and cannot be directly applied to dynamic tensors due to their poor efficiency. Additionally, existing online approaches have various issues, limiting their applications on dynamic tensors in the real world. Moreover, most of current online techniques are designed for dense tensors while encounter significant efficiency and scalability issues for sparse data.

To fill this gap, in this thesis we investigate the problem of how to adaptively track the CP decomposition of a tensor that is dynamically changing over time without computing from scratch, given that the previous decomposition is known. We study both *dense* and *sparse* dynamic tensors. Two types of dynamic changes are discussed, including *slice-wise* (new data are new slices that are appended on the time mode, *e.g.*, time-evolving networks) and *element-wise* (each update is an individual cell value at arbitrary position, *e.g.*, new ratings in recommender systems) updates. First, we propose *OnlineCP* algorithm to address the slice-wise updates on dense dynamic tensors. By using complementary matrices to store the historical information and caching the intermediate results, our algorithm shows significant improvements than state-of-the-arts methods on several aspects, including decomposition quality, efficiency, scalability and stability. Second, we design a specific algorithm for sparse tensors with slice-wise updates, *OnlineSCP*. By taking advantage of data sparsity, our approach is able to produce high quality decompositions that are comparable to the most accurate algorithm, ALS, whilst at the same time achieving dramatic speed-up and memory-saving, compared to existing approaches. Lastly, for sparse tensors, a new algorithm to handle element-wise updates on-the-fly is proposed. We also explore how to incorporate constraints into this dynamic setting and validate the merits of our algorithm in both synthetic and several context-aware recommender system datasets.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Shuo Zhou, May 2019

Preface

This thesis has been written at the School of Computing and Information Systems, The University of Melbourne. The main parts of this thesis are the Chapter 3, 4 and 5. They are based on published papers and I declare that I am the primary author and have contributed to more than 50% of each of the following papers.

1. Zhou, S., Vinh, N.X., Bailey, J., Jia, Y. and Davidson, I., 2016, August. Accelerating online CP decompositions for higher order tensors. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1375-1384). ACM.

2. Zhou, S., Erfani, S. and Bailey, J., 2018, November. Online CP decomposition for sparse tensors. In *2018 IEEE International Conference on Data Mining (ICDM)* (pp. 1458-1463). IEEE.

3. Zhou, S., Erfani, S.M. and Bailey, J., 2017, November. SCED: a general framework for sparse tensor decomposition with constraints and elementwise dynamic learning. In *2017 IEEE International Conference on Data Mining (ICDM)* (pp. 675-684). IEEE.

During the course of this thesis, several fruitful collaborations have also led to the following publications. They are not discussed within this thesis.

1. Vinh, N.X., **Zhou, S.**, Chan, J. and Bailey, J., 2016. Can high-order dependencies improve mutual information based feature selection?. *Pattern Recognition*, 53, pp.46-58.

2. Ma, X., Wijewickrema, S., **Zhou, S.**, Zhou, Y., Mhammedi, Z., O'Leary, S. and Bailey, J., 2017, August. Adversarial generation of real-time feedback

with neural networks for simulation-based training. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence* (pp. 3763-3769). AAAI Press.

3. Ma, X., Wijewickrema, S., Zhou, Y., **Zhou, S.**, OLeary, S. and Bailey, J., 2017, September. Providing effective real-time feedback in simulation-based surgical training. In *International Conference on Medical Image Computing and Computer-Assisted Intervention (IJCAI)* (pp. 566-574). Springer, Cham.

4. Ma, X., Wang, Y., Houle, M.E., **Zhou, S.**, Erfani, S.M., Xia, S.T., Wijewickrema, S. and Bailey, J., 2018. Dimensionality-driven learning with noisy labels. In *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 3361-3370).

Acknowledgements

PhD study is the toughest challenge that I have taken so far, but it is also a wonderful journey that is full of amazing and vibrant memories. It would not be possible for me to complete this thesis, without the great help and support I received from my supervisors, colleagues, family and friends. Here I would like to take this opportunity to acknowledge and thank everyone who helped me throughout this journey.

First of all, I would like to express my greatest gratitude to my supervisors: Prof. James Bailey, Dr. Sarah Erfani, and Dr. Vinh Nguyen, for their constant support and help during the entire PhD. I sincerely thank James for giving me this unique opportunity to be his student. He has always been supportive with all the patient guidance and insightful suggestions. From him, I have learned not only the knowledge and skills to be a qualified researcher, but also his enthusiasm and kindness as a person. I would like to thank Sarah for all the technical discussions we have had that helped me gain better and deeper understanding of my research. She has also been a warm-hearted friend who helped me shape my career path with her experience and advice. Special thank goes to Vinh, who was the supervisor for my Master thesis and the first year of PhD. As a great mentor, he has opened up the gate of research world to me and educated me step by step with his expertise. I am more than fortunate to have these brilliant academics and wonderful persons as supervisors of my PhD.

I would like to thank Prof. Udaya Parampalli for being the chair of my advisory committee and for his guidance and advice that kept me on the right track throughout my candidature. Meanwhile, I would like to acknowledge the University of Melbourne, the School of Computing and Information Systems, the former and current Head of School, Prof. Justin Zobel and Prof. Uwe Aickelin, for the great working environment and community I had. In addition, thanks to Dr. Mukesh Mohania from IBM research, who helped me win the PhD fellowship that provided huge support and resources to my research. I am also grateful to Prof. Tony Wirth for giving me the chance to work with him as a tutor, and Prof. Dinh Phung and Dr. Nayyar Zaidi from Credit AI Lab, Trusting Social, for taking me as a research intern and mentoring me with their proficiency.

In addition, I would like to say thank you to all my colleagues in our lab: Xingjun, Yunzhe, Florin, Donia, Sukarna, Jiazhen, Yang, Simone, Sergey, Mohadesen, Yamuna, Anam and Sobia. Thanks for all the wonderful moments and it was a great honor to have them as companions during the PhD journey.

Finally, I would like to sincerely thank my family for their unlimited support and unconditional love. Of course, finishing this PhD journey would not be possible without the tremendous support and constant encouragement from my dear wife, Xi. Thank you for believing in me.

Contents

List of Symbols

a	scalar
$\mathbf{a}$	vector
$\mathbf{A}$	matrix
$\mathcal{X}$	tensor
$\mathbf{a}_{i\cdot}$	i-th row vector of $\mathbf{A}$
$\mathbf{a}_{\cdot j}$	j-th column vector of $\mathbf{A}$
a_{ij}	(i,j)-th element of $\mathbf{A}$
$\mathbf{A}^\top$	transpose of $\mathbf{A}$
$\mathbf{A}^{-1}$	inverse of $\mathbf{A}$
$\mathbf{A}^\dagger$	pseudoinverse of $\mathbf{A}$
$\|\|\mathbf{A}\|\|$	Frobenius norm of $\mathbf{A}$
$\mathbf{A}^{(n)}$	n-the matrix of a sequence of matrices
$\circ$	outer product
$\odot$	Khatri-Rao product
$\circledast$	Hadamard product
$\oslash$	element-wise division
$\bigodot_{i=1}^{N}\mathbf{A}^{(i)}$	$\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(i)} \odot \cdots \odot \mathbf{A}^{(1)}$
$\circledast_{i=1}^{N}\mathbf{A}^{(i)}$	$\mathbf{A}^{(N)} \circledast \cdots \circledast \mathbf{A}^{(i)} \circledast \cdots \circledast \mathbf{A}^{(1)}$
$\mathbf{X}_{(n)}$	mode-n unfolding of $\mathcal{X}$
$\times_n$	n-mode product
$[\![\bullet]\!]$	CP decomposition operator
R	number of components
N	order of tensor
I_n	length of the n-th mode of $\mathcal{X}$

List of Figures

List of Tables

Chapter 1

Introduction

Multi-dimensional data are quite common to see nowadays, from video clips spread in people's daily life [Mu et al., 2014], to time-evolving graphs and networks such as social networks [Anandkumar et al., 2014], to spatio-temporal data like fMRI [Davidson et al., 2013, Ma et al., 2016]. Traditional data modelling tools such as matrices and vectors are not ideal for representing such data since the interaction among different dimensions is omitted. The *Tensor*, a multi-way generalization of the matrix, is a more useful data representation format for those data. For instance, in context-aware recommender systems, users tend to interact with different items in various ways depending on different context: they watch news clips on the way to work on their smart phones and tablets while spend a Saturday night on the big screen television with a family-friendly movie. Such events represent different behavior patterns of users under different circumstances, and can be effectively modeled by a 3^{rd}-order tensor as $user \times item \times context$. Whereas in traditional recommender systems that only maintains the user-item interactions with a matrix, these valuable contextual information is missing.

Working with tensors is not easy due to the complex relationships among different dimensions. Similar to matrix analysis tools such as Principal Component Analysis (PCA) and Singular Value Decomposition (SVD), *tensor decomposition* is a popular approach for feature extraction, dimensionality reduction and knowledge discovery on multi-way data. It has been extensively studied and widely applied in various fields of science, including chemometrics [Acar et al., 2011a], signal processing [Cichocki et al., 2015], computer vision [Hu et al., 2011, Vasilescu and Terzopoulos, 2002b], graph and network analysis [Dunlavy et al., 2011, Liu et al., 2012b] and time series analysis [Cai et al., 2015]. One of the most popular tensor decomposition algorithms is *CANDECOMP/PARAFAC* (CP) decomposition, which is an extension of the well known Matrix Factorization (MF) technique to tensors. Similar to MF, CP decomposition is also a low-rank latent factor model that seeks the latent structure underneath the complex data source. By applying CP decomposition, the complicated tensor data can be simplified and latent representations can be learned for different entities in the tensor, which can be further used for more in-depth analysis such as clustering, forecasting and classification [Kolda and Bader, 2009, Papalexakis et al., 2017].

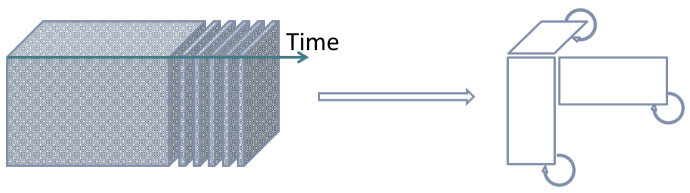
In the era of big data, data is often dynamically changing over time, and a large volume of new data can be generated at high velocity. Take the previous $user \times item \times context$ for example, new users are constantly joining the system and existing users may choose to terminate their services, a large number of new items are generated every day and added to the database, new interactions, (user, item, context) tuples, are appended in existing data tensor at high frequency. Given such a time-evolving tensor, it is clear that one cannot keep using the same decomposition obtained from previous timestamp since it will easily be outdated with the increase of time. As a result, it is desirable to have efficient algorithms to maintain the latest decomposition when new data is presented. However, most existing tensor decomposition approaches are designed for static tensors and fail in this highly dynamic scenario due to their poor efficiency performance, in terms of both time and space. For example, the typical algorithm for CP decomposition, *Alternating Least Squares* (ALS), is a method that computes the decomposition of a tensor from scratch and has linear complexity to the overall size of the input, which is too expensive and impractical to be applied on dynamic tensors.

In addition, the sparsity of the data is also an important factor that needs to be considered when dealing with dynamic tensors. Some real-world tensors are dense ones (*e.g.* videos, fMRI) while some have high sparsity (*e.g.* social networks, traffic graphs, recommender systems). The methods work well on dense dynamic tensors are not necessarily having good performance on sparse ones. In fact, algorithms designed for dense tensors usually encounter various issues in terms of both efficiency and memory usage when applied to sparse data, since sparse tensors are often of large-scale. As a result, special consideration should be given to sparse dynamic tensors in order to obtain better performance.

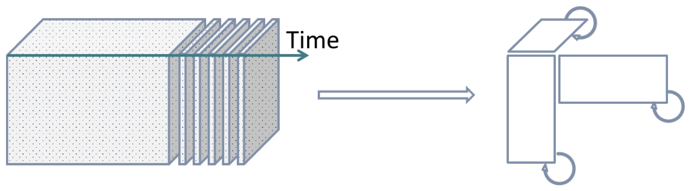
Finally, it is quite common to see that constraints such as non-negativity and sparseness are widely used in addition to latent factor models such as MF and CP decomposition. Domain knowledge usually exists in various real-world tensors and this prior understanding of data can be modeled as constraints imposed to the conventional decomposition, helping to improve the interpretability of the produced latent factors. For example, in computer version [Lee and Seung, 2001], non-negativity is extensively used to ensure a part-based basis and the input images can be viewed as additive combinations of a selected set of facial components. Consequently, having a dynamic tensor decomposition algorithm that takes constraints into consideration is of great benefit for building a meaningful and effective model.

1.1 Research Questions

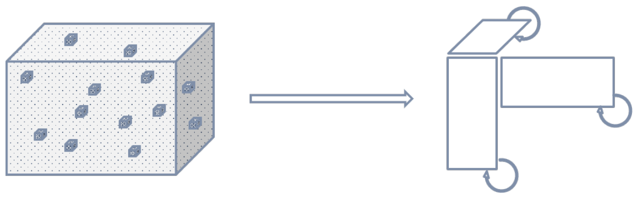
In this thesis, the general research problem we aim to address is *given the existing decomposition of a tensor, how can we efficiently obtain its new decomposition when new data is added, without computing it from scratch*. We are particularly interested into CP decomposition because of its popularity and flexibility. Both *dense* and *sparse* dynamic tensors are discussed in this work and we focus on two most typical types of dynamic updates: 1) *slice-wise* updates, where the new data is incrementally appended on the time mode, while the other dimensions remain unchanged; 2) *element-wise*



(a) Slice-wise updates on dense dynamic tensors



(b) Slice-wise updates on sparse dynamic tensors



(c) Element-wise updates on sparse dynamic tensors

Figure 1.1: Research questions

updates, which are consisted of individual values that are added to existing tensor at arbitrary positions. In addition, we also look at constrained case on top of our main focus on dynamic tensor decompositions. According to the sparseness of the dynamic tensor and the categorization of the dynamic updates, we divide the overall problem into the following three sub-questions, as demonstrated in Figure 1.1:

1. **Slice-wise updates on dense dynamic tensors:** Given an existing CP decomposition that approximates a dense tensor at a timestamp, a new incoming dense tensor that has the same non-temporal modes while much smaller length of time, a new tensor can be formed as appending the new data to the current one along the time mode. Can we find the decomposition of the newly formed tensor efficiently by just adaptively updating the current decomposition? Examples of this type of dynamic tensors include fMRI and videos.

2. **Slice-wise updates on sparse dynamic tensors:** Similar problem as

above is discussed while this time both the existing and new data are highly sparse. A motivating example is analyzing time-evolving networks/graphs such as social networks data where new snapshots of user interactions are added to the current data at high velocity.

3. **Element-wise updates on sparse dynamic tensors:** Given a sparse tensor that stores the data at current state and its CP decomposition, how a new value that is added to an empty position of the existing data will affect the decomposition? The main application of this can be found in context-aware recommender systems where a large number of new ratings are generated frequently. In addition, considering the importance role of constraints played in the field of collaborative filtering, we also look at how to do this in a constrained setting.

It should be noted that above three research projects did not fully cover the whole problem space of dynamic tensor decomposition. For example, a dynamic tensor may have new data at multiple dimensions at the same time, existing slice or cells of a tensor may be removed or updated partially or entirely. These dynamic behaviours will also have impact to the decomposition of the tensor. Another less common scenario is that for a dense tensor, element-wise updates may be added to the date (*e.g.*, new records are added to replace the of wrong measurements in a $sensor \times type \times time$ tensor). However, we believe that the selected three research themes are the most common cases happened in real-world scenario and the concepts and insights we gained in them are of great potential to be applied in other types of dynamic learning problem, which is left for future work.

1.2 Thesis Overview

Chapter 2 provides the introduction to tensor-related notations, preliminaries and a literature review on the existing related methods and also builds a foundation for better understanding the proceeding chapters.

In Chapter 3, we study the problem of finding the CP decompositions of dynamic tensors with slice-wise updates. By analyzing the structure of the data tensor, we show that a key step in static ALS algorithm that dominates the computation cost can be decomposed into two parts. One is related to historical data while the other depends only on the size of the new data. To make use of this property, two complimentary matrices are created for each non-temporal modes, which are used for storing historical information and can be incrementally updated at a low cost. Based on this, we propose a new algorithm, OnlineCP, which is highly efficient for processing slice-wise update for a tensor. In addition, to generalize OnlineCP to higher-order tensors, we further design a procedure to calculate the Khatri-Rao products required for the updates of all modes simultaneously, by making use of intermediate results. As validated by various synthetic and real-world datasets, our algorithm achieves significant improvement over the state-of-the-arts, in terms of effectiveness, efficiency and scalability.

Chapter 4 discusses the same slice-wise updates for dynamic tensor decompositions as before. While the main focus on this work is sparse tensors. When applying OnlineCP algorithm proposed in previous chapter to sparse tensors, its performance, in terms of both time ans space usage, quickly drops to an unacceptable level and can be even worse than the static decomposition algorithms that are specifically designed for sparse tensors. To address this issue, we show that the same principle that partitions the computation into historical and new data parts is still applicable to sparse dynamic tensors, while the exact calculating of Khatri-Rao product is unnecessary and wasteful. As a result, we speed up the algorithm via applying a method that takes data sparsity into account. Moreover, the complimentary matrices used in OnlineCP are redesigned for better memory efficiency. Overall, a new algorithm, OnlineSCP, is proposed particularly for sparse dynamic tensors with slice-wise updates. Extensive experiments confirms its superior performance on saving computational time and memory usage.

In Chapter 5, the problem of finding the CP decomposition of sparse tensors with element-wise dynamic updates is studied. To be more specific, this research problems contains three parts: 1) we observe that the zeros in sparse tensors may play different roles depending on the applications. Take traffic network for example where a tensor as *intersection* $\times$ *intersection* $\times$ *time* is formed to store data, zero is the true observation indicating no direct road can be found between the corresponding intersection pair, which should be approximated by the model in the same way as non-zero values. While in social networks zeros should be treated as missing values since any two users are potential friends and should not be approximated. To model this difference, we propose a weighted decomposition formulation, which uniformly models different application scenarios by choosing different weights. 2) We look at constraints such as non-negativity and regularizations like ℓ_1-norm for sparseness and ℓ_2-norm to prevent overfitting. To solve the constrained decomposition problem, we apply an algorithm that estimates parameters one by one. Based on this algorithm, non-negativity can be simply handled as replacing negative parameters with zero, and regularizations can be imposed as penalty terms to the weighted decomposition formulation and easily solved. 3) To handle new data that is consisted of elements that will be added, we choose to perform local update in loading matrices by only learning new parameters in the corresponding rows of a new element. Overall, as verified by experiments on several datasets, the proposed algorithm delivers considerably better performance than the state-of-the-arts.

Finally, a summary of this thesis is presented in Chapter 6, along with a discussion on potential future works.

Chapter 2

Background

In this chapter, we discuss the background of our research, review current literature, and identify the limitations and gaps that we will address in the proceeding chapters. We start with an introduction of tensor-related terminologies and operations, followed by a brief review of different tensor decomposition algorithms and their applications. After that, we discuss CP decomposition, the model that we are particularly interested in this thesis, into more details. Finally, we introduce the dynamic tensor decomposition problem in the last section and review existing works and discuss their limitations.

2.1 Notation and Tensor Algebra

This section introduces basic notation and operations that are widely used in the field. A summary of symbols that we use through the whole thesis can be found in the List of Symbols.

A tensor is a multi-way array, which is a generalization of vector and matrix. An example is illustrated in Figure 2.1. We denote vectors by boldface lowercase letters, e.g., $\mathbf{a}$, matrices by boldface uppercase letters, e.g., $\mathbf{A}$, and tensors by boldface Euler script letters, e.g., $\boldsymbol{\mathcal{X}}$. The *order* of a tensor, also known as the number of *ways* or *modes*, is its number of dimensions. For example, vectors and matrices are 1^{st}-order and 2^{nd}-order tensors, re-

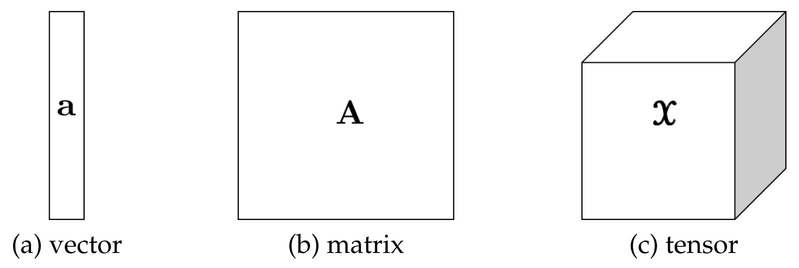


Figure 2.1: A tensor is a multi-way generalization of vector and matrix

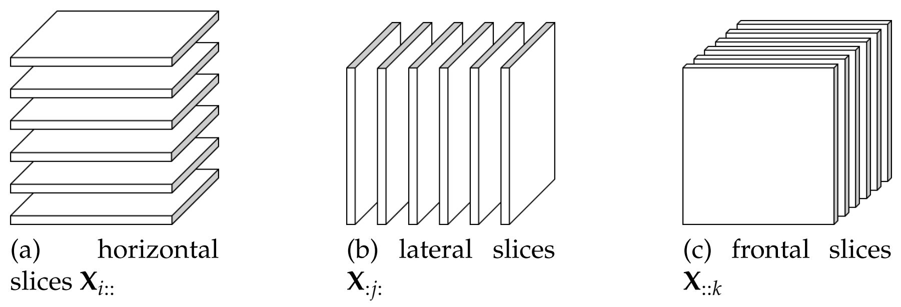

(a) horizontal slices $\mathbf{X}_{i::}$
(b) lateral slices $\mathbf{X}_{:j:}$
(c) frontal slices $\mathbf{X}_{::k}$

Figure 2.2: Slices of a 3^{rd}-order tensor

spectively; and a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is an N^{th}-order one consisting of real numbers and the cardinality of its n-th order, $n \in [1, N]$ is I_n. We refer to tensors with more than 3 modes as *higher*-order ones.

In order to index matrix $\mathbf{A} \in \mathbb{R}^{I \times J}$, we denote its (i, j)-th element by a_{ij}, i-th row vector by $\mathbf{a}_{i:}$, and j-th column vector by $\mathbf{a}_{:j}$. Similarly, the (i, j, k)-th element of a 3^{rd}-order tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ is denoted by x_{ijk} and the $(i_1, i_2, \ldots, i_N)$-th element of an N^{th}-order tensor $\mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is denoted by $y_{i_1 i_2 \ldots i_N}$.

A *slice* of a 3^{rd}-order tensor $\mathcal{X}$ is a two-dimensional matrix, defined by fixing the index of a particular mode. For example, as shown in Figure 2.2, the horizontal, lateral and frontal slides of $\mathcal{X}$ are denoted by $\mathbf{X}_{i::}, \mathbf{X}_{:j:}$ and $\mathbf{X}_{::k}$. Likewise, the i_n-th slice on the n-th mode of an N^{th}-order tensor $\mathcal{Y}$ is an $(N-1)^{th}$-order tensor, denoted by $\mathcal{Y}_{:\cdots:i_n:\cdots:}$.

The *transpose* of matrix $\mathbf{A}$ is denoted by $\mathbf{A}^\top$ such that $\mathbf{A}^\top_{ij} = \mathbf{A}_{ji}$. The *inverse* of an invertible square matrix $\mathbf{A}$ is denoted by $\mathbf{A}^{-1}$ such that $\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. The Moore-Penrose *pseudoinverse*

[Ben-Israel and Greville, 2003] is a generalization of inverse. Specifically, the pseudoinverse of $\mathbf{A}$ is denoted by $\mathbf{A}^{\dagger}$ such that $\mathbf{A}\mathbf{A}^{\dagger}\mathbf{A} = \mathbf{A}$, where $\mathbf{A}$ can be a non-square matrix.

The *Frobenius norm* of a matrix $\mathbf{A} \in \mathbb{R}^{I \times J}$ is denoted by $||\mathbf{A}||$ and defined as

$$||\mathbf{A}|| = \sqrt{\sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}^2}.$$

The *outer* product of two vectors $\mathbf{a} \in \mathbb{R}^I$ and $\mathbf{b} \in \mathbb{R}^J$ is denoted by $\mathbf{a} \circ \mathbf{b}$. This is a matrix of size $I \times J$, where its (i, j)-th entry is $a_i b_j$. This definition can be extended to more than two vectors. For example, $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$ forms a 3^{rd}-order tensor $\mathcal{X}$ where $x_{ijk} = a_i b_j c_k$. Furthermore, we refer to such tensor that can be expressed by the outer product a series of vectors as *rank-one* tensor.

The *Khatri-Rao* product [Kolda and Bader, 2009] of two matrices $\mathbf{A} \in \mathbb{R}^{I \times K}$ and $\mathbf{B} \in \mathbb{R}^{J \times K}$ is denoted by $\mathbf{A} \odot \mathbf{B}$, the output of which is a $IJ \times K$ matrix defined as

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{b}_{\cdot 1} & a_{12}\mathbf{b}_{\cdot 2} & \cdots & a_{1K}\mathbf{b}_{\cdot K} \\ a_{21}\mathbf{b}_{\cdot 1} & a_{22}\mathbf{b}_{\cdot 2} & \cdots & a_{2K}\mathbf{b}_{\cdot K} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{b}_{\cdot 1} & a_{I2}\mathbf{b}_{\cdot 2} & \cdots & a_{IK}\mathbf{b}_{\cdot K} \end{bmatrix}.$$

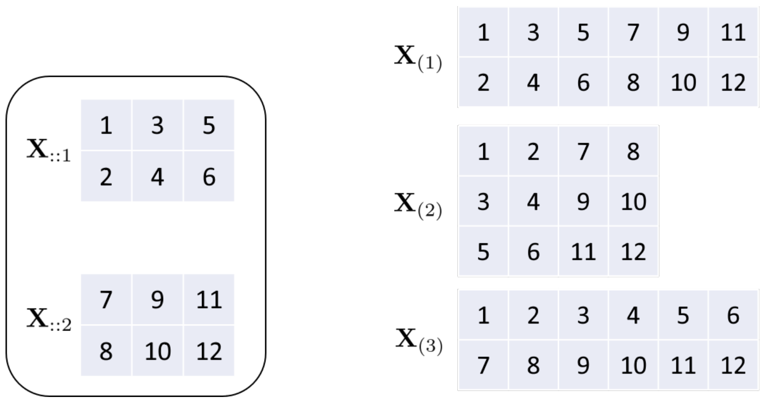
The *Hadamard* product of two equal size matrices $\mathbf{A}$ and $\mathbf{B}$ is denoted by $\mathbf{A} \circledast \mathbf{B}$, which is their element-wise product. Similarly, we denote their element-wise division by $\mathbf{A} \oslash \mathbf{B}$.

It should be noted that the following properties [Smilde et al., 2005] are used in the discussions of proceeding chapters:

$$(\mathbf{A} \odot \mathbf{B})^{\top}(\mathbf{A} \odot \mathbf{B}) = \mathbf{A}^{\top}\mathbf{A} \circledast \mathbf{B}^{\top}\mathbf{B} \tag{2.1}$$

$$(\mathbf{A} \odot \mathbf{B})^{\dagger} = (\mathbf{A}^{\top}\mathbf{A} \circledast \mathbf{B}^{\top}\mathbf{B})^{\dagger}(\mathbf{A} \odot \mathbf{B})^{\top} \tag{2.2}$$

Furthermore, let $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$ represent a list of N matrices. The Khatri-Rao product series $\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(i)} \odot \cdots \odot \mathbf{A}^{(1)}$ is denoted by $\bigodot_{i=1}^{N} \mathbf{A}^{(i)}$



(a) $\mathcal{X} \in \mathbb{R}^{2\times3\times2}$ (b) $\mathbf{X}_{(n)}, n \in (1,2,3)$

Figure 2.3: An example of mode-n unfolding

and the Hadamard product sequence $\mathbf{A}^{(N)} \circledast \cdots \circledast \mathbf{A}^{(i)} \circledast \cdots \circledast \mathbf{A}^{(1)}$ is denoted by $\circledast_{i=1}^{N}\mathbf{A}^{(i)}$.

Tensor unfolding, or *matricization*, is a procedure that transforms a tensor into a matrix [Bader and Kolda, 2006]. This is usually done by reordering the element of a tensor into a matrix. One type of unfolding that has been found particularly useful is *mode-n* unfolding. The mode-n unfolding of $\mathcal{X}$ is denoted by $\mathbf{X}_{(n)}$. Generally speaking, given an N^{th}-order tensor $\mathcal{X} \in \mathbb{R}^{I_1\times\cdots\times I_N}$, $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n\times\prod_{i\neq n}^{N} I_i}$ can be obtained by permuting the dimensions of $\mathcal{X}$ as $[I_n, I_1, \ldots, I_{n-1}, I_{n+1}, \ldots, I_N]$ and then reshaping the permuted tensor into a matrix of size $I_n \times \prod_{i\neq n}^{N} I_i$. An example of mode-$n$ unfolding of a 3^{rd}-order tensor can be found in Figure 2.3.

The *n-mode* product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ with a matrix $\mathbf{U} \in \mathbb{R}^{J\times I_n}$ is denoted by $\mathcal{X} \times_n \mathbf{U}$ and is of size $I_1 \times I_2 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$. Element-wisely, we have

$$(\mathbf{X} \times_n \mathbf{U})_{i_1\cdots i_{n-1} j i_{n+1}\cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2\cdots i_N} u_{j i_n}.$$

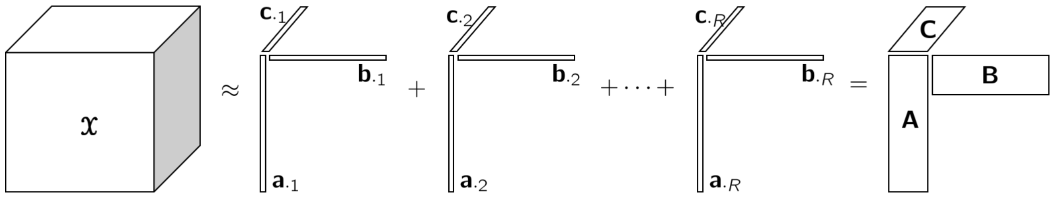
Equivalently, this can also be expressed in terms of unfolded tensors as

$$\mathcal{Y} = \mathcal{X} \times_n \mathbf{U} \Longleftrightarrow \mathbf{Y}_{(n)} = \mathbf{U}\mathbf{X}_{(n)}$$

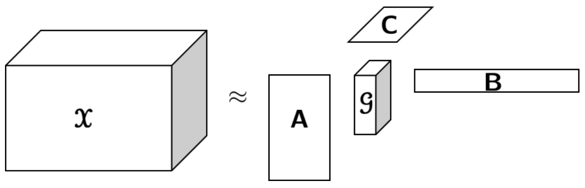
2.2 Tensor Decompositions and Applications

There is a rich variety of tensor decomposition models existing in the literature. In this section, we provide an overview of the most widely used decompositions and their applications. It should be noted that due to tensor decomposition is such a broad research area, here we are not aiming at conducting a comprehensive and in-depth survey of the entire field. Instead, in this section we only focus on providing a high level understanding of how different decompositions work and illustrating their usefulness via a general introduction on their applications in a wide range of domains such as signal processing, data mining, and machine learning. The main idea of some of the most commonly used decomposition models are illustrated in Figure 2.4. In addition, for the decomposition method we are particularly interested in this thesis, CP decomposition, we discuss more details of it in §2.3, followed by an explanation and discussion on the research question we focus in this thesis: how to efficiently find the CP decompositions of dynamic tensors.

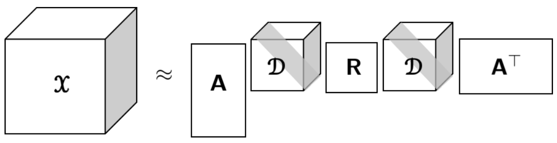
For readers who are interested into and willing to gain more insights about tensor decomposition models other than CP, there are several studies and surveys that are worthy being mentioned. A good starting point for beginners is [Kolda and Bader, 2009], which is a highly-cited and clearly-written survey that provides very detailed overview of different tensor decompositions. It starts with basic concepts and figure illustrations to help readers establish multi-way thinking and then covers basic tensor decomposition models and the typical solving algorithms for them. Compared to this nine-year-old review, recent surveys [Grasedyck et al., 2013, Cichocki et al., 2015, Sidiropoulos et al., 2017, Papalexakis et al., 2017] include more advanced models such as tensor networks that have been proposed in recent years and more in-depth theoretical analysis underneath different models. These are nice resources to keep up-to-date with the recent advance and research trends in tensor decompositions. In addition, there are a few valuable reviews for audiences that are from a different background other than machine learning and data mining. For example, [Comon, 2014]


$\mathcal{X}$
$\approx$
$\mathbf{c}_{\cdot 1}$ $\mathbf{b}_{\cdot 1}$ $\mathbf{a}_{\cdot 1}$
$+$
$\mathbf{c}_{\cdot 2}$ $\mathbf{b}_{\cdot 2}$ $\mathbf{a}_{\cdot 2}$
$+\cdots+$
$\mathbf{c}_{\cdot R}$ $\mathbf{b}_{\cdot R}$ $\mathbf{a}_{\cdot R}$
$=$
$\mathbf{C}$ $\mathbf{B}$ $\mathbf{A}$

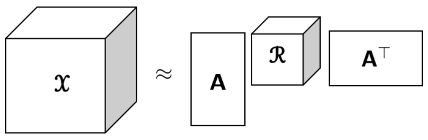
(a) CP decomposition



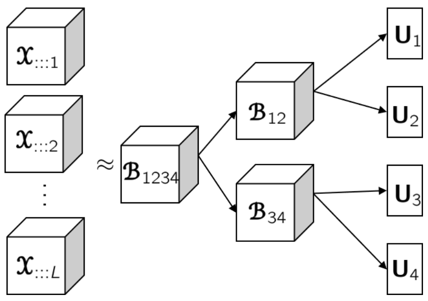
(b) Tucker decomposition



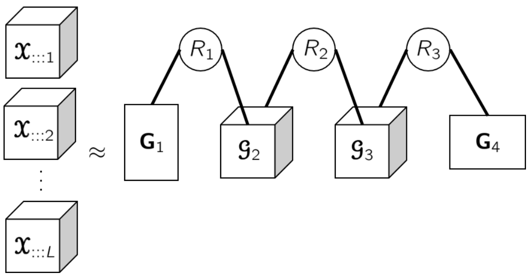
(c) DEDICOM decomposition



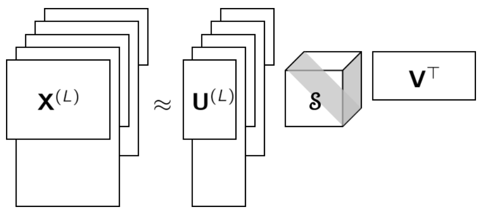
(d) RESCAL decomposition



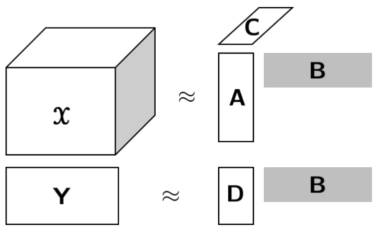
(e) Hierarchical tucker decomposition



(f) Tensor train decomposition



(g) PARAFAC2 decomposition



(h) CMTF decomposition

Figure 2.4: Different types of tensor decompositions

offers an introduction to tensors and tensor decompositions from a more mathematical point of view. Similarly, [Hackbusch, 2012, Vervliet et al., 2014] provide an overview of tensor related techniques in numerical analysis and scientific computing, and a general but clear introduction to tensor decompositions can be found in some highly-cited chemometrics publications such as [Bro, 1997, Smilde et al., 2005].

2.2.1 CP Decomposition and its Applications

In the literature, CP decomposition has been independently proposed and re-discovered by several researchers under different names [Hitchcock, 1927, Carroll and Chang, 1970, Harshman, 1970]. Basically, as shown in Figure 2.4a, a 3rd-order tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ can be expressed by the sum of R rank-one tensors by CP decomposition as

$$\mathcal{X} \approx \sum_{r=1}^{R} \mathbf{a}_{\cdot r} \circ \mathbf{b}_{\cdot r} \circ \mathbf{c}_{\cdot r}, \tag{2.3}$$

where $\mathbf{a}_{\cdot r} \in \mathbb{R}^{I}, \mathbf{b}_{\cdot r} \in \mathbb{R}^{J}, \mathbf{c}_{\cdot r} \in \mathbb{R}^{K}$. R is the number of components or latent factors. The minimal R that yields equality in Eq. (2.3) is called the *rank* of the tensor and the exact CP decomposition with such R is called *rank decomposition*. Under this framework, each rank-one tensor can be interpreted as a latent concept/topic/group/cluster of the data. Alternatively, one can understand CP decomposition from another angle by stacking the vectors for the same mode into a matrix and constructing the so-called *loading* or *factor* matrices as

$$\begin{aligned}
\mathbf{A} &= [\mathbf{a}_{\cdot 1}, \mathbf{a}_{\cdot 2}, \ldots, \mathbf{a}_{\cdot R}] \in \mathbf{R}^{I \times R} \\
\mathbf{B} &= [\mathbf{b}_{\cdot 1}, \mathbf{b}_{\cdot 2}, \ldots, \mathbf{b}_{\cdot R}] \in \mathbf{R}^{J \times R} \\
\mathbf{C} &= [\mathbf{c}_{\cdot 1}, \mathbf{c}_{\cdot 2}, \ldots, \mathbf{c}_{\cdot R}] \in \mathbf{R}^{K \times R}.
\end{aligned}$$

Under this representation, each loading matrix can be understood as the linear mapping from a particular mode to the latent space or an embedding of original information into a certain mode via a projection through other two factors. For convenience, we denote a CP decomposition of a 3rd-order tensor by $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$.

Due to its ease of interpretation, CP decomposition has became one of the most popular tensor decomposition techniques that has been extensively and widely applied in different fields. Earlier applications of CP decomposition can be found in a range of domains other than data mining. For example, [Carroll and Chang, 1970] use CP decomposition in psychometrics on tensors that are consisted of similarity or dissimilarity matrices from a variety of subjects. In [Harshman, 1970], an $individual \times vowel \times formant$ tensor is analyzed by CP in order to obtain explainable factors. In chemometrics, CP decomposition has been proven to be useful in the modeling of fluorescence excitation-emission data and its usage can be found in [Appellof and Davidson, 1981, Andersen and Bro, 2003, Smilde et al., 2005]. CP decomposition is also a popular analyzing tool for Neuroscientists. [Mocks, 1988, Field and Graupe, 1991] apply CP decomposition on event-related potentials in brain imaging. [Andersen and Rayens, 2004, Davidson et al., 2013, He et al., 2014] use CP decomposition on fMRI data, which is a series of brain scans over time and can be organized with a 4th-order tensor as $voxel \times voxel \times voxel \times time$. Similar to fMRI, electroencephalogram (EEG) is also a typical type of data being analyzed by CP decomposition. [Acar et al., 2007] study EEG data from patients with epilepsy and use CP decomposition to localize the origin of the seizure. [Mørup et al., 2006, Mørup et al., 2008, Mørup et al., 2011] propose a few modified CP models such as shift invariant CP (Shift-CP) and convolutional CP (Conv-CP) by taking the shift and dependencies on the time mode into consideration and demonstrate better reconstruction of EEG data.

From 2005, tensors and CP decomposition have attracted increasing interests from data mining researchers. One of the earliest work is [Acar et al., 2005], where they model the online chat-room data as an $user \times keyword \times time$ tensor and then apply different tensor decompositions, including CP, to demonstrate the usefulness of expressing the data as a tensor. Similarly, [Bader et al., 2008] organize email transactions in the Enron dataset in a *term* $\times$ *author* $\times$ *time* tensor and use CP to decompose it for automatically detecting conversations. In addition, one particular type of data that has been extensively analyzed by CP decomposition is network data that contains additional information such as time and context to form the extra modes. For example, [Papalexakis et al., 2012] apply CP decomposition on large-scale social network data and further perform pattern mining and outlier detection on the decomposed temporal mode. [Dunlavy et al., 2011] apply CP decomposition on temporal link prediction problem, which is to predict future links for time-evolving networks based on historical data. It has been shown in their work that periodic behaviors in the links can be effectively predicted by tensor-based techniques. Community detection problem is also tackled with CP decomposition on time-evolving networks [Araujo et al., 2014] and multi-view social networks [Papalexakis et al., 2013]. [Maruhashi et al., 2011] apply CP decomposition on network traffic dataset from Lawrence Berkeley National Labs (LBNL). They use the decomposed temporal mode to detect spikes in the traffic, which in turn to identify irregular events. Similar task is addressed by [Mao et al., 2014] on more network data and clustering techniques are applied on the factor matrices for different temporal resolutions. Furthermore, applications of CP decomposition can also be found on web data. [Kolda et al., 2005] analyze web graph that is built by hyperlinks across web pages. The anchor text has been used as additional information to formulate a 3^{rd}-order tensor. They extend the HITS algorithm [Kleinberg, 1999] to this tensor by applying CP decomposition to measure the authoritativeness and hubness of each web page. [Agrawal et al., 2015a, Agrawal et al., 2015b] study the similarity and difference between search engines and social networks like Twitter by constructing a tensor as *query* $\times$ *keyword* $\times$ *date* $\times$ *service* and then decomposing it with CP. As a result, different service providers are mapped into the same latent space, which enables a comparison among them. [Kang et al., 2012, Papalexakis et al., 2012] propose efficient distributed CP decomposition algorithms for analyzing the data from the Never-Ending Language Learning project (NELL) [Mitchell et al., 2018], which contains a huge collection of (entity, entity, context) triplets that are crawled from millions of web pages.

In fact, the amount of work that apply CP decomposition as the key component to solve different research problems is much greater than what we summarized above. There are a substantial number of researches can be found in the literature that CP decomposition is applied on critical real-world problems (*e.g.*, healthcare [Ho et al., 2014, Yang et al., 2017], recommender system [Xiong et al., 2010, Rendle, 2010, Frolov and Oseledets, 2017], signal processing [Nion et al., 2010, Cichocki et al., 2015, Zhou et al., 2017]). Overall, it is clear that tensor is a powerful data modelling tool for a lot of real-world data and CP decomposition is of great potential and usage for analyzing them.

2.2.2 Tucker Decomposition and its Applications

Apart from CP decomposition, another extremely popular tensor decomposition technique is Tucker decomposition. For a 3^{rd}-order tensor example, compared to CP where three loading matrices are used to approximate the data, an additional *core* tensor is used in Tucker. This core tensor can be viewed as a compression of the original tensor and the input tensor is reconstructed by multiplying the core tensor with loading matrices along each mode, as shown in Figure 2.4b. Formally speaking, for a 3^{rd}-order tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, Tucker decomposes it as

$$\begin{aligned}\mathcal{X} &\approx \mathcal{G} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \\ &= \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} g_{r_1 r_2 r_3} \mathbf{a}_{\cdot r_1} \circ \mathbf{b}_{\cdot r_2} \circ \mathbf{c}_{\cdot r_3},\end{aligned} \tag{2.4}$$

where $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ is the core tensor, $\mathbf{A} \in \mathbb{R}^{I \times R_1}, \mathbf{B} \in \mathbb{R}^{J \times R_2}, \mathbf{C} \in \mathbb{R}^{K \times R_3}$ are the loading matrices and R_1, R_2, R_3 are the number of components on each mode. Compared Eq. (2.4) to Eq. (2.3), one can see that even though Tucker is a more complex model than CP, both of them are sums of rank-one tensors. The difference is that Tucker is consisted of $R_1 \times R_2 \times R_3$ such rank-one tensors whereas there are only R of them being used in CP. As a result, CP decomposition can be viewed as a special case of Tucker with super-diagonal core tensor, whose off-diagonal elements are zeros. In practice, CP is preferable if the interpretability is essential while Tucker is most often used to compress the tensor into smaller size.

Because of the close relationship between CP and Tucker decompositions, a lot of researches of applying Tucker can be found in similar domains as CP, including chemometrics [Henrion, 1994], psychometrics [Kiers and Mechelen, 2001], signal processing [De Lathauwer and Vandewalle, 2004, Muti and Bourennane, 2005], web search [Sun et al., 2005] and information retrieval [Chang et al., 2013], network and graph analysis [Sun et al., 2006, Sun et al., 2008, Kolda and Sun, 2008, Jiang et al., 2014], data mining and machine learning [Acar et al., 2005, Liu et al., 2005, Savas and Eldén, 2007, Mu et al., 2011, Karatzoglou et al., 2010], *etc.*

A special area that Tucker decomposition received slightly more attentions than CP is computer vision and image processing. One of the first researches on this line is [Vasilescu and Terzopoulos, 2002a]. They propose TensorFaces, which is essentially an application of Tucker to decompose facial images from multiple subjects where each subject has multiple photos that are taken under varying conditions. One example is lighting condition such that the data can be organized into a 3^{rd}-order tensor as $subject \times pixel \times lighting$. Compared to standard PCA-based technique [Vasilescu and Terzopoulos, 2002b], TensorFaces provides significantly better recognition performance and a compression of the original data can be obtained via retaining key facial features while removing other irrelevant effects. [Wang et al., 2003, Wang and Ahuja, 2004] also study the usage of Tucker decomposition on facial images while facial expressions are considered as the extra mode and [Tao et al., 2008] propose to use Tucker to model 3D faces but from a Bayesian point of view. Furthermore, Tucker is also used for other image processing tasks. For example, [Vasilescu and Terzopoulos, 2004] apply Tucker decomposition for texture rendering in images, [Nagy and Kilmer, 2006] use Tucker to approximate Kronecker product for preconditioners in image processing, [Abdallah et al., 2007] propose a method for watermarking videos via Tucker, and [Hu et al., 2011] design an incremental version of Tucker for foreground segmentation and tracking in videos.

2.2.3 Other Decompositions

Although in most of the time, CP and Tucker decompositions are commonly used for analyzing tensors, there are other types of tensor decomposition techniques that have been proposed for tensors with specific properties.

One example is the extension of DEDICOM proposed by [Harshman, 1978] to three-way tensors. It is a specifically designed decomposition for tensors that represent asymmetric relationships between objects. For instance, a tensor $\mathfrak{X} \in \mathbb{R}^{I\times I\times T}$ can be used to record the re-tweets behaviors between users in Twitter where x_{ijt} stands for a transaction that user i retweet one post from user j at time t. Typical decompositions such as CP and Tucker do not consider the fact that in this tensor, both the first and second modes represent the same set of objects and are not able to capture the relationships among them. On the other hand, DEDICOM can be applied to overcome this issue. Specifically, as demonstrated in Figure 2.4c, DEDICOM decomposes this tensor as

$$\mathfrak{X} \approx \mathbf{A}\mathfrak{D}\mathbf{R}\mathfrak{D}\mathbf{A}^\top,$$

where $\mathbf{A} \in \mathbb{R}^{I\times R}$ is the loading matrix and a_{ir} indicates the participation of object i in group r. $\mathfrak{D} \in \mathbb{R}^{R\times R\times T}$ is a tensor consisted of diagonal matrices that are stacked on the time mode. One element in $\mathfrak{D}$, d_{rrt}, indicates the participation of the r-th latent factor at time t. $\mathbf{R} \in \mathbb{R}^{R\times R}$ stands for the interactions between different latent factors. Several applications of DEDICOM can be found in the literature. [Harshman and Lundy, 1992] apply DEIDCOM on an $import \times export \times year$ tensor constructed from trading data among a group of nations. [Lundy et al., 2003] study relationships between chronic back pain treatments via DEDICOM with additional constraints. [Bader et al., 2007] present an application of DEDCIOM on Enron email dataset where data are organized as *sender* $\times$ *receiver* $\times$ *time*.

Similar to DEDICOM, another decomposition technique that is also target on relational tensors is RESCAL [Nickel et al., 2011]. In DEDICOM, only one type of relationship is considered and its temporal variations are studied. However, RESCAL cares more about the relationship types between a set of objects, rather than the changes over time. It is not uncommon to see that one pair of objects can have different types of relationships in-between. Takes social networks for example, user i can follow user j, re-post his/her content, comment on his/her photo, and send message to him/her, *etc.*. Different types of interactions can be found between them. By considering the relationship type as the extra mode, one can construct a tensor $\mathcal{X} \in \mathbb{R}^{I \times I \times K}$, where K is the number of different relationships. As shown in Figure 2.4d, $\mathcal{X}$ can be decomposed by RESCAL as

$$\mathcal{X} \approx \mathbf{A}\mathcal{R}\mathbf{A}^{\top},$$

where $\mathbf{A} \in \mathbb{R}^{I \times R}$ is the loading matrix and R is the number of latent factors. $\mathcal{R} \in \mathbb{R}^{R \times R \times K}$ is an asymmetric tensor and its k-th frontal slice indicates the interactions between latent components given the k-th relationship. Example applications of this model on knowledge based data can be found in [Nickel et al., 2012, Chang et al., 2014].

CP and Tucker decompositions are widely applied to analyze three-way tensors. However, may real-world applications can have tensors with higher order. Tucker decomposition is particularly challenged by high tensor order, as the number of parameters (mainly the size of the core tensor) it has to estimated increases exponentially with the growth of tensor order. In order to deal with this curse of dimensionality, Hierarchical Tucker decomposition (H-Tucker) is proposed by [Hackbusch and Kühn, 2009] and further fulfilled by [Grasedyck, 2010, Kressner and Tobler, 2012, Ballani et al., 2013]. Basically, H-Tucker assumes that there is a hierarchical structure can be found in the modes of the tensor and a binary tree can be constructed. An example of such tree can be found in Figure 2.4e, where the modes of a 4th-order tensor is split into two sets $\{1, 2\}$ and $\{3, 4\}$ and each of them is further divided into sets with one single mode as $\{1\}, \{2\}, \{3\}$ and $\{4\}$. On this tree, the i-th leaf node is an matrix of size $I_i \times R_i$, where I_i is the cardinality of the i-th mode and R_i is its corresponding rank. The internal node, say, the parent of nodes i and j, is a 3^{rd}-order tensor of size $R_i \times R_j \times R_{ij}$, which is refereed to *transfer* tensor and R_{ij} is one of the so-called hierarchical rank that is defined as no larger than the rank of the tensor. In addition, the root node is a matrix of size $R_{12} \times R_{34}$, as per our example. More details of how to get these ranks and to obtain the H-Tucker decomposition can be found in [Kressner and Tobler, 2012]. Generally speaking, for an N^{th}-order tensor, suppose the cardinally of each mode is I and the rank of each mode in the standard Tucker decomposition is R, the storage cost for Tucker is $NIR + R^N$, which can easily excess the memory limit of a standard computer. In contrast, the storage consumption of H-Tucker is consisted of at most NIR space for loading matrices, $(N-2)R^3$ for transfer tensors, and R^2 space for the root node. As a result, when the tensor order is very high, H-Tucker is more preferable than conventional Tucker and an example application of H-Tucker to higher-order healthcare tensor can be found in [Perros et al., 2015].

The main assumption of H-Tucker is the prior knowledge of a hierarchy of the tensor modes. However, this may not be true for most tensor data. To overcome this issue, Tensor-Train decomposition is an alternative for analyzing tensors with large number of orders. Similar to H-Tucker, Tensor-Train also decomposes a higher-order tensor into a set of matrices and 3^{rd}-order transfer tensors. While in Tensor-Train, a chain structure is used to organize them as a matrix for the first mode, followed by series of 3^{rd}-order transfer tensors of size $R_{i-1} \times I_i \times R_i, i \in (1, N-1)$, followed by a matrix for the last mode. As shown in Figure 2.4f, for a 4^{th}-order tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K \times L}$, its (i, j, k, l)-th element can be approximated as

$$\mathcal{X}(i,j,k,l) \approx \sum_{r_1,r_2,r_3} \mathbf{G}_1(i,r_1)\mathcal{G}_2(r_1,j,r_2)\mathcal{G}_3(r_2,k,r_3)\mathbf{G}_4(r_3,l),$$

where $\mathbf{G}_1 \in \mathbf{R}^{I \times R_1}$, $\mathcal{G}_2 \in \mathbf{R}^{R_1 \times J \times R_2}$, $\mathcal{G}_3 \in \mathbf{R}^{R_2 \times K \times R_3}$ and $\mathbf{G}_4 \in \mathbf{R}^{R_3 \times L}$.

So far we only consider using tensor decomposition to analysis one data source. However, in many real-world applications, it is quite often that multiple data sources are available and jointly modeling these datasets together would be beneficial. One of the earlier researches on this line is PARAFAC2 [Harshman, 1972], where a collection of matrices (each matrix represents one dataset) that share the same column space but have different number of rows are decomposed as a whole. As demonstrated in Figure 2.4g, L matrices are decomposed simultaneously and the l-th one, $\mathbf{X}^{(l)}, l \in [1, L]$, is factorized as

$$\mathbf{X}^{(L)} \approx \mathbf{U}^{(L)}\mathbf{S}\mathbf{V}^{\top}, \tag{2.5}$$

where $\mathbf{V}$ is the shared latent factors among all datasets. When columns represent features, it means that one can learn a joint feature mapping for all data sources while different instant mappings for each of them. And it is easy to see that PARAFAC2 is also able to learn the shared instance latent factors by transposing the input data.

In terms of applications, [Bro et al., 1999] apply PARAFAC2 to chromatographic data to handle time shifts. Usage of PARAFAC2 in chemometrics for fault detection and diagnosis can be found in [Wise et al., 2001, Kamstrup-Nielsen et al., 2013]. [Chew et al., 2007] use PARAFAC2 for clustering documents across multiple languages such that the same collection of documents have been translated to different languages. Several authors have been applied PARAFAC2 for tagging music [Panagakis and Kotropoulos, 2011] and images [Pantraki and Kotropoulos, 2015]. More recently, [Sun et al., 2016] use PARAFAC2 for tracking users' contextual intent in order to provide personal assistants and [Perros et al., 2017] propose new algorithm for using PARAFAC2 on large-scale sparse data such as time-evolving medical records.

Coupled Matrix and Tensor Factorization (CMTF) [Acar et al., 2011b, Acar et al., 2013] goes further than PARAFAC2 in terms of blending information from multiple datasets. In CMTF, data are not limited in matrix format. One can have multiple tensors and matrices that share on at least one mode. For example, in context-aware recommender systems, the main data source is a 3^{rd}-order rating tensor as $user \times item \times context$, while side information like user and item meta-data can be organized into two matrices as $user \times feature$ and $item \times feature$, which share the same set of objects with the first and second modes in the rating tensor, respectively. Figure 2.4h gives an example of CMTF on a 3^{rd}-order tensor $\mathcal{X}$ and a two dimensional matrix $\mathbf{Y}$ that are sharing a common second mode factors. To decompose them, the following joint optimization problem,

$$\underset{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}}{\arg\min} ||\mathcal{X} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]||^2 + \lambda||\mathbf{Y} - \mathbf{D}\mathbf{B}^\top||^2,$$

need to be solved, where $|| \bullet ||$ is the Frobenius norm, $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ stands for the CP decomposition of $\mathcal{X}$ and λ is a parameter to control the relative importance of two data sources. It should be noted that the individual decomposition algorithms used for the tensor and matrix can be any methods chose by user (*e.g.*, the tensor decomposition part can be replaced by Tucker [Ermiş et al., 2015]), as long as the produced loading matrix for the shared mode is the same across the decompositions of all datasets.

As a relatively new method, CMTF has attracted increasing interests from researchers. Applications of CMTF on link prediction problem can be found in [Şimşekli et al., 2015, Ermiş et al., 2015] where information from different sources are fused such that cold start problem can be well-handled. [Khan et al., 2016] propose a Bayesian version of CMTF, which has been applied on structural toxicongenomics and functional neuroimaging. In order to identify patterns that are related to certain diseases, [Acar et al., 2015] jointly model data collected from multiple biological analytic techniques via CMTF. In [Yılmaz et al., 2011], similar idea as CMTF is proposed for fusing information from multiple tensors and a show case of it on audio restoration problem is presented. On the other hand, several researches have been reported on improving the efficiency and scalability of CMTF via techniques such as MapReduce [Beutel et al., 2014, Papalexakis et al., 2014, Jeon et al., 2016, Choi et al., 2017].

2.3 CP Decomposition

We briefly introduced CP decomposition in §2.2.1, in this section, we discuss more details of it. We start with a more well-known technique, Matrix Factorization (MF), and make the linkage between conventional MF to CP decomposition. After that, we focus on the solving algorithms and then discuss some recent variants and extensions to CP decomposition.

2.3.1 Extending Matrix Factorization to Tensors

In data mining and machine learning, one common assumption is that a simple and compact representation can be found underneath the complex and noisy data [Wang and Zhang, 2013]. In many real world applications, the dataset can be organized in a matrix and one can decompose the data matrix into the product of a series of matrices, which is known as matrix factorization. Many fundamental machine learning algorithms like PCA [Abdi and Williams, 2010], Latent Semantic Analysis (LSA) [Deerwester et al., 1990], Independent Component Analysis (ICA) [Lee, 1998], Non-negative Matrix Factorization (NMF) [Lee and Seung, 2001], *etc.*, can be viewed as instances of the MF framework, differing from each other in terms of the constraints imposed. For example, orthogonal constraint is used in PCA, while non-negative constraint that restricts the sign of the input and factorized matrices is incorporated in NMF. As one of the most popular algorithms, MF has been extensively applied in a wide range of problems such as classification, regression, collaborative filtering, factor analysis, dimensionality reduction and clustering.

Several nice properties contribute to the popularity of MF in data science: 1) it reveals latent structures underneath the data while data sparsity is not a issue [Koren, 2008], 2) there usually is meaningful probabilistic interpretation of the produced latent factors [Mnih and Salakhutdinov, 2008], 3) it is a flexible framework that can easily adapt domain-specific prior knowledge or constraints [Huang et al., 2016], 4) there are many optimization techniques can be used to find a good solution[Lu and Yang, 2015].

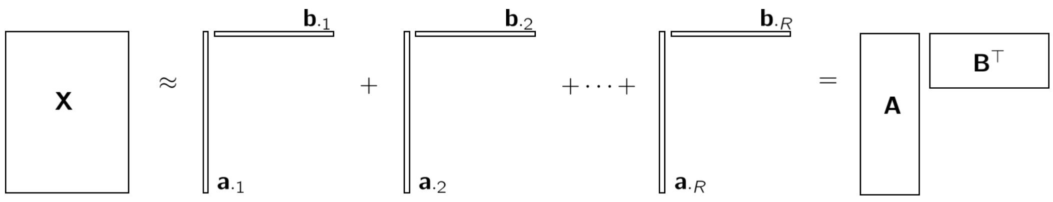


Figure 2.5: Matrix factorization

An illustration of MF can be found in Figure 2.5. Specifically, given a data matrix $\mathbf{X} \in \mathbb{R}^{M \times N}$ with M instances and N features, the basic MF model decomposes it into a product of two matrices as

$$\mathbf{X} \approx \mathbf{A}\mathbf{B}^{\top}, \tag{2.6}$$

where $\mathbf{A} \in \mathbb{R}^{M \times R}, \mathbf{B} \in \mathbb{R}^{N \times R}$ and R is the decomposition rank. For dimensionality reduction tools such as PCA, R is usually much smaller than both M and N to cure the curse of dimensionality. While in sparse coding and dictionary learning [Olshausen and Field, 1997, Lee et al., 2007, Mairal et al., 2010], R is chosen to be large for an over-complete basis.

Alternatively, Eq. (2.6) can be rewritten as a sum of R rank-one matrices as:

$$\mathbf{X} \approx \sum_{r=1}^{R} \mathbf{a}_{\cdot r} \circ \mathbf{b}_{\cdot r}. \tag{2.7}$$

Recall that a 3rd-order tensor $\mathcal{X}$ can be decomposed by the CP decomposition in a similar form as Eq. (2.7) by adding one more loading matrix for the extra mode as

$$\mathcal{X} \approx \sum_{r=1}^{R} \mathbf{a}_{\cdot r} \circ \mathbf{b}_{\cdot r} \circ \mathbf{c}_{\cdot r},$$

one can easily see that CP decomposition is the direct extension of MF to multi-dimensional data. Similarly, we can also express CP decomposition into the similar format as Eq. (2.6) by

$$\begin{aligned}\mathbf{X}_{(1)} &\approx \mathbf{A}(\mathbf{C} \odot \mathbf{B})^\top, \\ \mathbf{X}_{(2)} &\approx \mathbf{B}(\mathbf{C} \odot \mathbf{A})^\top, \\ \mathbf{X}_{(3)} &\approx \mathbf{C}(\mathbf{B} \odot \mathbf{A})^\top.\end{aligned} \tag{2.8}$$

So far we only introduced the formulation of CP decomposition for 3^{rd}-order tensors while above formulation is readily extendable to higher-order tensors. Generally speaking, given an N^{th}-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, its CP decomposition is denoted by $[\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$, which is consisted of N loading matrices. Each loading matrix, $\mathbf{A}^{(n)}, n \in [1, N]$, is of size $I_n \times R$, where R is the so-called CP rank of $\mathcal{X}$, indicating the number of latent factors used to approximate this tensor. By using these loading matrices, the mode-n unfolding of $\mathcal{X}$ can be approximated as

$$\begin{aligned}\mathbf{X}_{(n)} &\approx \mathbf{A}^{(n)}(\mathbf{A}^{(N)} \odot \cdots \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \cdots \odot \mathbf{A}^{(1)})^\top \\ &= \mathbf{A}^{(n)}(\bigodot_{i \neq n}^{N} \mathbf{A}^{(i)})^\top.\end{aligned} \tag{2.9}$$

It is clear that when $\mathcal{X}$ is a matrix ($N = 2$), above formulation for CP decomposition reduces to the MF model in Eq. (2.6).

Because of the close relationship between MF and CP, researchers extensively applied CP decompositions in the similar problem setting as MF but on multi-dimensional data (examples can be found in §2.2.1). More importantly, the properties popularizing MF are applicable to CP decomposition as well. For example, varies solving algorithms exist for finding the CP decomposition of a tensor (discussed in §2.3.2) and it is also a highly flexible framework with several extensions can be found in latest researches (shown in §2.3.3). Additionally, a specific and beneficial property of CP decomposition is its *uniqueness*. That is, there is often only one unique solution that equalizes the input tensor and its CP decomposition with the minimal rank needed (see [Kruskal, 1977, Coppi and Bolasco, 1988, Sidiropoulos and Bro, 2000, De Lathauwer, 2008] for more details on the uniqueness of CP decomposition). While for MF, such property does not exist.

2.3.2 Algorithms for CP Decomposition

Typically CP decomposition tries to find the latent structure underneath the input tensor by a set of low rank matrices. It means that we need to solve an optimization problem that minimizes the error between the tensor and its decomposition. Specifically, given an N^{th}-order tensor $\mathcal{X}$ and its CP decomposition, $[\![\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)}]\!]$, the most commonly used optimization formulation is based on the squared Euclidean distance as

$$\begin{aligned} &\min_{\hat{\mathcal{X}}} \frac{1}{2} \left\| \mathcal{X} - \hat{\mathcal{X}} \right\|^2, \\ &\text{subject to } \hat{\mathcal{X}} = \sum_{r=1}^{R} \mathbf{a}_{\cdot r}^{(1)} \circ \mathbf{a}_{\cdot r}^{(2)} \circ \cdots \circ \mathbf{a}_{\cdot r}^{(N)} = [\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)} \dots, \mathbf{A}^{(N)}]\!]. \end{aligned} \tag{2.10}$$

However, it is very difficult to solve above optimization problem jointly over all loading matrices, since it is not convex w.r.t. $\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)}$. Here we introduce some of the most popular algorithms for solving this problem. For ease of readability, we first derive and demonstrate most of algorithms on the 3^{rd}-order case with decomposition as $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, then move on to their extension of higher-order tensors.

Alternating Least Squares

ALS [Carroll and Chang, 1970, Harshman, 1970] is the most well-known, and in most of the time, the default or work-horse algorithm for CP decomposition. The main idea of ALS is simple. Even though Eq. (2.10) is non-convex, by fixing all other loading matrices while just updating one at each time, the problem can reduce to a linear least-squares problem, which is convex and easy to solve. As a result, each time ALS updates only one loading matrix by fixing all others, then move on to another loading matrix. Such update schema is repeated several iterations until converged, as described in Algorithm 2.1.

To be more specific, for the 3^{rd}-order case, $\mathbf{A}$ is updated at first by fixing $\mathbf{B}$ and $\mathbf{C}$. According to Eq. (2.8), Eq. (2.10) can be rewrite into a simple

Algorithm 2.1: Alternating Least Squares for CP Decomposition

```
Input: input tensor $\mathcal{X}$, decomposition rank $R$
Output: CP decomposition $[\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)} \dots, \mathbf{A}^{(N)}]\!]$
1 Initialize $\mathbf{A}^{(1)}, \mathbf{A}^{(2)} \dots, \mathbf{A}^{(N)}$
2 while not converged do
3     for $n \leftarrow 1$ to $N$ do
4         $\mathbf{A}^{(n)} \leftarrow \frac{\mathbf{X}_{(n)}(\bigodot_{i \neq n}^{N} \mathbf{A}^{(i)})}{\circledast_{i \neq n}^{N} (\mathbf{A}^{(i)\top} \mathbf{A}^{(i)})}$
5     end
6 end
```

minimization problem in matrix form as

$$\min_{\mathbf{A}} \frac{1}{2} \left\| \mathbf{X}_{(1)} - \mathbf{A}(\mathbf{C} \odot \mathbf{B})^\top \right\|^2. \tag{2.11}$$

By taking the derivative w.r.t. $\mathbf{A}$ and setting it to zero, one can easily obtain the optimal solution for this as

$$\mathbf{A} \leftarrow \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})[(\mathbf{C} \odot \mathbf{B})^\top (\mathbf{C} \odot \mathbf{B})]^\dagger.$$

From Eq. (2.1), above equation can be simplified as

$$\mathbf{A} \leftarrow \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})(\mathbf{C}^\top \mathbf{C} \circledast \mathbf{B}^\top \mathbf{B})^\dagger.$$

This reduces the computational complexity since $\mathbf{C} \odot \mathbf{B}$ is a very tall matrix of size $JK \times R$, where J and K are the number of rows in $\mathbf{B}$ and $\mathbf{C}$, respectively.

$\mathbf{B}$ and $\mathbf{C}$ can be optimized in a similar way and generally speaking, for an N^{th}-order tensor, the update rule for its n-th loading matrix can be expressed as

$$\mathbf{A}^{(n)} \leftarrow \frac{\mathbf{X}_{(n)}(\bigodot_{i \neq n}^{N} \mathbf{A}^{(i)})}{\circledast_{i \neq n}^{N} (\mathbf{A}^{(i)\top} \mathbf{A}^{(i)})}. \tag{2.12}$$

ALS has gained great popularity due to its simplicity and ease of implementation. However, there are several shortcomings of it should be noticed. First, ALS assumes that the decomposition rank is known in advance while it is a NP-hard problem to determine the best rank for a given tensor [Håstad, 1990]. As a result, in practice it is quite common to try ALS with different ranks until finding one with good fit. Second, ALS usually takes multiple iterations to converge and there is no guarantee on the global optimality of solution. Meanwhile, the quality of the final solution can be quite sensitive to the initial guess so that multiple repetitions with different initial seeds may be required for applying ALS.

The major criticize of ALS is its efficiency. The main computational bottleneck is the calculation of $\mathbf{X}_{(n)}(\bigodot_{i\neq n}^{N}\mathbf{A}^{(i)})$, which is often referred to as *matricized tensor times Khatri-Rao products* (MTTKRP) [Smith et al., 2015]. Specifically, the first term, $\mathbf{X}_{(n)}$, is a $I_n \times \prod_{i\neq n}^{N} I_i$ flat matrix and the Khatri-Rao product produces a super tall matrix with size $\prod_{i\neq n}^{N} I_i \times R$, resulting in an overall time complexity as $\mathcal{O}(R\prod_{i=1}^{N} I_i)$, which is linear to the size of $\mathcal{X}$. In addition, in terms of space usage, ALS is challenged by the so-called *intermediate data explosion* problem [Kang et al., 2012], since it requires allocating a huge amount of temporal memory space to store the matricized tensor and the Khatri-Rao products. Overall, both the time and space complexities of naive MTTKRP calculation are linear to the size of the input tensor, which dominants the high computational cost of ALS.

A number of researches can be found in the literature to overcome the efficiency issue imposed by the MTTKRP procedure and to improve the performance of ALS. One popular direction is to take advantage of the increasingly accessible computational resources by using techniques such as distributed and parallel computing [Kang et al., 2012,Smith et al., 2015,Jeon et al., 2015, Li et al., 2017], and GPU acceleration [Liu et al., 2017]. Meanwhile, another way to accelerate MTTKRP is to design new algorithms. For example, [Phan et al., 2013a] propose a faster algorithm by using tensor reshaping and taking the ranking of cardinality of different modes into consideration, resulting a significant speed-up especially for higher-order tensors, compared to the conventional approach. At the same time, a particularly interesting case of calculating MTTKRP is sparse tensor, which is

Algorithm 2.2: MTTKRP via Sparse Tensor-Vector Products

```
Input: non-zeros $\mathbf{z}$, indices of non-zeros $\mathbf{j}^{(1)}, \ldots, \mathbf{j}^{(N)}$, loading
       matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$, mode $n$
Output: mode-$n$ MTTKRP $\mathbf{P}^{(n)}$
1 for $r \leftarrow 1$ to $R$ do
2     $\mathbf{t} \leftarrow \mathbf{z}$
3     for $i \leftarrow 1$ to $N$ do
4         $\mathbf{t} \leftarrow \mathbf{t} \circledast \mathbf{a}^{(i)}_{\mathbf{j}^{(i)}r}$
5     end
6     $\mathbf{p}^{(n)}_{\cdot r} \leftarrow$ ACCUMARRAY$(\mathbf{j}^{(n)}, \mathbf{t})$
7 end
```

often large-scale. Compared to its size, the amount of valuable information, i.e., the number of non-zeros, is much reduced. As a result, it is not desirable or feasible to explicitly calculate MTTKRP for sparse tensors. The most popular method that has been widely recognized as a gold standard is [Bader and Kolda, 2007] (summarized in Algorithm 2.2), which dramatically reduces the complexity of MTTKRP to be linear in the number of non-zeros in a sparse tensor.

In addition to accelerate ALS via utilizing the MTTKRP procedure, several works have been proposed to further improve ALS. To cope with the slow convergence issue of ALS, line search [Bro, 1998, Rajih et al., 2008, Nion and De Lathauwer, 2008] can be applied to the ALS procedure. At the end of each iteration, a line search step it performed to find the direction where the objective function will be reduced and then all loading matrices will be forced to be updated along this direction. To speed up ALS for large-scale tensors, [Kiers and Mechelen, 2001] reduce the scale of the optimization problem in Eq. (2.11) by data compression. Specifically, the input data is compressed and then decomposed by typical ALS. After that, the produced loading matrices are recovered to the original space. Similarly, [Cheng et al., 2016, Reynolds et al., 2016, Battaglino et al., 2018] replace the least square problem with a randomized version where sampling technique is used to limit the problem scale for a better efficiency and convergence. ParCube [Papalexakis et al., 2012] and SamBaTen [Gujral et al., 2018a] are also sampling-based approaches where multiple small sub-tensors are sampled from the original data and decomposed by ALS concurrently, then the produced latent factors are merged together to construct the final result.

Other Algorithms for Computing CP Decomposition

Apart from ALS, there are other algorithms can be applied to find the CP decomposition of a tensor. Here we introduce some of the most popular ones.

Broadly speaking, ALS belongs to the Block Coordinate Decent (BCD) family for non-linear optimization [Xu and Yin, 2013, Kim et al., 2014], which is a divide-and-conquer algorithm that splits parameter set into several disjoint small subsets and iteratively optimizes one subset at a time so that the overall objective is minimized.

Consider the following optimization problem

$$\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}_1,\ldots,\mathbf{x}_M), \tag{2.13}$$

where $\mathbf{x} = (\mathbf{x}_1,\ldots,\mathbf{x}_M) \in \mathbb{R}^N$ is a variable that can be decomposed into M blocks, the set $\mathcal{X}$ of feasible points is assumed to be a closed convex subset of $\mathbb{R}^N$, $\mathbf{x}_m, m = 1,\ldots,M$, is a closed convex subset of $\mathbb{R}^{N_m}$ satisfying $N = \sum_{m=1}^{M} N_m$, f is assumed to be a differentiable and convex function w.r.t. each $\mathbf{x}_m$ when all other blocks are fixed, while can be non-convex w.r.t. $\mathbf{x}$.

BCD minimizes f cyclically over each block in $\mathbf{x}_1,\ldots,\mathbf{x}_M$ by fixing the remaining blocks at their latest updated values. That is, if $\mathbf{x}^{(i)} = (\mathbf{x}_1^{(i)},\ldots,\mathbf{x}_M^{(i)})$ is the current value at the i-th iteration, BCD updates $\mathbf{x}^{(i)}$ block by block to produce $\mathbf{x}^{(i+1)}$ by solving the following sub-problem as

$$\mathbf{x}_m^{(i+1)} \leftarrow \arg\min_{\mathbf{x}_m\in\mathcal{X}_m^{(i)}} f(\mathbf{x}_1^{(i+1)},\ldots,\mathbf{x}_{m-1}^{(i+1)},\mathbf{x}_m,\mathbf{x}_{m+1}^{(i)},\ldots\mathbf{x}_M^{(i)})$$

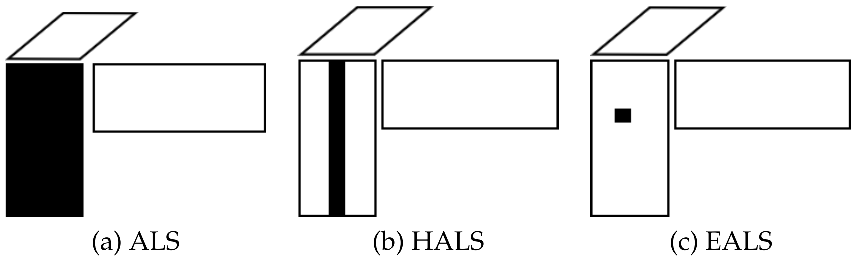


Figure 2.6: CP decomposition algorithms under the BCD Frameworks

In the case of CP decomposition, it is easy to see that ALS is a special case of BCD with $R \times \prod_{n=1}^{N} I_n$ parameters need to be estimated and they are divided into N blocks for an N^{th}-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$. Under the BCD framework, a natural question to ask is whether we can partition the parameters with different schemes other than the matrix by matrix strategy used in ALS. In fact, it turns out that there are a few partition methods can be used, which are more beneficial than ALS. We give a high-level illustration of these algorithms and a comparison to ALS in Figure 2.6.

One example of BCD for CP decomposition is Hierarchical Alternating Least Squares (HALS) [Cichocki and Anh-Huy, 2009] that updates the loading matrices column by column. Let $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ be the input tensor, $\hat{\mathcal{X}} = [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$ be its CP decomposition, $\mathbf{K}^{(n)} = \bigodot_{i \neq n}^{N} \mathbf{A}^{(i)}$ be the Khatri-Rao product of all loading matrices but the n-th one, R be the decomposition rank, we have

$$\begin{aligned} f(\mathbf{a}_{\cdot 1}^{(1)}, \ldots, \mathbf{a}_{\cdot R}^{(1)}, \ldots, \mathbf{a}_{\cdot 1}^{(N)}, \ldots, \mathbf{a}_{\cdot R}^{(N)}) &= \frac{1}{2} \left\| \mathcal{X} - [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!] \right\|^2 \\ &= \frac{1}{2} \left\| \mathbf{X}_{(n)} - \mathbf{A}^{(n)} \mathbf{K}^{(n)^\top} \right\|^2 \\ &= \frac{1}{2} \left\| \mathbf{X}_{(n)} - \sum_{r=1}^{R} \mathbf{a}_{\cdot r}^{(n)} \mathbf{k}_{\cdot r}^{(n)^\top} \right\|^2 \end{aligned} \tag{2.14}$$

By defining the residue as

$$\mathbf{X}_{(n)}^{r} = \mathbf{X}_{(n)} - \sum_{\tilde{r} \neq r}^{R} \mathbf{a}_{\cdot r}^{(n)} \mathbf{k}_{\cdot r}^{(n)^\top} = \mathbf{X}_{(n)} - \mathbf{A}^{(n)} \mathbf{K}^{(n)^\top} + \mathbf{a}_{\cdot r}^{(n)} \mathbf{k}_{\cdot r}^{(n)^\top}, \tag{2.15}$$

we can rewrite Eq. (2.14) into an alternative minimization problem as

$$\min_{\mathbf{a}_{\cdot r}^{(n)}} \frac{1}{2} \left\| \mathbf{X}_{(n)}^{r} - \mathbf{a}_{\cdot r}^{(n)} \mathbf{k}_{\cdot r}^{(n)\top} \right\|^2,$$

whose optimal solution can be easily found by normal equation as

$$\mathbf{a}_{\cdot r}^{(n)} \leftarrow \frac{\mathbf{X}_{(n)}^{r} \mathbf{k}_{\cdot r}^{(n)}}{\mathbf{k}_{\cdot r}^{(n)\top} \mathbf{k}_{\cdot r}^{(n)}}$$

Compared to ALS, the update rule of HALS is more efficient since the time-consuming matrix inversion operation is avoided as $\mathbf{k}_{\cdot r}^{(n)\top} \mathbf{k}_{\cdot r}^{(n)}$ is the vector dot product that produces a scalar. More importantly, HALS is a more flexible algorithm to incorporate constraints such as non-negativity. For ALS, special solver and algorithms is required to solve the non-negative least squares problem. However, by using the column-wise update rule in HALS, non-negativity can be simply enforced by replacing negative values with zero, of which the correctness is proved in [Kim et al., 2014].

In addition to HALS, another algorithm under the BCD framework for CP decomposition is Element-wise Alternating Least Squares (EALS) [He et al., 2016]. Different from ALS and HALS, EALS updates the parameter set in a finer grain that only one element in a loading matrix is updated by fixing all others. Specifically, each element in a tensor can be approximated by the loading matrices as

$$\begin{aligned} \hat{x}_{i_1,\ldots,i_N} &= \sum_{r=1}^{R} \prod_{n=1}^{N} a_{i_n r}^{(n)} = \sum_{r=1}^{R} a_{i_n r}^{(n)} k_{j_n r}^{(n)} \\ &= \sum_{\tilde{r} \neq r}^{R} a_{i_n \tilde{r}}^{(n)} k_{j_n \tilde{r}}^{(n)} + a_{i_n r}^{(n)} k_{j_n r}^{(n)} = \hat{x}_{i_n j_n}^{r} + a_{i_n r}^{(n)} k_{j_n r}^{(n)}, \end{aligned} \tag{2.16}$$

where $j_n = (i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N)$, $k_{j_n r}^{(n)} = \prod_{\tilde{n} \neq n}^{N} a_{i_{\tilde{n}} r}^{(\tilde{n})}$ and $\hat{x}_{i_n j_n}^{r}$ is the residue defined in a similar way as Eq. (2.15). The objective that EALS aims to minimize is

$$f(a_{11}^{(1)}, \dots, a_{I_1R}^{(1)}, \dots, a_{11}^{(N)}, \dots, a_{I_NR}^{(N)}) = \frac{1}{2}\sum_{i_nj_n}(x_{i_nj_n} - \hat{x}_{i_n,j_n})^2$$
$$= \frac{1}{2}\sum_{i_nj_n}(x_{i_nj_n} - \hat{x}_{i_nj_n}^r - a_{i_nr}^{(n)}k_{j_nr}^{(n)})^2.$$

Similar to HALS, by normal equation we have

$$a_{i_nr}^{(n)} \leftarrow \frac{\sum(x_{i_nj_n} - \hat{x}_{i_nj_n}^r)k_{j_nr}^{(n)}}{\sum(k_{j_nr}^{(n)})^2}.$$

EALS is more efficient than ALS and HALS as it further reduces the time complexity to linear to the decomposition rank, R. On top of its advantages in efficiency, this algorithm is often used for sparse tensor decompositions since it approximates the tensor in an element-wise fashion so the objective can be limited to the non-zero entries in the tensor. We show an example of using this algorithm for sparse tensors later in Chapter 5.

Even though ALS remains as the most popular algorithms for CP decomposition and a significant amount of works are proposed to improve its performance, there still are a considerable number of algorithms can be found in the literature that take a different path other than alternating minimization. One group of such algorithms are doing all-at-once optimization that update all loading matrices simultaneously via non-linear least square solvers such as gradient-based algorithm [Acar et al., 2011a] and damped Gauss-Newton algorithms [Tomasi and Bro, 2005,Phan et al., 2013b]. Compared to ALS, all-at-once algorithms usually show better performance in terms of convergence, but at a cost of a much higher computational load per iteration [Cichocki et al., 2015]. On the other hand, in recent years, an increasingly popular type of algorithms for CP decomposition are probabilistic models, which are motivated by the success of probabilistic matrix factorization [Mnih and Salakhutdinov, 2008]. Probabilistic tensor decomposition algorithms treat values in loading matrices as random variables and approximate the joint probability distribution across these variables based on the observed data. Typical methods used for parameter estimation include Markov Chain Monte Carlo (MCMC) [Xiong et al., 2010], Gibbs sampling [Rai et al., 2014] and variational inference [Zhao et al., 2015, Zhao et al., 2016].

2.3.3 Extensions of CP Decomposition

Since CP decomposition is one of the most popular algorithm for analyzing tensors, researchers have proposed a wide variety of extensions to the CP model, according to different real-world requirements.

The most common extension to CP decomposition is imposing constraints, which is usually known as Constrained CP (CCP) decomposition. Non-negativity, which enforces both the input tensor and loading matrices to be non-negative, is one example of CCP. The usage of non-negative constraint has a long history in MF [Lee and Seung, 2001], which shows that such non-negativity can lead to a part-based model where the input data is an additive combination of primitive patterns presented in the data. Due to the close relationship between MF and CP decomposition, it is natural to incorporate non-negativity to seek a more interpretable model. Early approaches adapt the multiplicative update rule in [Lee and Seung, 2001] to tensors [Welling and Weber, 2001, FitzGerald et al., 2005, Bader et al., 2008] or use specially designed non-negative least square algorithms [Bro and De Jong, 1997, Friedlander and Hatz, 2008]. While HALS algorithm proposed by [Cichocki and Anh-Huy, 2009] is more efficient, simple to implement, and shows better convergence. Another type of constraint is sparseness on the loading matrices of the CP model. Having sparsity on the latent factors also benefits the interpretability of the model, as one instance will only be reconstructed by a few important factors while ignoring the noise. Usually sparseness is imposed by regularizing the conventional CP model with a ℓ_1-norm penalty term, which can be easily solved by the ALS-like algorithms. Besides this, [Papalexakis et al., 2012] shows that their sampling-based algorithm can naturally produce sparse result without any explicitly formulated sparsity constraint. Similar to penalize the CP decomposition with ℓ_1-norm, Frobenius norm on the loading matrices is also a commonly used regularization. This helps to restrict the range of the values in the loading matrices, which in turn, to prevent overfitting.

Another extension to CP decomposition is to take input data type into account. One example is Boolean CP decomposition [Miettinen, 2011, Erdős and Miettinen, 2013] where the data tensor is consisted of binary values and the CP model is expressed with Boolean operation (*i.e.* $1 + 1 = 1$). Compared to normal CP decomposition, Boolean CP is more desirable for binary data since it respects the nature of the data and will produce more interpretable results. Similarly, for sparse count tensor where data are non-negative integers, for example, a *user* $\times$ *user* $\times$ *day* tensor that indicates how many times user i called user j on day t, conventional CP decomposition that minimizes the Frobenius norm may not be appropriate since it assumes the data are normally distributed. Special consideration need to be given to this factor. As demonstrated by [Chi and Kolda, 2012], Poisson distribution is a more suitable assumption and KL-Divergence should be used for count data.

Finally, one increasing research trend in tensor decomposition is to extend CP decomposition to tensors that are dynamically changing over time. However, directly applying static algorithms (e.g. aforementioned ALS, HALS algorithms) to such dynamic tensors is time-consuming and a waste of computational resources. As a result, new algorithms that can adapt the current decomposition in real-time is a must for dynamic tensors. In fact, this thesis mainly focuses on this direction of designing effective, efficient and scalable algorithms for dynamic tensors. We provide a detailed discussion on dynamic tensor decomposition in the following section.

2.4 Dynamic Tensor Decomposition

This section gives the background of the main research problem we focus in this thesis: dynamic tensor decomposition. We start with an introduction to dynamic tensors and two typical dynamic updates. After that, we show some representative researches that have been proposed, including detailed discussions on baseline algorithms used in our experiment (the only available methods by the time of publishing) and a brief review of the more recent and advanced approaches.

2.4.1 Dynamic Tensors

A lot of real-world tensors are not static. In fact, a data tensor is often dynamically changing over time, and a large volume of new data can be generated at high velocity. In such dynamic environments, a data tensor may be expanded, shrunk or modified on any of its dimensions. For example, given a network monitoring tensor as $source \times destination \times port \times time$, a large number of network transactions are generated every second, which can be recorded by appending new slices to the tensor on its time mode. Additionally, new IP addresses may be added and invalid addresses may be removed from the data tensor. Another example of non-static tensor can be found in context-aware recommender systems where the rating data is represented by a tensor as $user \times item \times context$. A large amount of new ratings, (user, item, context) tuples in this example, will be generated and appended to the system at a high speed. Overall, real-world tensors can be high dynamic and we refer to such tensors as *dynamic tensors*.

In this thesis, we specifically focus on two types of dynamic tensors, since they are the most commonly seen tensors that occur in practice. The first is dynamic tensors with *slice-wise* updates, which we refer to as online tensors, also known as tensor streams and incremental tensors [Sun et al., 2006, Sun et al., 2008]. An online tensor is a tensor that are incrementally growing on its time mode, while the other dimensions remain unchanged. Example of online tensors includes time-evolving networks, fMRI data and movies where new snapshots, new scans and new frames, respectively, are kept appending on the existing data along the time. The second type of dynamic tensors we are interested in are tensors with *element-wise* updates. The motivation of this mainly comes from recommender system problems where new ratings can be appended to existing tensor at arbitrary positions. In such setting, the static model learned from historical data can quickly become outdated and an efficient algorithm that meet the real-time requirement with good accuracy is necessary.

A few key differences between the slice-wise and element-wise updates on dynamic tensors should be highlighted here. First, in practice, slice-wise updates can happen on both dense and sparse tensors while element-wise updates are usually found in sparse tensors only. Since sparse tensor usually means large-scale data, when processing dynamic tensors, data sparsity is a factor that need to be taken into consideration in terms of efficiency and scalability performance. Second, the data generation speed of a dynamic tensor with element-wise updates is often much higher than that of online tensors, resulting a higher requirement for real-time processing. Third, compared to slice-wise updates, element-wise dynamic updates contains a relatively smaller amount of new data. As a result, its contribution to the new decomposition should be less significant than the slice-wise update.

It should be noted the dynamic update happened on a tensor can be more complex than aforementioned two cases. For example, sometimes one may observe expansion on the other modes of online tensor (e.g., new node added to a time-evolving network) or even modifications on ratings in a recommender system. However, compared to the slice-wise and element-wise updates discussed above, those special cases are less common to see. In addition, we only focus on learning the CP decomposition of dynamic tensors, since it is one of the most commonly used tensor decomposition algorithm with various applications. As a result, in this thesis, we limit the scope of our research on find the CP decomposition of dynamic tensors with slice-wise and element-wise updates only, while leaving the other cases as future work.

2.4.2 Algorithms for Slice-wise Dynamic Updates

Earlier researches of decomposing dynamic tensors with slice-wise updates are mainly target on the Tucker decomposition [Sun et al., 2006, Sun et al., 2008, Ma et al., 2009, Hu et al., 2011, Liu et al., 2012b]. However, there is limited works can be found for CP decompositions until recent years. Here we describe the main idea of existing algorithms for incrementally decomposing an online tensors under the CP model. For ease of readability, a 3^{rd}-order online tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times (t_{old}+t_{new})}$ is used as a running example, where $\mathcal{X}$ is expanded from $\mathcal{X}_{old} \in \mathbb{R}^{I \times J \times t_{old}}$ by appending a new chunk of data $\mathcal{X}_{new} \in \mathbb{R}^{I \times J \times t_{new}}$ at its last mode. Considering that in most online systems, the size of the new incoming data is usually much smaller than that of all existing historical data, thus we assume $t_{new} \ll t_{old}$. The CP decomposition of $\mathcal{X}_{old}$ is written as $[\![\mathbf{A}_{old}, \mathbf{B}_{old}, \mathbf{C}_{old}]\!]$ and the aim is to find the CP decomposition $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of $\mathcal{X}$.

SDT and RLST [Nion and Sidiropoulos, 2009]: Simultaneous Diagonalization Tracking (SDT) and Recursive Least Squares Tracking (RLST) are the two earliest and most cited algorithms proposed for finding the CP decomposition of a dynamic tensor with slice-wise updates. Both SDT and RLST transform the online tensor decomposition problem into an incremental matrix factorization problem. Recall that $\mathbf{X}_{(3)} = \mathbf{C}(\mathbf{B} \odot \mathbf{A})^\top$ and let $\mathbf{D} = \mathbf{B} \odot \mathbf{A}$, the tensor decomposition can be rewritten into matrix format as $\mathbf{X}_{(3)} = \mathbf{C}\mathbf{D}^\top$. Then the problem is how to estimate $\mathbf{C}$ and $\mathbf{D}$, and then how to estimate $\mathbf{A}$ and $\mathbf{B}$ from $\mathbf{D}$.

Different strategies are used in SDT and RLST for calculating $\mathbf{C}$ and $\mathbf{D}$. SDT chooses to do this by making use of the SVD of $\mathbf{X}_{old(3)}$, $\mathbf{U}_{old}\mathbf{\Sigma}_{old}\mathbf{V}_{old}^\top$. Specifically, there will always be a matrix $\mathbf{W}_{old}$ that $\mathbf{C}_{old} = \mathbf{U}_{old}\mathbf{W}_{old}^{-1}$ and $\mathbf{D}_{old} = \mathbf{V}_{old}\mathbf{\Sigma}_{old}\mathbf{W}_{old}^\top$. Similarly, a matrix $\mathbf{W}$ can be found to make $\mathbf{C} = \mathbf{U}\mathbf{W}^{-1}$ and $\mathbf{D} = \mathbf{V}\mathbf{\Sigma}\mathbf{W}^\top$, where $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ is the SVD of $\mathbf{X}_{(3)}$, which can be efficiently calculated by incremental SVD algorithms. Furthermore, the authors assume that there is only a tiny difference between $\mathbf{D}$ and $\mathbf{D}_{old}$ so that the first t_{old} rows of $\mathbf{C}$ are approximately equal to $\mathbf{C}_{old}$. Under this assumption, $\mathbf{W}^{-1}$ can be calculated as $\widetilde{\mathbf{U}}^\dagger\mathbf{U}_{old}\mathbf{W}_{old}^{-1}$, where $\widetilde{\mathbf{U}}$ is the first t_{old} rows of $\mathbf{U}$. Consequently, $\mathbf{W}$, $\mathbf{C}$ and $\mathbf{D}$ can be obtained.

In contrast, RLST follows a more direct approach to get $\mathbf{C}$ and $\mathbf{D}$. Recall that $\mathbf{X}_{(3)} = \mathbf{C}\mathbf{D}^\top$, firstly, $\mathbf{C}_{new}$ is calculated as $\mathbf{X}_{new(3)}(\mathbf{D}_{old}^\top)^\dagger$ and $\mathbf{C}$ is updated by appending $\mathbf{C}_{new}$ to $\mathbf{C}_{old}$. Then $\mathbf{D}$ is incrementally estimated with $\mathbf{X}_{new(3)}$ and $\mathbf{C}_{new}$ based on the matrix inversion and pseudo-inversion lemmas. Due to the page limit, we refer interested readers to the original papers [Nion and Sidiropoulos, 2009] for more details.

After getting $\mathbf{C}$ and $\mathbf{D}$, the last step for both SDT and RLST is to estimate $\mathbf{A}$ and $\mathbf{B}$ from $\mathbf{D}$. This process is done by applying SVD on the matrix formed by each column of $\mathbf{D}$, and then putting the left and right principal singular vectors into $\mathbf{A}$ and $\mathbf{B}$, respectively.

Overall, SDT and RLST both deal with the online CP decomposition problem by flattening the non-temporal modes. However, this is the main limitation to their performance. Firstly, it is time consuming due to the cost of SVD. Although the authors replaced the traditional SVD with the Bi-SVD algorithm, the complexity of this is still $\mathcal{O}(R^2IJ)$, which limits their applications on large-scale tensors. Additionally, this flattening process makes SDT and RLST not easy to extend to higher-order tensors, since the flattened matrix will be much larger and it has to be recursively decomposed to loading matrices in the end, which is also costly.

GridTF [Phan and Cichocki, 2011]: Prior to our work, SDT and RLST are the only algorithms that can incrementally update an online tensor under the CP framework. However, among the studies that focus on improving CP decomposition for handling large-scale tensors, GridTF, a grid-based tensor factorization algorithm is potentially applicable to the online CP decomposition problem. The basic idea of GridTF is to partition the whole tensor into smaller tensors and then combine their CP decompositions together. In regards to the example here, $\mathcal{X}$ is the online tensor, $\mathcal{X}_{old}$ and $\mathcal{X}_{new}$ are the two partitions. To obtain the CP decomposition of $\mathcal{X}$, the first step of GridTF is to decompose $\mathcal{X}_{new}$ by ALS as $[\![\mathbf{A}_{new}, \mathbf{B}_{new}, \mathbf{C}_{new}]\!]$, while the decomposition of $\mathcal{X}_{old}$ is already known from the last time step.

The combination step is a recursive update procedure such that in every iteration, each mode is updated with a modified ALS rule, until the whole estimation converges, or the maximum number of iterations has been reached. In our notation, the update rules are given as follows, of which further details can be found in [Phan and Cichocki, 2011].

1) For non-temporal modes **A** and **B**

$$\mathbf{A} \leftarrow \frac{\mathbf{A}_{old}(\mathbf{P}_{old} \oslash (\mathbf{A}_{old}^{\top}\mathbf{A}))}{\mathbf{Q} \oslash (\mathbf{A}^{\top}\mathbf{A})} + \frac{\mathbf{A}_{new}(\mathbf{P}_{new} \oslash (\mathbf{A}_{new}^{\top}\mathbf{A}))}{\mathbf{Q} \oslash (\mathbf{A}^{\top}\mathbf{A})}$$
$$\mathbf{B} \leftarrow \frac{\mathbf{B}_{old}(\mathbf{P}_{old} \oslash (\mathbf{B}_{old}^{\top}\mathbf{B}))}{\mathbf{Q} \oslash (\mathbf{B}^{\top}\mathbf{B})} + \frac{\mathbf{B}_{new}(\mathbf{P}_{new} \oslash (\mathbf{B}_{new}^{\top}\mathbf{B}))}{\mathbf{Q} \oslash (\mathbf{B}^{\top}\mathbf{B})}$$

2) For temporal mode **C**

$$\mathbf{C} \leftarrow \begin{bmatrix} \dfrac{\mathbf{C}_{old}(\mathbf{P}_{old} \oslash (\mathbf{C}_{old}^{\top}\mathbf{C}))}{\mathbf{Q} \oslash (\mathbf{C}^{\top}\mathbf{C})} \\ \dfrac{\mathbf{C}_{new}(\mathbf{P}_{new} \oslash (\mathbf{C}_{new}^{\top}\mathbf{C}))}{\mathbf{Q} \oslash (\mathbf{C}^{\top}\mathbf{C})} \end{bmatrix}$$

where **A**, **B**, **C** are randomly initialized at the beginning and $\mathbf{P}_{old} = (\mathbf{A}_{old}^{\top}\mathbf{A}) \circledast (\mathbf{B}_{old}^{\top}\mathbf{B}) \circledast (\mathbf{C}_{old}^{\top}\mathbf{C})$, $\mathbf{P}_{new} = (\mathbf{A}_{new}^{\top}\mathbf{A}) \circledast (\mathbf{B}_{new}^{\top}\mathbf{B}) \circledast (\mathbf{C}_{new}^{\top}\mathbf{C})$, and $\mathbf{Q} = (\mathbf{A}^{\top}\mathbf{A}) \circledast (\mathbf{B}^{\top}\mathbf{B}) \circledast (\mathbf{C}^{\top}\mathbf{C})$.

The main issue of applying GridTF to online CP decomposition is its efficiency. Firstly, even though only the new data need to be decomposed, this is still expensive, especially when the size of new data is large. Secondly, for estimating **C** and calculating $\mathbf{P}_{old}$ and **Q**, $\mathbf{C}_{old}^{\top}\mathbf{C}$ needs to be calculated, costing $R^2 t_{old}$ operations. This means the time complexity of the update procedure is linear in the length of the existing data, t_{old}, which can be huge, thus significantly limiting its ability for processing online tensors.

Others: Apart from aforementioned algorithms, there are some new approaches proposed in recent years for online CP decomposition. SamBaTen [Gujral et al., 2018a] is a sampling-based approach that is extended from ParCube [Papalexakis et al., 2012]. Basically, it tries to reduce the scale of the problem by sub-sampling a few sub-tensors, which are simultaneously decomposed by standard ALS algorithm and combined to produce the final results. Compared to the static ParCube algorithm that is sampling directly from the input data, SamBaTen replaces the historical data part with the existing decomposition to achieve memory-save and speedup. However, multiple repetitions are required for a stable result and the efficiency of SamBaTen is still depending on its core ALS procedure. Similar approach that merging results from multiple sub-tensors is employed in OCTen [Gujral et al., 2018b], while the sub-sampling procedure in SamBaTen is replaced with random projection for data compression. CP-stream [Smith et al., 2018] is another recent work on dynamic tensors with slice-wise updates. In this work, the importance of historical data is discounted with the growth of time and constraints such as non-negativity is imposed and solved by Alternating Direction Method of Multipliers (ADMM). Furthermore, there exist a few works that solved slightly different but related problems to online CP decomposition. One example is [Song et al., 2017] where the expansion of the dynamic tensors can be found in multiple dimensions at the same time. An ALS-like algorithm is proposed and the major speedup is gained by replacing historical data with existing decomposition. An Bayesian approach for the same multi-aspect dynamics on tensors can be found in [Du et al., 2018]. [Xu et al., 2016] apply incremental CP decomposition in a supervised setting on spatial-temporal data that are growing on its time mode, or multiple dimensions at one time [Xu et al., 2018].

2.4.3 Algorithms for Element-wise Dynamic Updates

Compared to the increasing attention has been given to dynamic tensors with slice-wise updates, studying of how to handle the finer-grain element-wise updates is less active research area. To the best of our knowledge, the only work that study the tensors with element-wise updates is [Shin et al., 2017], while the focus of this work is to detect dense sub-tensors in a dynamic tensor, not to decompose it for learning latent factors. The most related works on element-wise dynamic updates can be found in the literature are for MF [Devooght et al., 2015,He et al., 2016]. The assumption is that the new data will only has considerable impact to its local features, while the global model will not be significantly affected. That is, given a matrix $\mathbf{X} \in \mathbb{R}^{M \times N}$ and its decomposition $\mathbf{W} \in \mathbb{R}^{M \times R}$ and $\mathbf{H} \in \mathbb{R}^{N \times R}$, a new element appended to the data, x_{mn} can be learned as just updating only the m-th and n-th rows of $\mathbf{W}$ and $\mathbf{H}$, while the remaining values in the loading matrix are kept unchanged. Both [Devooght et al., 2015] and [He et al., 2016] choose to learn element-wise update in such schema, while Stochastic Gradient Decent (SGD) is used in the former, and EALS (similar to what presented in §2.3.2) is applied in the latter.

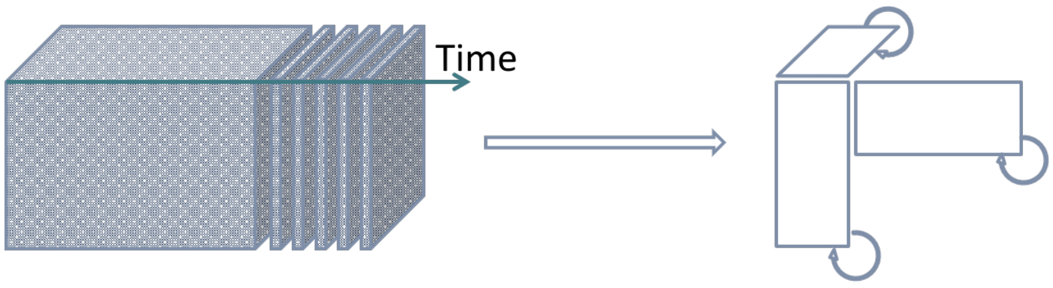
Inspired by the success of this idea in MF, it is natural to consider employing the same strategy that only updates the corresponding rows in the loading matrices when element-wise updates are observed, in dynamic tensors. One approach that is able to perform row-wise update on loading matrices is iTALS [Hidasi and Tikk, 2012], which is a static CP decomposition algorithm proposed for sparse tensors with implicit feedbacks. However, since it employs a row-wise ALS algorithm, one can easily adapt iTALS to the element-wise dynamic learning on tensors. The details of iTALS can be found in the original paper and we give a brief introduction of this algorithm in §5.3. In addition, we also choose to use the same local update strategy and proposed a new algorithm based on EALS that is more flexible and efficient than iTALS, which is presented in Chapter 5.

2.5 Summary

In this chapter, we introduced the main concepts, algorithms and applications of tensor decomposition technologies, which built a solid foundation to better understanding the following chapters. In addition, we provided a detailed explanation of CP decomposition and reviewed related literature. Furthermore, specially emphasis was given to dynamic tensors and a summary of existing algorithms of how to track the CP decomposition of dynamic tensors was presented.

Chapter 3

Dense Tensor Decomposition with Slice-wise Updates



This chapter introduces the problem of tracking CP decompositions of dense tensors with slice-wise updates. An incremental algorithm, OnlineCP is presented, which is the first work that is able to address such problem in tensors with order that is higher than 3. The content of this chapter is adapted from the following published paper:

Zhou, S., Vinh, N.X., Bailey, J., Jia, Y. and Davidson, I., 2016, August. Accelerating online CP decompositions for higher order tensors. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1375-1384). ACM.

3.1 Introduction

Numerous types of data are naturally represented as multi-dimensional structures. The tensor, a multi-way generalization of the matrix, is useful for representing such data. Similar to matrix analysis tools, such as PCA and SVD, tensor decomposition is a popular approach for feature extraction, dimensionality reduction and knowledge discovery on multi-way data. It has been extensively studied and widely applied in various fields of science, including chemometrics [Acar et al., 2011a], signal processing [Cichocki et al., 2015], computer vision [Hu et al., 2011, Vasilescu and Terzopoulos, 2002b], graph and network analysis [Dunlavy et al., 2011, Liu et al., 2012b] and time series analysis [Cai et al., 2015].

In the era of big data, data is often dynamically changing over time, and a large volume of new data can be generated at high velocity. In such dynamic environments, a data tensor may be expanded, shrunk or modified on any of its dimensions. For example, given a network monitoring tensor structured as $source \times destination \times port \times time$, a large number of network transactions are generated every second, which can be recorded by appending new slices to the tensor on its time mode. Additionally, new IP addresses may be added and invalid addresses may be removed from the data tensor. Overall, this data tensor is highly dynamic.

As tensor decomposition is usually the first and necessary step for analyzing multi-way data, in this work, we aim to address the problem of how to adaptively track the decompositions for such time-evolving tensors. Specifically, we are particularly interested in dynamic tensors that are incrementally growing over time, while the other dimensions remain unchanged. These are the most common type of dynamic tensors that occur in practice. We refer to such tensors as *online tensors*, also known as tensor streams and incremental tensors [Sun et al., 2006, Sun et al., 2008].

Finding the decompositions for large-scale online tensors is challenging. The difficulty mainly arises from two factors. First, as online tensors are growing with time, their overall size is potentially unbounded. Thus, tensor decomposition techniques for such tensors need to be highly efficient and scalable, from both time and space perspectives. Second, a high data generation rate requires tensor decomposition methods that are able to provide real-time or near real-time performance [Cichocki, 2014]. However, traditional tensor decomposition techniques, such as Tucker and CP decompositions, cannot be directly applied to this scenario because: (i) they require the availability of the full data for the decomposition, thus having a large memory requirement; and (ii) their fitting algorithms, Higher-Order SVD (HOSVD) for Tucker [De Lathauwer et al., 2000], and ALS or other variants for CP [Comon et al., 2009], are usually computationally too expensive for large-scale tensors.

A recipe for addressing the above challenges is to adapt existing approaches using online techniques. In recent years, several studies have been conducted on tracking the Tucker decomposition of online tensors by incorporating online techniques, such as incremental SVD [Hu et al., 2011, Ma et al., 2009, Sobral et al., 2014], and incremental update of covariance matrices [Sun et al., 2006, Sun et al., 2008]. However, there is a limited amount of work on tracking the CP decomposition of an online tensor. The only work in the literature, proposed in [Nion and Sidiropoulos, 2009], specifically only deals with 3^{rd}-order tensors and there is no provision for higher-order tensors with more than 3 dimensions. To fill this gap, we propose an efficient algorithm to find the CP decomposition of large-scale high-order online tensors, with low space and time usage. We summarize our contributions as follows:

- We propose a scalable algorithm for efficiently tracking the CP decompositions of dynamic tensors with slice-wise updates. Not limited to basic 3^{rd}-order tensors, our model can also handle higher-order tensors that have more than 3 dimensions.

- Through experimental evaluation on seven real-world datasets, we show that our approach can provide more accurate results and higher efficiency, compared with state-of-the-art approaches.

- Based on empirical analysis on synthetic datasets, our algorithm produces more stable decompositions than existing online approaches, as well as better scalability.

The rest of this chapter is organized as follows. §3.2 gives a review of current techniques on decomposing dynamic tensors with slice-wise updates. Our proposal is discussed in §3.3. We start from 3^{rd}-order tensors and then extend this model to general tensors that have an arbitrary number of dimensions. After that, the performance of our approach is evaluated on both real-world and synthetic datasets in §3.4. Lastly, §3.5 concludes this chapter.

3.2 Related Work

The problem of decomposing online tensors was originally proposed in [Sun et al., 2006, Sun et al., 2008], wherein they refer to this problem as Incremental Tensor Analysis (ITA). Three variants of ITA are discussed in their work: (1) dynamic tensor analysis (DTA) modifies the covariance matrices calculation step in typical HOSVD in an incremental fashion; (2) stream tensor analysis (STA) is an approximation of DTA by the SPIRIT algorithm [Papadimitriou et al., 2005]; and (3) window-based tensor analysis (WTA) uses a sliding window strategy to improve the efficiency of DTA. The main issue of these techniques is that they have not fully optimized the most time consuming step, i.e., diagonalizing the covariance matrix for each mode, which limits their efficiency. To overcome this issue, [Liu et al., 2012b] propose an efficient algorithm that enforces the diagonalization on the core tensors only. Additionally, another trend for improving the efficiency of HOSVD on online tensors is to replace SVD with incremental SVD algorithms. Several applications of this idea can be found in computer vision [Hu et al., 2011, Ma et al., 2009, Sobral et al., 2014] and anomaly detection [Shi et al., 2015] fields.

The major difference between the aforementioned techniques and our approach is that they are online versions of Tucker decomposition. Although CP decomposition can be viewed as a special case of Tucker with super-diagonal core tensor, none of the above methods provides a way to enforce this constraint. As a result, these algorithms are not suitable for tracking the CP decompositions of online tensors.

Unlike the existing extensive studies on online Tucker decomposition, there is a limited research reported for online CP decomposition. The most related work to ours was [Nion and Sidiropoulos, 2009], which introduce two adaptive algorithms that specifically focus on CP decomposition: Simultaneous Diagonalization Tracking (SDT) that incrementally tracks the SVD of the unfolded tensor; and Recursive Least Squares Tracking (RLST), which recursively updates the decomposition factors by minimizing the mean squared error. However, the major drawback of this work is that they work on 3^{rd}-order tensors only, while in contrast we propose a general approach that can incrementally track the CP decompositions of tensors with arbitrary dimensions.

In another related area, among the studies that focus on improving CP decomposition for handling large-scale tensors, GridTF, a grid-based tensor factorization algorithm [Phan and Cichocki, 2011] is particularly related to our problem. The main idea of GridTF is to partition the large tensor into a number of small grids. These grids are then factorized by a typical ALS algorithm in parallel. Finally, the resulting decompositions of these sub-tensors are combined together using an iterative approach. In fact, as stated by the authors, if such partitioning is enforced on the time mode only, then GridTF can be used for online CP decomposition problem.

3.3 OnlineCP Algorithm

In this section, we introduce our proposal for tracking the CP decomposition of online multi-way data in an incremental setting. For presentation clarity, initially a 3^{rd}-order case will be discussed. Then, we further extend to more general situations, where our proposed algorithm is able to handle tensors that have arbitrary number of modes. Without loss of generality, we assume the last mode of a tensor is always the one growing over time, while the size of the other modes are kept unchanged with time.

3.3.1 Third-order Tensors

By using notations introduced in §2.1, formally speaking, the problem we are addressing is defined as: given *(i)* an online tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times (t_{old}+t_{new})}$, where $\mathcal{X}$ is expanded from $\mathcal{X}_{old} \in \mathbb{R}^{I \times J \times t_{old}}$ by appending a new chunk of data $\mathcal{X}_{new} \in \mathbb{R}^{I \times J \times t_{new}}$ at its last mode, *(ii)* the CP decomposition of $\mathcal{X}_{old}$, $[\![\mathbf{A}_{old}, \mathbf{B}_{old}, \mathbf{C}_{old}]\!]$, how can we find the CP decomposition $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of $\mathcal{X}$ at a low cost? Considering that in most online systems, the size of the new incoming data is usually much smaller than that of all existing historical data, thus we assume $t_{new} \ll t_{old}$.

Similar to the classic ALS algorithm, our approach handles the problem in an alternating update fashion. That is, we first fix $\mathbf{A}$ and $\mathbf{B}$, to update $\mathbf{C}$, and then sequentially update $\mathbf{A}$ and $\mathbf{B}$, by fixing the other two.

Update Temporal Mode C

Recall that to learn $\mathbf{C}$, ALS solves a least square problem similar to Eq. (2.11) by fixing $\mathbf{A}$ and $\mathbf{B}$. With above notations, this can be rewritten as

$$\begin{aligned}
\mathbf{C} &\leftarrow \arg\min_{\mathbf{C}} \frac{1}{2} \left\| \mathbf{X}_{(3)} - \mathbf{C}(\mathbf{B} \odot \mathbf{A})^\top \right\|^2 \\
&= \arg\min_{\mathbf{C}} \frac{1}{2} \left\| \begin{bmatrix} \mathbf{X}_{old(3)} \\ \mathbf{X}_{new(3)} \end{bmatrix} - \begin{bmatrix} \mathbf{C}^{(1)} \\ \mathbf{C}^{(2)} \end{bmatrix} (\mathbf{B} \odot \mathbf{A})^\top \right\|^2 \\
&= \arg\min_{\mathbf{C}} \frac{1}{2} \left\| \begin{bmatrix} \mathbf{X}_{old(3)} - \mathbf{C}^{(1)}(\mathbf{B} \odot \mathbf{A})^\top \\ \mathbf{X}_{new(3)} - \mathbf{C}^{(2)}(\mathbf{B} \odot \mathbf{A})^\top \end{bmatrix} \right\|^2
\end{aligned} \tag{3.1}$$

It is clear that the norm of the first row is minimized with $\mathbf{C}_{old}$, since $\mathbf{A}$ and $\mathbf{B}$ are fixed as $\mathbf{A}_{old}$ and $\mathbf{B}_{old}$ from the last time step. The optimal solution to minimize the second row is $\mathbf{C}^{(2)} = \mathbf{X}_{new(3)}((\mathbf{B} \odot \mathbf{A})^\top)^\dagger$. As a result, $\mathbf{C}$ is updated by appending the projection $\mathbf{C}_{new}$ of $\mathbf{X}_{new(3)}$ via the loading matrices $\mathbf{A}$ and $\mathbf{B}$ of previous time step, to $\mathbf{C}_{old}$, i.e.,

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_{old} \\ \mathbf{C}_{new} \end{bmatrix} = \begin{bmatrix} \mathbf{C}_{old} \\ \mathbf{X}_{new(3)}((\mathbf{B} \odot \mathbf{A})^\top)^\dagger \end{bmatrix} \tag{3.2}$$

Update Non-temporal Modes A and B

First, we update $\mathbf{A}$. By fixing $\mathbf{B}$ and $\mathbf{C}$, the estimations error $\mathcal{L}$ can be written as $\frac{1}{2}\left\|\mathbf{X}_{(1)} - \mathbf{A}(\mathbf{C} \odot \mathbf{B})^\top\right\|^2$, and the derivative of $\mathcal{L}$ w.r.t. $\mathbf{A}$ is

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B}) - \mathbf{A}(\mathbf{C} \odot \mathbf{B})^\top(\mathbf{C} \odot \mathbf{B})$$

By setting the derivative to zero and letting $\mathbf{P} = \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})$ and $\mathbf{Q} = (\mathbf{C} \odot \mathbf{B})^\top(\mathbf{C} \odot \mathbf{B})$, we have

$$\mathbf{A} = \mathbf{P}\mathbf{Q}^{-1} \tag{3.3}$$

Directly calculating $\mathbf{P}$ and $\mathbf{Q}$ is costly. This is mainly because the output of $(\mathbf{C} \odot \mathbf{B})$ is a huge matrix of size $J(t_{old} + t_{new}) \times R$, where R is the tensor rank. It further results in $\mathcal{O}(RIJ(t_{old} + t_{new}))$ and $\mathcal{O}(R^2 J(t_{old} + t_{new}))$ operations to get $\mathbf{P}$ and $\mathbf{Q}$, respectively. Although for $\mathbf{Q}$, according to Eq. (2.1), the Khatri-Rao product can be avoided by calculating it as $(\mathbf{C}^\top\mathbf{C}) \circledast (\mathbf{B}^\top\mathbf{B})$, which has a complexity of $\mathcal{O}(R^2(J + t_{old} + t_{new}))$, this is still expensive since t_{old} is usually quite large. As a result, in order to improve the efficiency, we need a faster approach.

Firstly, let us look at $\mathbf{P}$. By representing $\mathbf{X}_{(1)}$ and $\mathbf{C}$ with the *old* and *new* components, we have

$$\begin{aligned}
\mathbf{P} &= \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B}) \\
&= \left[\mathbf{X}_{old(1)}, \mathbf{X}_{new(1)}\right]\left(\begin{bmatrix}\mathbf{C}_{old} \\ \mathbf{C}_{new}\end{bmatrix} \odot \mathbf{B}\right) \\
&= \left[\mathbf{X}_{old(1)}, \mathbf{X}_{new(1)}\right]\begin{bmatrix}\mathbf{C}_{old} \odot \mathbf{B} \\ \mathbf{C}_{new} \odot \mathbf{B}\end{bmatrix} \\
&= \mathbf{X}_{old(1)}(\mathbf{C}_{old} \odot \mathbf{B}) + \mathbf{X}_{new(1)}(\mathbf{C}_{new} \odot \mathbf{B})
\end{aligned} \tag{3.4}$$

recall that $\mathbf{B}$ has been fixed as $\mathbf{B}_{old}$, so the first part of the last line of Eq. (3.4) only contains components from the previous time step. Suppose we know this part already and denote it by $\mathbf{P}_{old}$, then Eq. (3.4) can be rewritten

as

$$\mathbf{P} = \mathbf{P}_{old} + \mathbf{X}_{new(1)}(\mathbf{C}_{new} \odot \mathbf{B}) \tag{3.5}$$

This means that by keeping a record of the previous $\mathbf{P}$, the large computation can be avoided and it can be efficiently updated in an incremental way. Specifically, suppose $\mathbf{P}$ is initialized with a small partition $\mathcal{X}(\tau) \in \mathbb{R}^{I \times J \times \tau}$ that contains the first τ slices of the data, where $\tau \ll t_{old}$, we only need $\mathcal{O}(RIJ\tau)$ operations to construct $\mathbf{P}$. Afterwards, whenever new data comes, $\mathbf{P}$ can be efficiently updated at the cost of $\mathcal{O}(RIJt_{new})$, which is independent to t_{old}.

Likewise, $\mathbf{Q}$ can be estimated as

$$\begin{aligned} \mathbf{Q} &= \mathbf{Q}_{old} + (\mathbf{C}_{new} \odot \mathbf{B})^\top (\mathbf{C}_{new} \odot \mathbf{B}) \\ &= \mathbf{Q}_{old} + (\mathbf{C}_{new}^\top \mathbf{C}_{new}) \circledast (\mathbf{B}^\top \mathbf{B}) \end{aligned} \tag{3.6}$$

Thus, by storing the information of previous decomposition with *complementary* matrices $\mathbf{P}$ and $\mathbf{Q}$, we achieve the update rule for $\mathbf{A}$ as follows,

$$\begin{aligned} &\mathbf{P} \leftarrow \mathbf{P} + \mathbf{X}_{new(1)}(\mathbf{C}_{new} \odot \mathbf{B}) \\ &\mathbf{Q} \leftarrow \mathbf{Q} + (\mathbf{C}_{new}^\top \mathbf{C}_{new}) \circledast (\mathbf{B}^\top \mathbf{B}) \\ &\mathbf{A} \leftarrow \mathbf{P}\mathbf{Q}^{-1} \end{aligned} \tag{3.7}$$

The update rule for $\mathbf{B}$ can be derived in a similar way as

$$\begin{aligned} &\mathbf{U} \leftarrow \mathbf{U} + \mathbf{X}_{new(2)}(\mathbf{C}_{new} \odot \mathbf{A}) \\ &\mathbf{V} \leftarrow \mathbf{V} + (\mathbf{C}_{new}^\top \mathbf{C}_{new}) \circledast (\mathbf{A}^\top \mathbf{A}) \\ &\mathbf{B} \leftarrow \mathbf{U}\mathbf{V}^{-1} \end{aligned} \tag{3.8}$$

where $\mathbf{U} = \mathbf{X}_{(2)}(\mathbf{C} \odot \mathbf{A})$, $\mathbf{V} = (\mathbf{C} \odot \mathbf{A})^\top (\mathbf{C} \odot \mathbf{A})$ are the two complementary matrices of mode 2.

To sum up: For a 3^{rd}-order tensor that grows with slice-wise updates, we propose a highly efficient algorithm for tracking its CP decomposition on the fly. We name this algorithm as *OnlineCP*, comprising the following two stages:

1) Initialization stage: for non-temporal modes, complementary matrices $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{U}$ and $\mathbf{V}$ are initialized with the initial tensor $\mathcal{X}_{init}$ and its CP decomposition $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ as

$$\mathbf{P} = \mathbf{X}_{init(1)}(\mathbf{C} \odot \mathbf{B}), \mathbf{Q} = (\mathbf{C}^\top \mathbf{C}) \circledast (\mathbf{B}^\top \mathbf{B})$$
$$\mathbf{U} = \mathbf{X}_{init(2)}(\mathbf{C} \odot \mathbf{A}), \mathbf{V} = (\mathbf{C}^\top \mathbf{C}) \circledast (\mathbf{A}^\top \mathbf{A})$$

2) Update stage: for each new incoming data $\mathcal{X}_{new}$, it is processed as

a) for the temporal mode 3, $\mathbf{C}$ is updated with Eq. (3.2)

b) for non-temporal modes 1 and 2, $\mathbf{A}$ is updated with Eq. (3.7) and $\mathbf{B}$ is updated with Eq. (3.8), respectively.

3.3.2 Extending to Higher-Order Tensors

We now show how to extend our approach to higher-order cases. Let $\mathcal{X}_{old} \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times t_{old}}$ be an N^{th}-order tensor, $[\![\mathbf{A}_{old}^{(1)}, \dots, \mathbf{A}_{old}^{(N-1)}, \mathbf{A}_{old}^{(N)}]\!]$ be its CP decomposition, the N-th mode be the time. A new tensor $\mathcal{X}_{new} \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times t_{new}}$ is added to $\mathcal{X}_{old}$ to form a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times (t_{old}+t_{new})}$, where $t_{old} \gg t_{new}$. In addition, we create and store two sets of complementary matrices $\mathbf{P}^{(1)}, \dots, \mathbf{P}^{(N-1)}$ and $\mathbf{Q}^{(1)}, \dots, \mathbf{Q}^{(N-1)}$, where $\mathbf{P}^{(n)}$ and $\mathbf{Q}^{(n)}, n \in [1, N-1]$, are the complementary matrices for mode n. We are interested in finding the CP decomposition $[\![\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N-1)}, \mathbf{A}^{(N)}]\!]$ of $\mathcal{X}$.

Update Temporal Mode

Similar to the 3rd-order case, the loading matrix of the time mode, $\mathbf{A}^{(N)}$, is updated at first by fixing the other loading matrices and minimizing the estimation error as

$$\mathbf{A}^{(N)} \leftarrow \underset{\mathbf{A}^{(N)}}{\arg\min} \frac{1}{2} \left\| \mathbf{X}_{(N)} - \mathbf{A}^{(N)} (\odot_{i=1}^{N-1} \mathbf{A}^{(i)})^\top \right\|^2$$

Basically, the above equation has the same structure as Eq. (3.1), so we have a similar update rule for $\mathbf{A}^{(N)}$

$$\mathbf{A}^{(N)} \leftarrow \begin{bmatrix} \mathbf{A}^{(N)}_{old} \\ \mathbf{A}^{(N)}_{new} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^{(N)}_{old} \\ \mathbf{X}_{new(N)}((\odot_{i=1}^{N-1}\mathbf{A}^{(i)})^\top)^\dagger \end{bmatrix}$$

Update Non-temporal Modes

For each non-temporal mode $n \in [1, N-1]$, the estimation error on mode n is $\frac{1}{2}\left\|\mathbf{X}_{(n)} - \mathbf{A}^{(n)}(\odot_{i\neq n}^{N}\mathbf{A}^{(i)})^\top\right\|^2$, and similar update rule as Eq. (3.7) can be applied, that is

$$\begin{aligned}
\mathbf{P}^{(n)} &\leftarrow \mathbf{P}^{(n)} + \mathbf{X}_{new(n)}\left(\mathbf{A}^{(N)}_{new} \odot \mathbf{K}^{(n)}\right) \\
\mathbf{Q}^{(n)} &\leftarrow \mathbf{Q}^{(n)} + \left({\mathbf{A}^{(N)}_{new}}^\top \mathbf{A}^{(N)}_{new}\right) \circledast \mathbf{H}^{(n)} \\
\mathbf{A}^{(n)} &\leftarrow \mathbf{P}^{(n)}(\mathbf{Q}^{(n)})^{-1}
\end{aligned}$$

where $\mathbf{K}^{(n)} = \odot_{i\neq n}^{N-1}\mathbf{A}^{(i)}$ and $\mathbf{H}^{(n)} = \circledast_{i\neq n}^{N-1}{\mathbf{A}^{(i)}}^\top\mathbf{A}^{(i)}$.

Avoid Duplicated Computation

In fact, if we were to compute every $\mathbf{K}^{(n)}$ for each $n \in [1, N-1]$, there would be some redundant computation among them. Take a 5^{th}-order tensor for example, where $\mathbf{K}^{(1)} = \mathbf{A}^{(4)} \odot \mathbf{A}^{(3)} \odot \mathbf{A}^{(2)}$, $\mathbf{K}^{(2)} = \mathbf{A}^{(4)} \odot \mathbf{A}^{(3)} \odot \mathbf{A}^{(1)}$, $\mathbf{K}^{(3)} = \mathbf{A}^{(4)} \odot \mathbf{A}^{(2)} \odot \mathbf{A}^{(1)}$, and $\mathbf{K}^{(4)} = \mathbf{A}^{(3)} \odot \mathbf{A}^{(2)} \odot \mathbf{A}^{(1)}$. It is clear that both $\mathbf{K}^{(1)}$ and $\mathbf{K}^{(2)}$ have computed $\mathbf{A}^{(4)} \odot \mathbf{A}^{(3)}$, and $\mathbf{K}^{(3)}$ and $\mathbf{K}^{(4)}$ share a common computation $\mathbf{A}^{(2)} \odot \mathbf{A}^{(1)}$. These redundant Khatri-Rao products are computationally expensive to calculate, and more importantly, the amount of redundancy will dramatically increase with the number of modes N, since more common components are shared. To overcome this issue, we use a dynamic programming strategy to compute all the $\mathbf{K}^{(n)}$'s in one run, by making use of intermediate results. This process is detailed in Algorithm 3.1 and an example of a 6^{th}-order tensor is given in Figure 3.1.

Algorithm 3.1: Get a list of Khatri-Rao products

```
Input: A list of loading matrices $[\mathbf{A}^{(N-1)}, \ldots, \mathbf{A}^{(1)}]$
Output: A list of Khatri-Rao products $[\mathbf{K}^{(1)}, \ldots, \mathbf{K}^{(N-1)}]$
$left \leftarrow [\mathbf{A}^{(N-1)}]$
$right \leftarrow [\mathbf{A}^{(1)}]$
if $N > 3$ then
    for $n \leftarrow 2$ to $N-2$ do
        $left[n] \leftarrow left[n-1] \odot \mathbf{A}^{(N-n)}$
        $right[n] \leftarrow \mathbf{A}^{(n)} \odot right[n-1]$
    end
end
$\mathbf{K}^{(1)} \leftarrow left[N-2]$
$\mathbf{K}^{(N-1)} \leftarrow right[N-2]$
if $N > 3$ then
    for $n \leftarrow 2$ to $N-2$ do
        $\mathbf{K}^{(n)} \leftarrow left[N-n-1] \odot right[n-1]$
    end
end
```

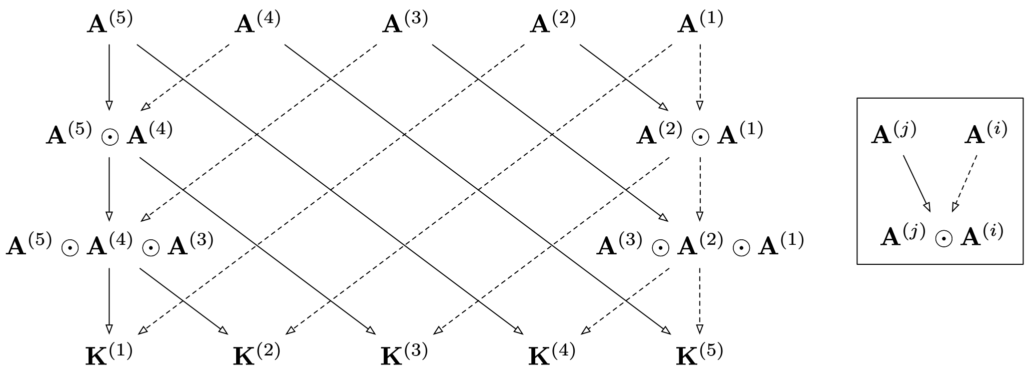

$\mathbf{A}^{(5)}$
$\mathbf{A}^{(4)}$
$\mathbf{A}^{(3)}$
$\mathbf{A}^{(2)}$
$\mathbf{A}^{(1)}$
$\mathbf{A}^{(5)} \odot \mathbf{A}^{(4)}$
$\mathbf{A}^{(2)} \odot \mathbf{A}^{(1)}$
$\mathbf{A}^{(5)} \odot \mathbf{A}^{(4)} \odot \mathbf{A}^{(3)}$
$\mathbf{A}^{(3)} \odot \mathbf{A}^{(2)} \odot \mathbf{A}^{(1)}$
$\mathbf{K}^{(1)}$
$\mathbf{K}^{(2)}$
$\mathbf{K}^{(3)}$
$\mathbf{K}^{(4)}$
$\mathbf{K}^{(5)}$
$\mathbf{A}^{(j)}$
$\mathbf{A}^{(i)}$
$\mathbf{A}^{(j)} \odot \mathbf{A}^{(i)}$

Figure 3.1: A 6th-order example to get all $\mathbf{K}^{(n)}$'s together. A Khatri-Rao product is represented by two arrows, of which the solid one linked to the first input. The two lists, *left* and *right*, are indicated by the two columns on the graph.

Algorithm 3.2: Initialization stage of OnlineCP

```
Input: Initial tensor $\mathcal{X}_{init}$, loading matrices $\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)}$
Output: complementary matrices $\mathbf{P}^{(1)}, \dots, \mathbf{P}^{(N-1)}$ and
        $\mathbf{Q}^{(1)}, \dots, \mathbf{Q}^{(N-1)}$
1  Get $\mathbf{K}^{(1)}, \mathbf{K}^{(2)}, \dots, \mathbf{K}^{(N-1)}$ by Algorithm 3.1
2  $\mathbf{H} \leftarrow \circledast^{N} \mathbf{A}^{(i)\top} \mathbf{A}^{(i)}$
3  for $n \leftarrow 1$ to $N-1$ do
4      $\mathbf{P}^{(n)} \leftarrow \mathbf{X}_{init(n)} (\mathbf{A}^{(N)} \odot \mathbf{K}^{(n)})$
5      $\mathbf{Q}^{(n)} \leftarrow \mathbf{H} \oslash (\mathbf{A}^{(n)\top} \mathbf{A}^{(n)})$
6  end
```

Algorithm 3.3: Update stage of OnlineCP

```
Input: Loading matrices $\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)}$, complementary matrices
       $\mathbf{P}^{(1)}, \dots, \mathbf{P}^{(N-1)}$, $\mathbf{Q}^{(1)}, \dots, \mathbf{Q}^{(N-1)}$, and new data tensor $\mathcal{X}_{new}$
Output: Updated loading matrices $\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)}$, and updated
        complementary matrices $\mathbf{P}^{(1)}, \dots, \mathbf{P}^{(N-1)}$, $\mathbf{Q}^{(1)}, \dots, \mathbf{Q}^{(N-1)}$
1  Get $\mathbf{K}^{(1)}, \mathbf{K}^{(2)}, \dots, \mathbf{K}^{(N-1)}$ by Algorithm 3.1
2  $\mathbf{H} \leftarrow \circledast_{i=1}^{N-1} \mathbf{A}^{(i)\top} \mathbf{A}^{(i)}$
   // update $\mathbf{A}^{(N)}$
3  $\mathbf{K}^{(N)} \leftarrow \mathbf{K}^{(1)} \odot \mathbf{A}^{(1)}$
4  $\mathbf{A}^{(N)}_{new} \leftarrow \mathbf{X}_{new(N)} ((\mathbf{K}^{(N)})^{\top})^{\dagger}$
5  $\mathbf{A}^{(N)} \leftarrow \begin{bmatrix} \mathbf{A}^{(N)}_{old} \\ \mathbf{A}^{(N)}_{new} \end{bmatrix}$
   // update other modes
6  for $n \leftarrow 1$ to $N-1$ do
7      $\mathbf{P}^{(n)} \leftarrow \mathbf{P}^{(n)} + \mathbf{X}_{new(n)} (\mathbf{A}^{(N)}_{new} \odot \mathbf{K}^{(n)})$
8      $\mathbf{H}^{(n)} \leftarrow \mathbf{H} \oslash (\mathbf{A}^{(n)\top} \mathbf{A}^{(n)})$
9      $\mathbf{Q}^{(n)} \leftarrow \mathbf{Q}^{(n)} + (\mathbf{A}^{(N)\top}_{new} \mathbf{A}^{(N)}_{new}) \circledast \mathbf{H}^{(n)}$
10     $\mathbf{A}^{(n)} \leftarrow \mathbf{P}^{(n)} (\mathbf{Q}^{(n)})^{-1}$
11 end
```

The main idea of Algorithm 3.1 is to go through the loading matrix list $\mathbf{A}^{(N-1)}, \ldots, \mathbf{A}^{(2)}, \mathbf{A}^{(1)}$ from both ends, until the algorithm reaches the results of $\mathbf{K}^{(1)}$ and $\mathbf{K}^{(N-1)}$ (lines 3 to 8). After that, for the rest of $\mathbf{K}^{(i)}$ where $i \in [2, N-2]$, they are computed as the Khatri-Rao products of the intermediate results from the last loop (lines 11 to 15).

For the $\mathbf{H}^{(n)}$'s, it is obvious that calculating each individual $\mathbf{H}^{(n)}$ by itself is inefficient. Exploiting the fact that for $\forall i, j \in [1, N-1]$, $\mathbf{H}^{(i)} \circledast (\mathbf{A}^{(i)\top}\mathbf{A}^{(i)}) = \mathbf{H}^{(j)} \circledast (\mathbf{A}^{(j)\top}\mathbf{A}^{(j)}) = \mathbf{H}$, in each round of update, $\mathbf{H}$ is calculated first, then each $\mathbf{H}^{(n)}$ is obtained as $\mathbf{H} \oslash (\mathbf{A}^{(n)\top}\mathbf{A}^{(n)})$, where $\oslash$ is the element-wise division.

Finally, by putting everything together, we obtain the general version of our OnlineCP algorithm, as presented in Algorithm 3.2 and 3.3.

3.3.3 Complexity Analysis

Following the same notation as §3.3.2, let R be the tensor rank, $S = \prod_{i=1}^{N-1} I_i$, and $J = \sum_{i=1}^{N-1} I_i$. To process a new chunk of data $\mathcal{X}_{new}$, it takes up to $(N-1)S$ operations to get all the $\mathbf{K}^{(n)}, n \in [1, N-1]$, and $\mathbf{H}$ can be obtained in $R^2J + (N-2)R^2$ operations (lines 1 and 2 in Algorithm 3.3). To update the time mode, RS, St_{new}, and $RSt_{new} + R^2t_{new} + R^3$ operations are required to get $\mathbf{K}^{(N)}$, $\mathbf{X}_{new(N)}$, and $\mathbf{A}_{new}^{(N)}$, respectively (lines 3, 4). Note that the pseudoinverse $((\mathbf{K}^{(N)})^\top)^\dagger$ can be replaced by $\mathbf{K}^{(N)}\mathbf{H}^\dagger$ as $(\mathbf{A} \odot \mathbf{B})^\dagger = ((\mathbf{A}^\top\mathbf{A}) \circledast (\mathbf{B}^\top\mathbf{B}))^\dagger(\mathbf{A} \odot \mathbf{B})^\top$ based on Eq. (2.2). For each non-temporal mode n (lines 7 to 10), St_{new}, RSt_{new}/I_n ($\approx St_{new}$, since R is usually smaller than I_n), RSt_{new} and RI_n operations are required for the unfolding, Khatri-Rao, multiplication and addition in the step to update $\mathbf{P}^{(n)}$; and $\mathbf{Q}^{(n)}$ takes $R^2I_n + R^2t_{new} + 3R^2$ operations to update; then the updated $\mathbf{A}^{(n)}$ can be calculated in $R^3 + R^2I_n$ operations. Thus, to update the loading matrix $\mathbf{A}^{(n)}$ of mode n, $(2R^2 + R)I_n + (R+2)St_{new} + R^3 + (3 + t_{new})R^2$ operations is required and the whole update procedure for non-temporal modes takes $(2R^2 + R)J + (N-1)(R+2)St_{new} + (N-1)(R^3 + (3 + t_{new})R^2)$ operations. Overall, as S is usually much larger than other factors, the time complexity of OnlineCP can be written as $\mathcal{O}(NRSt_{new})$, which is constant

Table 3.1: Complexity comparison between OnlineCP and existing methods.

	Time	Space
OnlineCP	$\mathcal{O}(NRSt_{new})$	$St_{new} + (2J + t_{old})R + (N-1)R^2$
ALS	$\mathcal{O}(NRS(t_{old} + t_{new}))$	$S(t_{old} + t_{new})$
SDT	$\mathcal{O}(R^2(t_{old} + S))$	$St_{new} + (J + S + 2t_{old})R + 3R^2$
RLST	$\mathcal{O}(R^2 S)$	$St_{new} + (J + t_{old} + 2S)R + 2R^2$
GridTF	$\mathcal{O}(NRSt_{new} + R^2(J + t_{old} + t_{new}))$	$St_{new} + (J + t_{old})R$

w.r.t. the length of processed data t_{old}.

In terms of space consumption, unlike ALS that needs to store all the data, OnlineCP is quite efficient since only the new data, previous loading matrices and complementary matrices need to be recorded. Hence, the total cost of space is $St_{new} + (2J + t_{old})R + (N-1)R^2$.

We summarize the complexity of our approach in Table 3.1, along with other existing approaches. Note that the complexities of SDT and RLST are based on the exponential window [Nion and Sidiropoulos, 2009], which considers all existing data while leverages their importance by a forgetting factor λ. In addition, as they only work on 3rd-order tensors, when other methods are compared to them, N should be set to 3. Another remark is that the time complexities of ALS and GridTF are based on one iteration only, in reality they would take a few iterations until convergence.

3.4 Empirical Analysis

In this section, we evaluate our OnlineCP algorithm, compared to existing techniques. We first examine their effectiveness and efficiency on seven real-world datasets. After that, based on the investigation on synthetic tensors, we further analyze the critical factors that can affect the performance of our approach, along with other baselines.

3.4.1 Real-world Datasets

Experimental Specifications

Datasets: The experiments are conducted on seven real-world datasets of varying characteristics, all are naturally of multi-way structures. These are two image datasets: (i) Columbia Object Image Library (COIL); (ii) ORL Database of Faces (FACE); three human activity datasets: (iii) Daily and Sports Activities Data Set (DSA); (iv) University of Southern California Human Activity Dataset (HAD); (v) Daphnet Freezing of Gait Data Set (FOG); one chemical laboratory dataset: (vi) Gas sensor array under dynamic gas mixtures Data Set (GAS); and a (vii) road traffic dataset collected from loop detectors in Victoria, Australia (ROAD). It should be noted that large-scale tensors are usually highly sparse, which are not the target type of dynamic tensors we are dealing with in this Chapter, the adaptive algorithm for addressing those tensors will be introduced in Chapter 4. In addition, due to the high time consumption of the static ALS algorithm, it is not feasible to conduct experiment on large-scale dense dynamic tensors in this experiment setting. However, we provide analysis on synthetic tensors of larger scale to demonstrate the efficiency and scalability of our method in §3.4.3.

Each dataset is represented by a tensor with its most natural structure. For instance, FACE is represented by a $pixel \times pixel \times shot$ 3^{rd}-order tensor, while DSA is stored as an 4^{th}-order tensor of $subject \times trail \times sensor \times time$. Furthermore, since some of our baselines can only work with 3^{rd}-order tensors, for tensors with higher-order, DSA, GAS, and HAD, we randomly extract 3^{rd}-order sub-tensors from them. Conversely, to enlarge the number of higher-order tensors, image datasets, COIL and FACE have been transformed into 4^{th}-order tensors by treating each image as a collection of small *patches*, which forms the extra order. As a result, there are five datasets having two versions of representation: a 3^{rd}-order one, indicated by suffix *3D*, and a higher-order form with suffix *HD*. The details of these datasets can be found in Table 3.2.

Baselines: In this experiment, five baselines have been selected as the

Table 3.2: Details of datasets

Datasets	Size	Slice Size $S = \prod_{i=1}^{N-1} I_i$	Source
COIL-3D	$128 \times 128 \times 240$	16,384	[Nene et al., 1996]
COIL-HD	$64 \times 64 \times 25 \times 240$	102,400	
DSA-3D	$8 \times 45 \times 750$	360	[Altun et al., 2010]
DSA-HD	$19 \times 8 \times 45 \times 750$	6,840	
FACE-3D	$112 \times 92 \times 400$	10,304	[Samaria and Harter, 1994]
FACE-HD	$28 \times 23 \times 16 \times 400$	10,304	
FOG	$10 \times 9 \times 1000$	90	[Bachlin et al., 2010]
GAS-3D	$30 \times 8 \times 2970$	240	[Fonollosa et al., 2015]
GAS-HD	$30 \times 6 \times 8 \times 2970$	1,440	
HAD-3D	$14 \times 6 \times 500$	64	[Zhang and Sawchuk, 2012]
HAD-HD	$14 \times 12 \times 5 \times 6 \times 500$	3,840	
ROAD	$4666 \times 96 \times 1826$	447,936	[Schimbinschi et al., 2015]

competitors to evaluate the performance.

(i) Batch Cold: an implementation of ALS algorithm in Tensor Toolbox [Bader et al., 2015] without special initialization.

(ii) Batch Hot: the same ALS as above but the CP decomposition of the last time step is used as the initialization for the current decomposition.

(iii) SDT [Nion and Sidiropoulos, 2009]: an adaptive algorithm based on incrementally tracking the SVD of the unfolded tensor.

(iv) RLST: another online approach proposed in [Nion and Sidiropoulos, 2009]. Instead of tracking the SVD, recursive updates are performed to minimize the mean squared error on new data.

(v) GridTF [Phan and Cichocki, 2011]: an divide-and-conquer based algorithm. To find CP decompositions for online tensors, the partitioning is enforced on the time mode only.

Evaluation metrics: Two performance metrics are used in our evaluation. *Fitness* is the effectiveness measurement defined as

$$fitness \triangleq \left(1 - \frac{\left\|\hat{\mathcal{X}} - \mathcal{X}\right\|}{\left\|\mathcal{X}\right\|}\right) \times 100\%$$

where $\mathcal{X}$ is the ground truth, $\hat{\mathcal{X}}$ is the estimation and $\left\|\bullet\right\|$ denotes the Frobenius norm. In addition, the *average running time* for processing one data slice, measured in seconds, is used as efficiency measurement.

Experimental setup: The experiments are divided into two parts. The first part is to decompose the 3^{rd}-order tensors with all baselines. For the second part, only Batch Hot, GridTF and our approach are used. This is because both SDT and RLST work on 3^{rd}-order tensors only, and Batch Cold does not show better performance compared to Batch Hot, while taking a much longer time to run.

Apart from the difference in the number of competitors, the experimental protocol is the same for both third and higher order tests. Specifically, for a given dataset, the first 20% of the data is decomposed by ALS and its CP decomposition is used to initialize all algorithms. After that, the remaining 80% of the data is appended to the existing tensor by one slice at a time. At each time step, after processing the appended data slice, all methods calculate the fitness of their current decomposition with their updated loading matrices, as well as their processing time for this new slice. The same experiment is replicated 10 times for all datasets on a workstation with dual Intel Xeon processors, 64 GB RAM. The final results are averaged over these 10 runs.

There are some settings of parameters that need to be clarified. Firstly, since we only care about the relative performance comparison among different algorithms, it is not necessary to pursue the best rank decomposition for each dataset. As a result, the rank R is fixed to 5 for all datasets. Additionally, for the initial CP decomposition, the tolerance ε is set to $1e-8$ and the maximum number of iterations *maxiters* is set to 100 to ensure a good start, as the performance of all online algorithms depends on the quality of the initial decomposition.

In terms of method-specific parameters, for the two batch algorithms, the default settings, $\varepsilon = 1e-4$ and $maxiters = 50$ are used. For GridTF, which contains an ALS procedure for the *new* data slice and a recursively update procedure for estimating the *whole* current tensor, the same default parameters are chosen for the ALS step; while $\varepsilon = 1e-2$ and $maxiters = 50$

are used for the update phase. Additionally, since batch algorithms do not provide a weighting strategy to differentiate the importance of data, in order to make a fair comparison, all the data slices are equally treated and there is no difference between older and newer ones in terms of their weights, which means the exponential window is used with $\lambda = 1$ in SDT and RLST.

Table 3.3: Mean fitness of 3^{rd}-order datasets over time (in %, the higher the values the better). For SDT, RLST, GridTF and OnlineCP, ratios of their fitness to the result of Batch Hot are shown in parenthesis. Boldface indicates the best result among these four online approaches.

Datasets	Batch Cold	Batch Hot	SDT	RLST	GridTF	OnlineCP
COIL-3D	58.31	58.76	51.31(0.87)	56.27(0.96)	54.13(0.92)	**57.43(0.98)**
DSA-3D	57.50	57.88	26.30(0.45)	27.44(0.47)	48.85(0.84)	**57.51(0.99)**
FACE-3D	75.35	75.69	70.64(0.93)	46.84(0.62)	71.91(0.95)	**75.31(0.99)**
FOG-3D	48.38	48.95	40.90(0.84)	41.28(0.84)	25.94(0.53)	**44.39(0.91)**
GAS-3D	86.56	87.06	43.93(0.50)	57.92(0.67)	48.64(0.56)	**84.94(0.98)**
HAD-3D	28.97	29.34	26.14(0.89)	28.33(0.97)	27.56(0.94)	**28.54(0.97)**
ROAD*	79.25	79.94	15.45(0.21)	61.23(0.77)	N/A	**79.72(1.00)**

Table 3.4: Mean running time of 3^{rd}-order datasets for processing one data slice (in seconds). For SDT, RLST, GridTF and OnlineCP, the ratios between the running time of Batch Hot and theirs are shown in parenthesis. Boldface indicates the best result among these four online approaches.

Datasets	Batch Cold	Batch Hot	SDT	RLST	GridTF	OnlineCP
COIL-3D	3.0255	0.1944	0.0037(52.06)	0.0041(47.79)	0.0446(4.36)	**0.0017(115.36)**
DSA-3D	0.3627	0.0246	0.0005(52.41)	**0.0003(72.45)**	0.0332(0.74)	0.0004(57.28)
FACE-3D	1.2616	0.1430	0.0034(42.06)	0.0037(38.47)	0.0439(3.26)	**0.0019(73.87)**
FOG-3D	0.2805	0.0198	0.0005(39.43)	**0.0003(72.01)**	0.0285(0.70)	0.0004(53.33)
GAS-3D	0.4095	0.0314	0.0015(20.82)	**0.0003(91.53)**	0.0265(1.18)	0.0005(64.91)
HAD-3D	0.1867	0.0154	0.0004(41.44)	**0.0003(60.94)**	0.0158(0.98)	0.0004(43.76)
ROAD*	309.5064	23.6683	0.0582(406.67)	0.0573(413.0593)	N/A	**0.0068(3480.63)**

* The result is based on only 1 run, due to the huge time consumption of batch methods. Figures of Batch Cold are estimated based on its average performance on other datasets, compared to Batch Hot. GridTF failed on this dataset because of the singular matrix problem, resulting from the large chunks of missing data in some sensors.

Results

Given a particular dataset and a specific algorithm, its fitness and processing time are two time series (averaged over 10 runs). We take the mean values of them and report the results for 3^{rd}-order tensors in Table 3.3 and 3.4, and for the higher-order tensors in Table 3.5 and 3.6. In addition, for the four online approaches, SDT, RLST, GridTF and OnlineCP, their relative performances compared to Batch Hot are also shown in the parenthesis. Finally, the best results among these four are indicated by boldface.

As can be seen from Table 3.3 and 3.4, for the two batch methods, there is no significant difference on their effectiveness. However, on all 3^{rd}-order datasets, the fitness of Batch Cold is slightly worse than that of Batch Hot. The main reason is that using the previous results as initialization can provide the ALS algorithm with a descent seeding point. In contrast, every time Batch Cold totally discards this useful information and starts to optimize from the beginning, which cannot guarantee a better or even same-quality estimation in the end. In fact, this also results in the longer running time of Batch Cold compared with Batch Hot, where the former is usually more than 10 times slower than the latter. On the other hand, even though Batch Hot improves the efficiency, its time cost is still considerably high, especially for large-scale datasets. For example, on average it takes more than 20 seconds to process one additional data slice on the ROAD dataset, while OnlineCP takes only 0.0068 seconds.

As the earliest studies of online CP decomposition, both SDT and RLST address this efficiency issue very well. Compared with Batch Hot, they shorten the mean running time by up to 400 times. RLST, in particular, was the most efficient online algorithm on 4 out of 7 3^{rd}-order tensor datasets. In fact, the efficiency of SDT is quite close to RLST, except for the GAS dataset, whose length of time mode is significantly higher than other datasets. This shows that SDT is more sensitive to the growth of time. However, the main issue of SDT and RLST is their estimation accuracy. For some datasets, such as COIL and HAD, they work fine, while for some others like DSA, they exhibit fairly poor accuracy, achieving only nearly half of the fitness of batch methods. The same accuracy problem

Table 3.5: Mean fitness of higher-order datasets over time (in %, the higher the values the better). For GridTF and OnlineCP, ratios of their fitness to the result of Batch Hot are also shown in parenthesis. Boldface indicates the best result between these two online approaches.

Datasets	Batch Hot	GridTF	OnlineCP
COIL-HD	57.43	42.69(0.74)	**56.83(0.99)**
DSA-HD	62.76	61.33(0.98)	**62.54(1.00)**
FACE-HD	74.78	67.47(0.90)	**74.45(1.00)**
GAS-HD	79.38	61.05(0.77)	**75.71(0.95)**
HAD-HD	67.36	65.57(0.97)	**67.28(1.00)**

Table 3.6: Mean running time of higher-order datasets for processing one data slice (in seconds). For GridTF and OnlineCP, the ratios between the running time of Batch Hot and theirs are shown in parenthesis. Boldface indicates the best result between these two online approaches.

Datasets	Batch Hot	GridTF	OnlineCP
COIL-HD	2.1264	0.1935(10.99)	**0.0076(280.44)**
DSA-HD	0.2217	0.0896(2.48)	**0.0029(75.43)**
FACE-HD	0.3795	0.1438(2.64)	**0.0040(94.15)**
GAS-HD	0.3154	0.1807(1.75)	**0.0016(203.47)**
HAD-HD	0.1750	0.0922(1.90)	**0.0041(42.23)**

can be observed in GridTF as well. In terms of efficiency, there is no significant difference between GridTF and Batch Hot, at least on the small size datasets. In fact, we notice that a substantial amount of time of GridTF is consumed by decomposing the new data slice and this cost is particularly dominant when the tensor size is not large enough. This can be confirmed by observing its generally better efficiency on higher-order datasets, compared to the 3^{rd}-order ones.

Our proposed algorithm, OnlineCP, shows very promising results in both accuracy and speed. On every dataset, both 3^{rd}-order and higher-order ones, our method reaches the best fitness among all online algorithms. More importantly, the estimation performance of our approach is quite stable and very comparable to the results of batch techniques. In most of the cases, the fitness of OnlineCP is less than 3% lower than that of the most accurate algorithm, Batch Hot. However, the speed of OnlineCP is orders of magnitudes faster than Batch Hot. On small and moderate size datasets, OnlineCP can be tens to hundreds of times faster than Batch Hot; and on the largest dataset, ROAD, OnlineCP improves the efficiency of Batch Hot by more than 3,000 times. Additionally, compared with another fast approach, RLST, although OnlineCP is outperformed on four datasets, its speed on these datasets is quite close to the best. On the other hand, we notice that all these datasets have fairly small slice size. In contrast, on those datasets with larger slices, such as COIL and FACE, the time consumption of our method clearly grows slower than that of RLST, showing that OnlineCP is less sensitive to the size of the data, and thus, having better scalability.

3.4.2 Sensitivity to Initialization

Throughout our experiments, an interesting observation was made: for all adaptive algorithms that make use of the previous step results, namely Batch Hot, SDT, RLST, GridTF, and OnlineCP, their best results are usually linked to a good initial fitness, while poor-quality initializations often lead them to subsequent under-fitting. To explore the impact of initialization to each algorithm, the following experiment has been conducted. We generate a synthetic tensor $\mathcal{X} \in \mathbb{R}^{20\times 20\times 100}$ by constructing from random loading matrices and then downgrade it by a Gaussian noise with a Signal-to-Interference Rate (SIR) of 20 dB. The best fitness to $\mathcal{X}$ in 10 runs of ALS is 90.14%. This tensor is then repeatedly decomposed by the above five methods for 200 times. At the beginning of each run, half of the data is used for initialization by ALS with a random tolerance from $9e-1$ to $1e-4$, to produce different level of initial fitness. Then the rest of data is sequentially added and processed by each online algorithm. The averaged final fitness over all runs is used as the effectiveness indicator, as well as the standard deviation. Table 3.7 and Figure 3.2 show the experimental results with average initial fitness as 65.78% and standard deviation as 15.3704%.

As shown in Table 3.7 and Figure 3.2, overall, the low quality initial-

Table 3.7: The final fitness averaged over 200 runs with different initial fitness. Results are displayed as *mean* ± *std*, where *mean* is the average final fitness and *std* is the standard deviation, both in % (the higher the values the better).

	Final Fitness
Batch Hot	82.57±10.1474
SDT	7.68±55.5205
RLST	33.15±36.9270
GridTF	57.43±15.2148
OnlineCP	67.67±12.9846

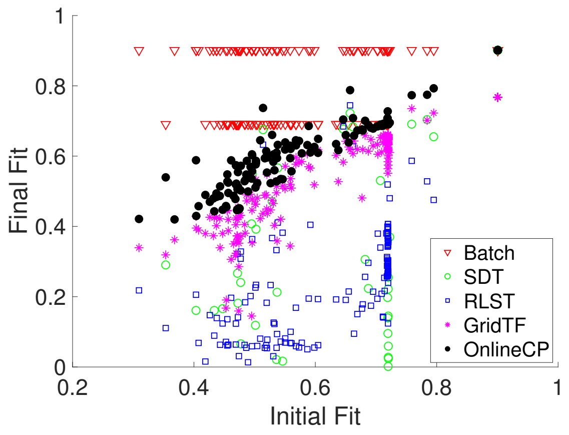


Figure 3.2: Initial fitness v.s. final fitness of 200 runs.

ization has a negative impact on all algorithms. Even the most powerful one, Batch Hot, cannot always reach the best fitness and shows a decline of 10%. For SDT and RLST, it turns out that both algorithms are significantly dependent on the initial fitness. When the initial fitness is lower than the best value, their performance can quickly drop to an unacceptable level. In addition, their results are also highly unstable, as demonstrated by a large variance. While both GridTF and OnlineCP exhibit much more stable performance, the final fitness of GridTF is considerably lower than our approach OnlineCP. This experimental evidence demonstrates that the proposed algorithm, OnlineCP, is less sensitive to the quality of the initialization, compared with exiting online methods. However, it can be seen that good initialization still plays an important role to our algorithm. Thus, for applying our method, we suggest to validate the goodness of the initialization at the beginning, in order to obtain the best subsequent effectiveness.

3.4.3 Scalability Evaluation

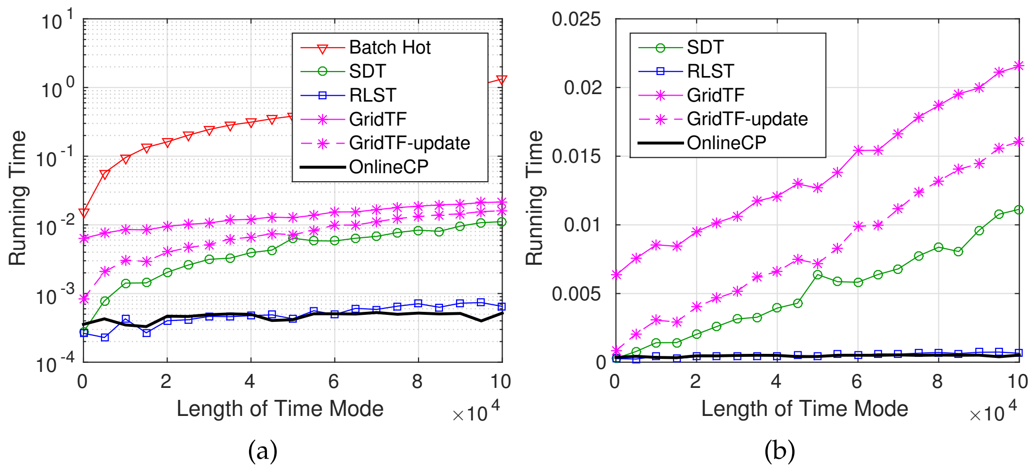


Figure 3.3: Running time (in seconds) for adding one slice to a $20 \times 20 \times (t-1)$ tensor at time t. Two figure represents the same information, differing only in the y-axis scale.

According to §3.3.3, the time complexity of each algorithm is mainly determined by the slice size and the length of processed data. To confirm our analysis and evaluate the scalability of our algorithm, firstly, a tensor $\mathcal{X} \in \mathbb{R}^{20\times20\times10^5}$ of small slice size but long time dimension is decomposed. After initializing with data of the first 100 timestamps, each method's running time for processing one data slice at each time step is measured and displayed in Figure 3.3. In addition, to examine the impact of slice size to efficiency, we fix the time mode to 100, and generate a group of tensors of different slice sizes, ranging from 100 to 9×10^6. For each tensor, its first 20% of data is used for initialization and the average running time for processing the rest data slices is shown in Figure 3.4. For better comparison,

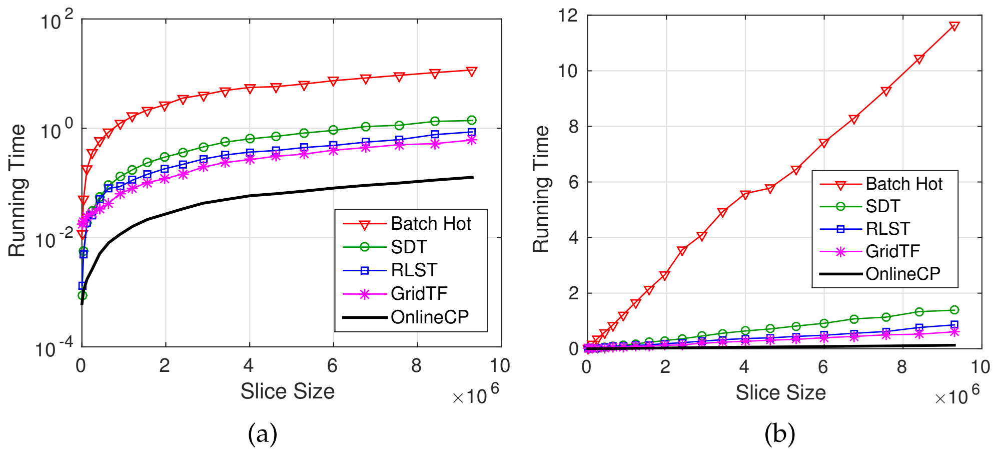


Figure 3.4: Running time (in seconds) for processing different size of tensors. Two figure represents the same information, differing only in the y-axis scale.

both Batch Hot and GridTF are forced to execute 1 iteration only in these two experiments. Note that the y-axis of Figures 3.3a and 3.4a is displayed in log scale and in Figure 3.3b, Batch Hot has been removed for better visibility.

As can be seen from Figure 3.3, both RLST and OnlineCP show constant complexities and the increasing length of processed data has no impact on them. For the other approaches, a clear linear growth with time can be observed in Batch Hot and SDT, which makes them less feasible for online learning purposes. The change of time consumption in GridTF is less obvious compared with Batch Hot and SDT. However, after removing the time used by its inner ALS procedure, similar linear trend can be seen, marked as GridTF-update in the figure.

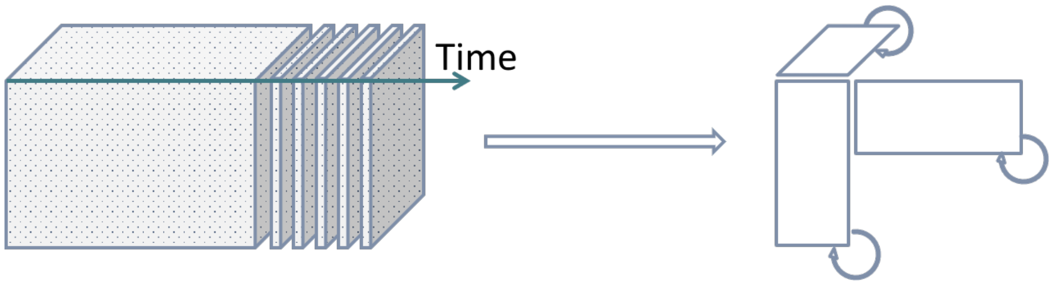
In terms of slice size, it turns out that the time consumption of all approaches are linearly increasing as the slice size grows. However, their slopes vary. Both Batch Hot and SDT show quicker growth compared to others. This is reasonable since the impact of slice size to them is also leveraged by the time mode. On the other hand, GridTF outperforms SDT and RLST when the slice gets larger. This is because the growth of slice size has impact only on its ALS procedure, which is scaled by R times, while the coefficients in the complexities of SDT and RLST w.r.t. slice size contain an R^2 term. Once again, our proposed algorithm illustrates the best performance in this experiment and even a large 3000×3000 data slice can be efficiently processed in 0.1 second.

3.5 Summary

To conclude, in this chapter, we address the problem of tracking the CP decomposition of online tensors. An online algorithm, OnlineCP, is proposed, which can efficiently track the new decomposition by using complementary matrices to temporally store the useful information of the previous time step. Furthermore, our method is not only applicable to 3^{rd}-order tensors, but also suitable for higher-order tensors that have more than 3 modes. As evaluated on both real-world and synthetic datasets, our algorithm demonstrates comparable effectiveness with the most accurate batch techniques, while significantly outperforms them in terms of efficiency. Additionally, compared with the state-of-art online techniques, the proposed algorithm shows advantages in many aspects, including effectiveness, efficiency, stability and scalability.

Chapter 4

Sparse Tensor Decomposition with Slice-wise Updates



This chapter addresses similar problem as last chapter: to find CP decompositions of dynamic tensors with slice-wise updates. However, we notice that directly applying OnlineCP algorithm to sparse tensors is of poor efficiency. To overcome this issue, we propose OnlineSCP, which shares similar principle to OnlineCP, but is fully optimized for sparse data. The content of this chapter is adapted from the following published paper:

Zhou, S., Erfani, S. and Bailey, J., 2018, November. Online CP decomposition for sparse tensors. In *2018 IEEE International Conference on Data Mining (ICDM)* (pp. 1458-1463). IEEE.

4.1 Introduction

Multi-dimensional datasets are common in a wide range of applications such as chemometrics [Acar et al., 2011a], signal processing [Cichocki et al., 2015], machine learning [Globerson et al., 2007] and data mining [Kolda et al., 2005]. A natural choice for representing such data is a tensor, which is a multi-way array that can preserve the complex relationships among different dimensions. Tensor decomposition is a fundamental technique for analyzing tensors, and it has been extensively studied and widely applied for varying tasks, including subspace learning in computer vision [Hu et al., 2011], community detection in time-evolving networks [Anandkumar et al., 2014], and feature extraction for spatial-temporal time series [Ma et al., 2016]. However, existing tensor decomposition techniques are usually designed for dense and static tensors, which makes them less suitable when the data is highly sparse and dynamically changing over time.

Many real-world tensors are very large and have high sparsity. Thus, compared to their overall size, the number of non-zero entries is small. As a consequence, it is difficult to apply existing tensor decomposition techniques, since these have often been designed for dense tensors, and thus have poor efficiency and scalability when applied to sparse scenarios. Moreover, sparse tensors are often not static. Instead, they are often changing over time, as large volumes of new data is generated and added to the the existing tensor. One typical type of dynamic update is appending new data along a specific dimension such as time, with the other dimensions remaining unchanged. An example of such a scenario is a time-evolving social network, as shown in Figure 4.1. The network can be represented by a *user* $\times$ *user* $\times$ *time* tensor and each slice at its time dimension is a collection of user interactions such as tagging a friend in a photo or visiting a friend's homepage. Overall, this tensor is highly sparse by its nature, and rapidly growing as new data (slices) are added. We refer to such tensors as *online sparse tensors*.

The research problem we address in this chapter is how to efficiently decompose online sparse tensors. We are particularly interested in CP de-

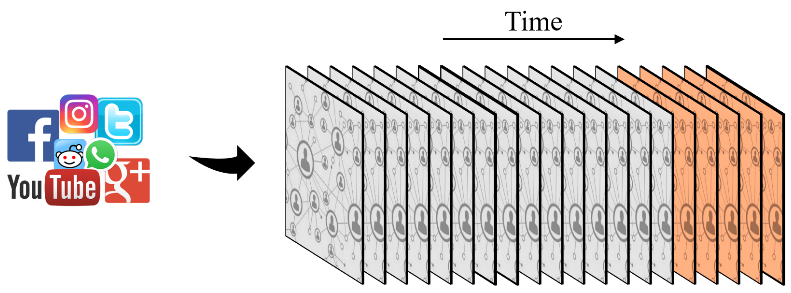


Figure 4.1: An example of online sparse tensors

composition, since it is one of the most popular tensor decomposition techniques having numerous applications across different domains [Kolda and Bader, 2009]. Specifically, given the existing CP decomposition of a sparse tensor and a batch of new sparse data slices, which get added to the tensor's time mode, *we would like to efficiently obtain the new CP decomposition of the newly formed tensor without computing it from scratch*.

To the best of our knowledge, this is an unresolved problem and existing approaches are not suitable for decomposing online sparse tensors. ALS has been widely considered as the workhorse for CP decomposition [Kolda and Bader, 2009] due to its simplicity and ease of implementation. However, it is infeasible for decomposing an online large-scale sparse tensor, due to poor runtime efficiency. In addition, batch methods like ALS require availability of the full data, while the size of an online tensor that grows over time is potentially unbounded, so we may not be able to fit it into the memory. Even though distributed and parallel algorithms [Choi and Vishwanathan, 2014, Jeon et al., 2015, Kang et al., 2012] may be used to accelerate ALS, the number of non-zeros in the new data can be too small, so it is wasteful to employ computational resources to decompose the full tensor from scratch. Lastly, there are existing approaches, including our OnlineCP algorithm proposed in Chapter 3, that have been designed for incremental, online decompositions [Nion and Sidiropoulos, 2009], but these are designed for online *dense* tensors and their complexities are usually linear w.r.t. the size of new data. This means that they require the same amount of time and space for processing both dense and sparse online tensors, and no optimization is employed with respect to sparsity. This significantly limits their applicability to online sparse tensors which can be very large, but contain few non-zero elements.

Overall, due to their poor efficiency and scalability, existing methods are not well suited for decomposing online sparse tensors. To address this gap, we propose a new algorithm, OnlineSCP. The contributions of our work are as follows:

- We propose a new algorithm having linear complexity to the number of non-zeros in the new data, for tracking the CP decompositions of sparse dynamic tensors with slice-wise updates.
- Via experiments on nine real-world datasets, our method demonstrates high decomposition quality, as well as significant efficiency improvements in both time and memory usage. Empirical analysis on real and synthetic datasets shows that our method is considerably more scalable than state-of-the-art techniques.

4.2 Related Work

Tensors are a generalization of matrices and CP decomposition is a powerful tool for data simplification, feature extraction and knowledge discovery on tensors. While much tensor-based work can be found in the literature, few studies have been conducted on decomposing online sparse tensors. In this section we review relevant literature in this area.

In order to decompose a large-scale sparse tensor efficiently, [Bader and Kolda, 2007] propose an algorithm based on ALS, tailoring it to sparse tensors, with support from special data structures and customized tensor-related operations. Following the same idea, [Smith et al., 2015] propose a new hierarchical, fiber-centric data structure called a Compressed Sparse Fiber (CSF). Compared to the coordinate (COO) format used in [Bader and Kolda, 2007], CSF is more memory-efficient and much easier to parallelize. [Li et al., 2017] also apply CSF as the storage format for sparse tensors, while a variant of CSF (vCSF) was used for recording intermediate results, which leads to more memory-savings and speedup. In addition, with advances in distributed computing, techniques like MapReduce have also been used to further speed up ALS for large-scale sparse tensors[Choi and Vishwanathan, 2014,Jeon et al., 2015,Kang et al., 2012]. Inspired by the success of parallel tensor decomposition algorithms, GPUs have been applied for accelerating sparse tensor computations [Liu et al., 2017].

In terms of online tensor decomposition, limited research can be found for online CP decompositions. An early exploration is [Nion and Sidiropoulos, 2009], introducing two adaptive algorithms that specifically focus on CP decomposition: Simultaneous Diagonalization Tracking (SDT), which incrementally tracks the SVD of the unfolded tensor; and Recursive Least Squares Tracking (RLST), which recursively updates the decomposition factors by minimizing the mean squared error. However, the limitation of this work is that it can only be applied to 3^{rd}-order tensors. Our earlier work (presented in chapter 3) propose a general approach that incrementally tracks the CP decompositions of online tensors with arbitrary dimensions. However, both [Nion and Sidiropoulos, 2009] and our work are specifically designed for dense online tensors, which means that their efficiency quickly drops to an unacceptable level and may potentially run out of memory for sparse online tensors having large non-temporal modes. A more recent technique proposed by Gujral *et al.* [Gujral et al., 2018a] can operate with both dense and sparse online tensors via sub-sampling, though multiple repetitions are required for a stable result[1].

4.3 OnlineSCP Algorithm

This section introduces our approach, OnlineSCP, an algorithm to track the CP decomposition of a sparse dynamic tensor with slice-wise update. We first discuss the main idea of our algorithm, followed by key improvements

[1]We cannot include this work in experiment due to technique issues. While we would like to highlight that the major improvement of this work is gained by parallel computing, while our method aims to address the efficiency issue from an algorithmic point of view.

that have been applied and a brief complexity analysis and a comparison to state-of-the-art methods.

Formally speaking, the research question we solve is defined as follows: given *(i)* an existing CP decomposition $[\![\widetilde{\mathbf{A}}^{(1)}, \ldots, \widetilde{\mathbf{A}}^{(N)}]\!]$ of R components that approximates a sparse tensor $\widetilde{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times t}$ at time t, *(ii)* a new incoming sparse tensor $\Delta\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times \Delta t}$ that is appended to $\widetilde{\mathcal{X}}$ at its last mode and forms a new tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times (t+\Delta t)}$, where $\Delta t \ll t$. The objective is to find the CP decomposition $[\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$ of $\mathcal{X}$.

4.3.1 The Principle of OnlineSCP

To address the problem, our method follows the same alternating update schema as ALS, such that only one loading matrix is updated at one time by fixing all others. Specifically, we update the time mode first, then move on to the rest of the non-temporal modes as

$$\mathbf{A}^{(N)} \rightarrow \mathbf{A}^{(1)} \rightarrow \mathbf{A}^{(2)} \cdots \rightarrow \mathbf{A}^{(N-1)}.$$

In order to take advantage of the fact that the tensor is only growing at its time mode, similar to existing online works [Nion and Sidiropoulos, 2009], the main assumption of our method is that the new incoming data, $\Delta\mathcal{X}$, will not have significant impact on the existing model, but only contribute to the update of its corresponding local features. By this, we mean when new data arrives, the first t rows in the loading matrix of the temporal mode, $\mathbf{A}^{(N)}$, will remain unchanged; while the loading matrices for non-temporal modes, $\mathbf{A}^{(1)}, \ldots \mathbf{A}^{(N-1)}$, will receive a minor update based on their existing values.

At a high level, as presented in Algorithms 4.1 and 4.2, our algorithm contains two procedures. Before the start of the learning phase, it initializes two small *helper* matrices for storing historical information by using the initial tensor and its decomposition (details in Algorithm 4.1). After that, in the online learning phase, the new incoming tensors can be efficiently processed by an incremental update schema (details in Algorithm 4.2).

Algorithm 4.1: Initialization Procedure of OnlineSCP

```
Input: initial data $\mathcal{X}_0$, its decomposition $[\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$
Output: helper matrices $\mathbf{Q}$ and $\mathbf{U}^{(N)}$
1  $\mathbf{U}^{(N)} \leftarrow \mathbf{A}^{(N)\top}\mathbf{A}^{(N)}$
2  $\mathbf{Q} \leftarrow \mathbf{U}^{(N)}$
3  for $n \leftarrow 1$ to $N-1$ do
4      $\mathbf{Q} \leftarrow \mathbf{Q} \circledast (\mathbf{A}^{(n)\top}\mathbf{A}^{(n)})$
5  end
```

Algorithm 4.2: Update Procedure of OnlineSCP

```
Input: loading matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N-1)}$, helper matrices $\mathbf{Q}$ and
       $\mathbf{U}^{(N)}$, new data $\Delta\mathcal{X}$
Output: updated loading matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N-1)}$, updated helper
        matrices $\mathbf{Q}$ and $\mathbf{U}^{(N)}$, coefficient of new data $\Delta\mathbf{A}^{(N)}$
1  $\widetilde{\mathbf{Q}} \leftarrow \mathbf{Q}$
   // process temporal mode
2  $\mathbf{Q}^{(N)} \leftarrow \mathbf{Q} \oslash \mathbf{U}^{(N)}$
3  $\Delta\mathbf{P}^{(N)} \leftarrow$ MTTKRP by Algorithm 2.2
4  $\Delta\mathbf{A}^{(N)} \leftarrow \Delta\mathbf{P}^{(N)}(\mathbf{Q}^{(N)})^{-1}$
   // update helper matrices
5  $\mathbf{U}^{(N)} \leftarrow \mathbf{U}^{(N)} + (\Delta\mathbf{A}^{(N)\top}\Delta\mathbf{A}^{(N)})$
6  $\mathbf{Q} \leftarrow \mathbf{Q}^{(N)} \circledast \mathbf{U}^{(N)}$
   // process non-temporal mode
7  for $n \leftarrow 1$ to $N-1$ do
8      $\widetilde{\mathbf{A}}^{(n)} \leftarrow \mathbf{A}^{(n)}$
9      $\mathbf{U}^{(n)} \leftarrow \mathbf{A}^{(n)\top}\mathbf{A}^{(n)}$
10     $\mathbf{Q}^{(n)} \leftarrow \mathbf{Q} \oslash \mathbf{U}^{(n)}$, $\widetilde{\mathbf{Q}}^{(n)} \leftarrow \widetilde{\mathbf{Q}} \oslash \mathbf{U}^{(n)}$
11     $\Delta\mathbf{P}^{(n)} \leftarrow$ MTTKRP by Algorithm 1
12     $\mathbf{A}^{(n)} \leftarrow (\mathbf{A}^{(n)}\widetilde{\mathbf{Q}}^{(n)} + \Delta\mathbf{P}^{(n)})(\mathbf{Q}^{(n)})^{-1}$
       // update helper matrices
13     $\mathbf{Q} \leftarrow \mathbf{Q}^{(n)} \circledast (\mathbf{A}^{(n)\top}\mathbf{A}^{(n)})$
14     $\widetilde{\mathbf{Q}} \leftarrow \widetilde{\mathbf{Q}}^{(n)} \circledast (\widetilde{\mathbf{A}}^{(n)\top}\mathbf{A}^{(n)})$
15 end
```

Compared to ALS, the major speedup gained by our approach comes from several aspects: (i) only a small fraction of new values are added to the loading matrix of the temporal mode while the remainder is kept unchanged (as discussed in §4.3.2); (ii) for non-temporal modes, we break down the costly MTTKRP (discussed in §2.3.2 calculation into historical and new data parts, the historical one is avoided by using helper matrices and the new data one is efficiently obtained by Algorithm 1 (as discussed in §4.3.3); (iii) the helper matrices can also be incrementally updated at a small cost (as discussed in §4.3.4).

4.3.2 Updating the Temporal Mode

Specifically, the temporal loading matrix, $\mathbf{A}^{(N)}$, is a $(t + \Delta t) \times R$ matrix as

$$\mathbf{A}^{(N)} \leftarrow \begin{bmatrix} \widetilde{\mathbf{A}}^{(N)} \\ \Delta\mathbf{A}^{(N)} \end{bmatrix},$$

where its first t rows is $\widetilde{\mathbf{A}}^{(N)} \in \mathbb{R}^{t \times R}$ and $\Delta\mathbf{A}^{(N)}$ is a small matrix of size $\Delta t \times R$. $\Delta\mathbf{A}^{(N)}$ can be easily obtained by projecting $\Delta\mathfrak{X}$ to the last mode via other non-temporal loading matrices as

$$\begin{aligned}\Delta\mathbf{A}^{(N)} &= \Delta\mathbf{X}_{(N)}((\odot_{i=1}^{N-1}\mathbf{A}^{(i)})^\top)^\dagger \\ &= \frac{\Delta\mathbf{X}_{(N)}(\odot_{i=1}^{N-1}\mathbf{A}^{(i)})}{\circledast_{i=1}^{N-1}(\mathbf{A}^{(i)\top}\mathbf{A}^{(i)})}.\end{aligned}$$

Let $\mathbf{U}^{(N)} = \mathbf{A}^{(N)\top}\mathbf{A}^{(N)}$, $\mathbf{Q} = \circledast_{i=1}^{N}(\mathbf{A}^{(i)\top}\mathbf{A}^{(i)})$, and $\Delta\mathbf{P}^{(N)} = \Delta\mathbf{X}_{(N)}(\odot_{i=1}^{N-1}\mathbf{A}^{(i)})$, recall that there is no loading matrices has been updated so far, we can rewrite the above equation with helper matrices $\mathbf{U}^{(N)}$ and $\mathbf{Q}$ as

$$\Delta\mathbf{A}^{(N)} \leftarrow \frac{\Delta\mathbf{P}^{(N)}}{\mathbf{Q} \oslash \mathbf{U}^{(N)}}, \tag{4.1}$$

where the MTTKRP, $\Delta\mathbf{P}^{(N)}$, can be efficiently calculated by Algorithm 2.2 [Bader and Kolda, 2007] at linear complexity to the number of non-zeros in

$\Delta\mathcal{X}$, which we refer to as $|\Delta\Omega^+|$.

4.3.3 Updating Non-Temporal Modes

Without loss of generality, here we show how to derive the incremental update rule for loading matrix at the first mode, $\mathbf{A}^{(1)}$. By using helper matrix $\mathbf{Q}$ defined as above, the update given by the typical ALS is

$$\mathbf{A}^{(1)} \leftarrow \frac{\mathbf{X}_{(1)}(\odot_{i=2}^{N}\mathbf{A}^{(i)})}{\circledast_{i=2}^{N}(\mathbf{A}^{(i)\top}\mathbf{A}^{(i)})} = \frac{\mathbf{X}_{(1)}(\odot_{i=2}^{N}\mathbf{A}^{(i)})}{\mathbf{Q}\oslash\mathbf{U}^{(1)}}. \tag{4.2}$$

Even though the MTTKRP operation, $\mathbf{X}_{(1)}(\odot_{i=2}^{N}\mathbf{A}^{(i)})$, can be accelerated by Algorithm 1 given that $\mathcal{X}$ is sparse, such cost is still not accepatable since $\mathcal{X}$ is potentially a large-scale tensor and the number of non-zeros in $\mathcal{X}$ will keep growing with the increase of time. As a matter of fact, by noticing that $\mathcal{X}$ is an online tensor that new data is only appended at its last mode, there are some patterns can be found in its mode-n unfolding and the corresponding Khatri-Rao product. As a result, we make use of such patterns to derive an efficient update rule as follows.

Recall that $\mathbf{A}^{(N)} = [\widetilde{\mathbf{A}}^{(N)}; \Delta\mathbf{A}^{(N)}]$ and let $\mathbf{B}^{(1)} = \odot_{i=2}^{N-1}\mathbf{A}^{(i)}$, Eq. (4.2) can be rewritten as

$$\begin{aligned}
\mathbf{A}^{(1)} \leftarrow & \frac{\left[\widetilde{\mathbf{X}}_{(1)}, \Delta\mathbf{X}_{(1)}\right]\left(\begin{bmatrix}\widetilde{\mathbf{A}}^{(N)} \\ \Delta\mathbf{A}^{(N)}\end{bmatrix}\odot\mathbf{B}^{(1)}\right)}{\mathbf{Q}\oslash\mathbf{U}^{(1)}} \\
= & \frac{\widetilde{\mathbf{X}}_{(1)}(\widetilde{\mathbf{A}}^{(N)}\odot\mathbf{B}^{(1)}) + \Delta\mathbf{X}_{(1)}(\Delta\mathbf{A}^{(N)}\odot\mathbf{B}^{(1)})}{\mathbf{Q}\oslash\mathbf{U}^{(1)}} \\
= & \frac{\widetilde{\mathbf{P}}^{(1)} + \Delta\mathbf{P}^{(1)}}{\mathbf{Q}\oslash\mathbf{U}^{(1)}}.
\end{aligned}$$

In this way, the MTTKRP is divided into two parts: the one that is related to the historical data, $\widetilde{\mathbf{P}}^{(1)}$; and the other that is only depending on the new incoming data, $\Delta\mathbf{P}^{(1)}$. In fact, since $[\![\widetilde{\mathbf{A}}^{(1)}, \ldots, \widetilde{\mathbf{A}}^{(N)}]\!]$ is the existing decomposition of $\widetilde{\mathcal{X}}$, we have

$$\widetilde{\mathbf{X}}_{(1)} \approx \widetilde{\mathbf{A}}^{(1)}(\odot_{i=2}^{N}\widetilde{\mathbf{A}}^{(i)})^\top = \widetilde{\mathbf{A}}^{(1)}(\widetilde{\mathbf{A}}^{(N)} \odot \widetilde{\mathbf{B}}^{(1)})^\top, \tag{4.3}$$

where $\widetilde{\mathbf{B}}^{(1)} = \odot_{i=2}^{N-1}\widetilde{\mathbf{A}}^{(i)}$. Based on this, $\widetilde{\mathbf{X}}_{(1)}$ in $\widetilde{\mathbf{P}}^{(1)}$ can be replaced by Eq. (4.3) and the final update rule for $\mathbf{A}^{(1)}$ is

$$\begin{aligned}
\mathbf{A}^{(1)} &\leftarrow \frac{\widetilde{\mathbf{X}}_{(1)}(\widetilde{\mathbf{A}}^{(N)} \odot \mathbf{B}^{(1)}) + \Delta\mathbf{P}^{(1)}}{\mathbf{Q} \oslash \mathbf{U}^{(1)}} \\
&\approx \frac{\widetilde{\mathbf{A}}^{(1)}(\widetilde{\mathbf{A}}^{(N)} \odot \widetilde{\mathbf{B}}^{(1)})^\top(\widetilde{\mathbf{A}}^{(N)} \odot \mathbf{B}^{(1)}) + \Delta\mathbf{P}^{(1)}}{\mathbf{Q} \oslash \mathbf{U}^{(1)}} \\
&= \frac{\widetilde{\mathbf{A}}^{(1)}(\widetilde{\mathbf{A}}^{(N)\top}\widetilde{\mathbf{A}}^{(N)}) \circledast (\widetilde{\mathbf{B}}^{(1)\top}\mathbf{B}^{(1)}) + \Delta\mathbf{P}^{(1)}}{\mathbf{Q} \oslash \mathbf{U}^{(1)}} \\
&= \widetilde{\mathbf{A}}^{(1)}\frac{\widetilde{\mathbf{Q}} \oslash \mathbf{U}^{(1)}}{\mathbf{Q} \oslash \mathbf{U}^{(1)}} + \frac{\Delta\mathbf{P}^{(1)}}{\mathbf{Q} \oslash \mathbf{U}^{(1)}},
\end{aligned} \tag{4.4}$$

where $\widetilde{\mathbf{Q}} = (\widetilde{\mathbf{A}}^{(N)\top}\widetilde{\mathbf{A}}^{(N)}) \circledast (\circledast_{i=1}^{N-1}(\widetilde{\mathbf{A}}^{(i)\top}\mathbf{A}^{(i)}))$.

Overall, the above update rule significantly reduces the computational cost by limiting the expensive MTTKRP operation to the new data only. We can interpret such update schema as follows: *the new loading matrix is a weighted combination of existing loading and a hyper loading learned from the new data. The weight between them is determined by the ratio of information contains in the historical data w.r.t. the full data.*

4.3.4 Incremental Update of Q and $\widetilde{\mathbf{Q}}$

As mentioned before, by definition we have

$$\begin{aligned}
\mathbf{Q} &= (\mathbf{A}^{(1)\top}\mathbf{A}^{(1)}) \cdots \circledast (\mathbf{A}^{(n)\top}\mathbf{A}^{(n)}) \circledast \cdots (\mathbf{A}^{(N)\top}\mathbf{A}^{(N)}), \\
\widetilde{\mathbf{Q}} &= (\widetilde{\mathbf{A}}^{(1)\top}\mathbf{A}^{(1)}) \cdots \circledast (\widetilde{\mathbf{A}}^{(n)\top}\mathbf{A}^{(n)}) \circledast \cdots (\widetilde{\mathbf{A}}^{(N)\top}\widetilde{\mathbf{A}}^{(N)}).
\end{aligned}$$

Their values are gradually changing with the updating of loading matrices. However, the main difference between the $\mathbf{Q}$ values before and after updating $\mathbf{A}^{(n)}$ is just the ${\mathbf{A}^{(n)}}^\top \mathbf{A}^{(n)}$ term. As a result, we do not need to calculate the $\mathbf{Q}$ values from scratch for each update. Instead, we initialize both $\mathbf{Q}$ and $\widetilde{\mathbf{Q}}$ as $\circledast_{i=1}^{N}({\widetilde{\mathbf{A}}^{(i)}}^\top \widetilde{\mathbf{A}}^{(i)})$. After processing the time mode, $\widetilde{\mathbf{Q}}$ remains unchanged, while $\mathbf{Q}$ can be updated as

$$\mathbf{Q} \leftarrow \mathbf{Q} \oslash ({\widetilde{\mathbf{A}}^{(N)}}^\top \widetilde{\mathbf{A}}^{(N)}) \circledast ({\mathbf{A}^{(N)}}^\top \mathbf{A}^{(N)}).$$

Similarly, the update after processing the n-th non-temporal mode is

$$\mathbf{Q} \leftarrow \mathbf{Q} \oslash ({\widetilde{\mathbf{A}}^{(N)}}^\top \widetilde{\mathbf{A}}^{(N)}) \circledast ({\mathbf{A}^{(N)}}^\top \mathbf{A}^{(N)}),$$
$$\widetilde{\mathbf{Q}} \leftarrow \widetilde{\mathbf{Q}} \oslash ({\widetilde{\mathbf{A}}^{(N)}}^\top \widetilde{\mathbf{A}}^{(N)}) \circledast ({\widetilde{\mathbf{A}}^{(N)}}^\top \mathbf{A}^{(N)}).$$

After processing all modes, the final $\mathbf{Q}$ is stored and used to initialize $\mathbf{Q}$ and $\widetilde{\mathbf{Q}}$ for the next batch of new data.

Additionally, since $\mathfrak{X}$ is an online tensor and the length of the time mode t can be potentially quite large, we choose to avoid directly computing ${\widetilde{\mathbf{A}}^{(N)}}^\top \widetilde{\mathbf{A}}^{(N)}$ and ${\mathbf{A}^{(N)}}^\top \mathbf{A}^{(N)}$ by storing ${\widetilde{\mathbf{A}}^{(N)}}^\top \widetilde{\mathbf{A}}^{(N)}$ into a small $R \times R$ matrix $\mathbf{U}^{(N)}$, and ${\mathbf{A}^{(N)}}^\top \mathbf{A}^{(N)}$ can be easily obtained as

$$\mathbf{U}^{(N)} \leftarrow \mathbf{U}^{(N)} + ({\Delta\mathbf{A}^{(N)}}^\top \Delta\mathbf{A}^{(N)}).$$

4.3.5 Complexity Analysis

Let R be the decomposition rank, N be the order of tensor, t be the time length of existing data, Δt be the time length of new data, $|\Omega^+|$ be the number of non-zeros in the full data, $|\Delta\Omega^+|$ be the number of non-zeros in the new tensor, $S = \prod_{i=1}^{N-1} I_i$, and $J = \sum_{i=1}^{N-1} I_i$, where I_i is the length of the i-th mode.

To process a new chunk of data, the overall time complexity for our algorithm is $\mathcal{O}((J + \Delta t)R^2 + |\Delta\Omega^+|NR)$, where $(J + \Delta t)R^2$ is corresponding to the $({\mathbf{A}^{(n)}}^\top \mathbf{A}^{(n)})$-like calculations (line 5, 9, 13, 14 in Algorithm 4.2) and the actual update for loading matrices (line 4, 12 in Algorithm 4.2); $|\Delta\Omega^+|NR$ is cost for MTTKRP operation (line 3, 11 in Algorithm 4.2).

Table 4.1: Complexity comparison between OnlineSCP and existing methods.

	Time	Memory
OnlineSCP	$\mathcal{O}((J+\Delta t)R^2 + \lvert\Delta\Omega^+\rvert NR)$	$\mathcal{O}((J+\Delta t)R + \lvert\Delta\Omega^+\rvert)$
ALS	$\mathcal{O}((J+t+\Delta t)R^2 + \lvert\Omega^+\rvert NR)$	$\mathcal{O}((J+t+\Delta t)R + \lvert\Omega^+\rvert)$
OnlineCP	$\mathcal{O}((J+\Delta t)R^2 + NRS\Delta t)$	$\mathcal{O}((J+\Delta t)R + SR\Delta t)$
SDT	$\mathcal{O}((S+\Delta t)R^2)$	$\mathcal{O}((J+S+t)R + SR\Delta t)$
RLST	$\mathcal{O}(SR^2)$	$\mathcal{O}((S+J+\Delta t)R + SR\Delta t)$

In terms of space consumption, since the update of the time mode is independent to historical data, our method only needs to store the loading matrices for non-temporal modes and two $R \times R$ helper matrices in memory. In addition, extra $\mathcal{O}(|\Delta\Omega^+|)$ memory needs to be used for the MTTKRP calculation. As a result, the overall space cost is $\mathcal{O}((J+\Delta t)R + |\Delta\Omega^+|)$.

We summarize the complexity of OnlineSCP and make a comparison to state-of-the-art methods in Table 4.1. It should be noted that the complexitie of SDT and RLST are measured based on the exponential window strategy [Nion and Sidiropoulos, 2009], which considers all updated-to-date data in a single window assessing the importance of data slices at different timestamps using a forgetting factor λ. In addition, as they only work on 3^{rd}-order tensors, when other methods are compared to them, N should be set to 3. Another remark is that the complexity of ALS is based on one iteration only, in reality it usually takes a few iterations until convergence.

4.4 Empirical Analysis

In this section, we compare the performance of our proposed OnlineSCP algorithm with state-of-the-art techniques. We first examine their effectiveness and efficiency, in terms of both time and space, on nine real-world datasets. In addition, we make further investigation on the scalability of our method and baselines using several synthetic datasets.

Table 4.2: Detail of real-world datasets
(K: thousands, M: millions)

Datasets	Description	Size	*nnz*[*]	Density	Batch Size
Facebook-Links	user × user × day	64K × 64K × 886	671K	2×10^{-7}	4
Facebook-Wall	wall owner × poster × day	47K × 47K × 2K	738K	2×10^{-7}	8
MovieLens	user × movie × time	6K × 4K × 1K	1M	4×10^{-5}	5
LastFM	user × artist × time	1K × 1K × 168	3M	2×10^{-2}	1
NIPS	paper × author × year × word	2K × 3K × 17 × 14K	3M	2×10^{-6}	70
Youtube	user × user × day	3M × 3M × 226	18M	8×10^{-9}	1
Enron	sender × receiver × word × day	6K × 6K × 244K × 1K	54M	5×10^{-9}	6
NELL-2	entity × relation × entity	12K × 9K × 30K	77M	2×10^{-5}	144
NELL-1	entity × relation × entity	3M × 2M × 25M	144M	9×10^{-13}	127,476

[*] number of non-zeros

4.4.1 Experiment Specifications

Datasets

Nine real-world datasets of varying characteristics have been used in our experiments and their detail can be found in Table 4.2. Facebook [Viswanath et al., 2009] and Youtube [Mislove, 2009] are time-evolving social network data obtained from KONECT [Kunegis, 2013]. Both MovieLens [Paranjape et al., 2017] and LastFM [Herrada, 2009] contain user rating data. The ratings in MovieLens are explicit, scaling from 1 to 5, whereas LastFM contains implicit ratings that are represented by the number of times a user listened a particular artist's songs for a given time interval. The rest of the datasets are publicly available on FROSTT [Smith et al., 2017]. NELL-1 and NELL-2 come from Never Ending Language Learning (NELL) project [Carlson et al., 2010] and represent (noun, verb, noun) triples. NIPS [Globerson et al., 2007] is a paper authorship network data and Enron [Shetty and Adibi, 2004] records email transactions within senior managers in Enron. Both of them are 4^{th}-order tensors.

Baselines

In our experiments, four baselines have been selected for performance comparison.

(i) ALS [Kolda and Bader, 2009]: an implementation of the ALS algorithm for sparse tensors provided by Tensor Toolbox [Bader et al., 2015]. Since it is a batch method, to make it work with online tensors, the CP decomposition of the last time step is used as the initialization for decomposing the current tensor.

(ii) OnlineCP (described in Chapter 3): an incremental ALS-like algorithm, which can track the decompositions of both 3^{rd}-order and higher-order online tensors.

(iii) SDT [Nion and Sidiropoulos, 2009]: an adaptive algorithm based on incrementally tracking the SVD of the unfolded tensor on the time mode.

(iv) RLST: another online approach proposed in [Nion and Sidiropoulos, 2009]. Instead of tracking the SVD, recursive updates are performed to minimize the mean squared error on new data.

Evaluation Metrics

The performance of each method is demonstrated by its effectiveness and efficiency.

In terms of effectiveness, we measure the *fitness* of each algorithm as

$$\mathit{fitness} \triangleq \left(1 - \frac{\left\|\hat{\mathcal{X}} - \mathcal{X}\right\|}{\left\|\mathcal{X}\right\|}\right),$$

where $\mathcal{X}$ is the original data, $\hat{\mathcal{X}}$ is the estimation and $\left\|\bullet\right\|$ denotes the Frobenius norm.

In order to validate the efficiency performance of an algorithm, both the *running time* and *memory usage* for processing one batch of new data are measured.

Since ALS is the most popular method for CP decomposition and in order to better interpret the results, we report the *relative performance* of an online method to ALS over all three metrics as

$$\begin{aligned} relative\ fitness &\triangleq \frac{fitness(baseline)}{fitness(ALS)}, \\ speedup &\triangleq \frac{time(ALS)}{time_(baseline)}, \\ relative\ memory &\triangleq \frac{memory(baseline)}{memory(ALS)}. \end{aligned}$$

Experimental Setup

Given a dataset, the first 50% data along the last mode is decomposed by ALS and this corresponding CP decomposition is used for initializing all methods. After that, the second half is further divided into 100 batches and each of them is sequentially appended to the existing data. All methods process one batch of data at a time. After learning a new batch, the updated decompositions of all methods are used to calculate the fitness, along with the time and memory consumption for this batch.

With respect to parameter settings, the decomposition rank R is fixed to 5 for all datasets, due to the poor efficiency of baselines. For the ALS algorithm used in the initialization stage, the tolerance ϵ is set to 1×10^{-8} and the maximum number of iterations is set to 100, to ensure quality starting points for all methods. In the online learning phase, ϵ of ALS is altered to 1×10^{-4} and we enforce it to run one iteration at one timestamp, for a fair comparison to other methods. For SDT and RLST, exponential window strategy is used with a forgetting factor $\lambda = 1$, as all other methods treat all data as equally important. In addition, it should be noted that the original implementation of SDT and RLST can only handle one slice of data at a time, so we modified them by executing a for loop to process a batch. Lastly, no parameters need to be tuned for our method and OnlineCP.

The experiments are conducted on Spartan [Lafayette and Wiebelt, 2017], a research platform with multiple computing nodes and each of them has 12 CPU cores and 251 GB RAM. Due to the fact that Spartan is a highly utilized system, we limit the computing resources for each algorithm-dataset pair as 4 CPU cores, 32 GB memory and 12-hour maximum running time. The whole experiment is replicated 5 times with Matlab and the final results are averaged over these repetitions.

4.4.2 Experimental Results

The experimental results on real-world datasets can be found in Table 4.3, 4.4 and 4.5.

Results on Effectiveness

Among all online methods, our proposal, OnlineSCP, is the only method that is able to produce results for all datasets, given the limited computa-

Table 4.3: Mean relative **fitness** to ALS over all batches. The higher the better (boldface means the best results).

Dataset	OnlineCP	SDT	RLST	OnlineSCP
Facebook-Links	n/a[*]	n/a	n/a	**1.00**
Facebook-Wall	n/a	n/a	n/a	**0.92**
MovieLens	**1.00**	0.31	**1.00**	**1.00**
LastFM	**0.91**	0.22	0.17	**0.91**
NIPS	n/a	n/a	n/a	**0.96**
Youtube	n/a	n/a	n/a	**1.00**
Enron	n/a	n/a	n/a	**0.93**
NELL-2	0.97	n/a	n/a	**0.98**
NELL-1	n/a	n/a	n/a	**0.84**

[*] n/a means the method is failed since it is not applicable/running out of memory/cannot finish within 12 hours

tional resources as stated before. In contrast, SDT and RLST only manage to process MovieLens and LastFM, which are two relatively small datasets in terms of slice size (6K $\times$ 6K for MovieLens and 1K $\times$ 1K for LastFM). OnlineCP can process one more dataset compared to them, NELL-2, which has a slice size of 108M. Ideally OnlineCP should also work on NIPS dataset as one slice of NIPS data is slightly smaller than that of NELL-2. However, OnlineCP has a specifically designed procedure to speedup calculations on higher-order tensors by costing more memory, hence it fails on this 4^{th}-order tensor.

Compared to existing online methods, our algorithm is able to produce the highest quality decompositions on-the-fly. The fitness of OnlineSCP is comparable to ALS being more than 90% as good in most cases. However, the performance of SDT and RLST is considerably worse on this criteria. Specifically, compared to ALS, the relative fitness of SDT is under one third for both MovieLens and LastFM, and RLST can only reach 17% relative fitness on LastFM. The relative fitness of OnlineCP is close to ours on MovieLens and LastFM, but 1% lower on NELL-2 dataset.

Table 4.4: Mean relative **speedup** to ALS over all batches. The higher the better (boldface means the best results).

Dataset	OnlineCP	SDT	RLST	OnlineSCP
Facebook-Links	n/a	n/a	n/a	**3.90**
Facebook-Wall	n/a	n/a	n/a	**5.65**
MovieLens	2.03	0.05	0.02	**42.06**
LastFM	70.68	12.41	6.53	**74.83**
NIPS	n/a	n/a	n/a	**102.08**
Youtube	n/a	n/a	n/a	**3.14**
Enron	n/a	n/a	n/a	**160.24**
NELL-2	30.12	n/a	n/a	**258.50**
NELL-1	n/a	n/a	n/a	**27.81**

Results on Time Efficiency

Lack of optimization on sparsity is the main contributing factor to the poor time efficiency of existing online methods on the MovieLens dataset, e.g., OnlineCP only speeds up ALS by 2 times, and SDT and RLST are much slower than ALS, while our method significantly reduces the time consumption by more than 40 times. Similarly, huge time efficiency difference can be observed between OnlineCP and our approach on NELL-2 dataset, where our method is more than 250 times faster than ALS, whilst the speedup of OnlineCP is only about 30 times. On LastFM dataset, SDT and RLST perform slightly better with around 12 and 7 times faster than ALS, respectively. However, they have been significantly outperformed by OnlineCP (x70 speedup) and our approach (x75 speedup).

Results on Space Efficiency

As shown in the complexity comparison (Table 4.1, all existing online methods have a space complexity that is linear w.r.t. the slice size, which can be much greater than necessary if the data is highly sparse. In fact, this is the main reason why they fail on most of the real-world datasets given the time and memory limits. Sometimes their memory usages can be even worse

Table 4.5: Mean relative **memory usage** to ALS over all batches.The lower the better (boldface means the best results).

Dataset	OnlineCP	SDT	RLST	OnlineSCP
Facebook-Links	n/a	n/a	n/a	**0.35**
Facebook-Wall	n/a	n/a	n/a	**0.32**
MovieLens	17.41	60.87	46.38	**0.02**
LastFM	0.36	1.24	0.94	**0.01**
NIPS	n/a	n/a	n/a	**0.01**
Youtube	n/a	n/a	n/a	**0.38**
Enron	n/a	n/a	n/a	**0.04**
NELL-2	1.49	n/a	n/a	**0.01**
NELL-1	n/a	n/a	n/a	**0.06**

than ALS, which is a batch method that stores all information with sparse indexing. For example, on MovieLens, the space usage of OnlineCP, SDT and RLST is 17, 60 and 46 times to the memory used by ALS. In contrast, we only use 2% of memory compared to ALS. In addition,

Discussion on the Efficiency Variance of OnlineSCP

We can see that the efficiency performance of our OnlineSCP method varies from dataset to dataset. For example, compared to ALS, the speedups of OnlineSCP on the social network datasets, Facebook-Links, Facebook-Wall and Youtube are only 3.9, 5.65 and 3.14 times, respectively; while it can improve the time consumption on NIPS, Enron and NELL-2 by more than 100 times. Similar patterns can be found for memory usage as well, where for social network data, around one third of memory used by ALS is needed for OnlineSCP, while for others the relative memory can be as less as 1% w.r.t. ALS.

One can understand this behavior based on the complexity analysis. Specifically, the complexity of OnlineSCP, both in time and space, consists of two parts: 1) one part related to the overall length of the non-temporal modes, J, 2) and the other part that depends on the number of non-zeros in the new data, $|\Delta\Omega^{+}|$. That is, if J is a relatively large number compared to

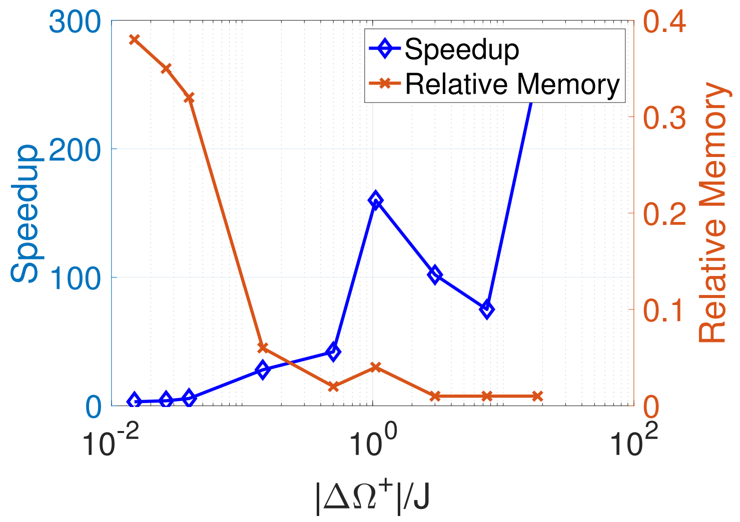
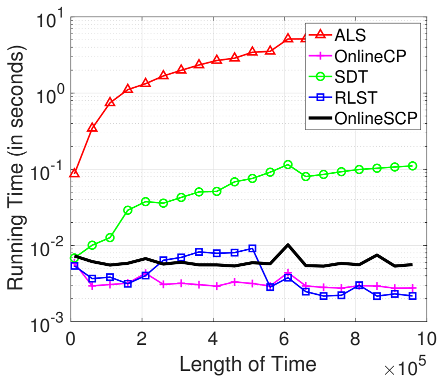
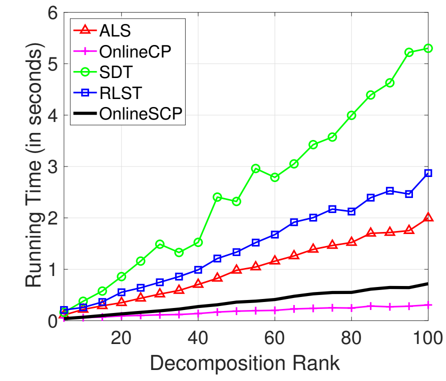


Figure 4.2: The efficiency of OnlineSCP is correlated to the ratio between the number of non-zeros in data and the overall length of its non-temporal modes

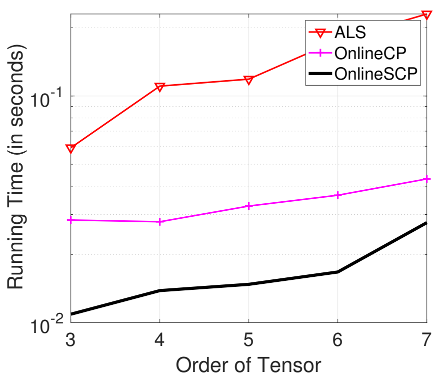
$|\Delta\Omega^+|$, then the speedup obtained by the incremental learning phase will be significantly exploited. In fact, by plotting $|\Delta\Omega^+|/J$ against the speedup (or relative memory) of our method w.r.t. ALS, we can see a clear correlation between this ratio and the efficiency performance as shown in Figure 4.2. This means that our method tends to work better for tensors that have relatively smaller overall length of the non-temporal modes. However, this does not necessarily means that our method is not suitable for large-scale sparse online tensors. In fact, it should be highlighted that in this experiment setting, each new data batch contains around 0.5% of overall data, so the maximum speedup can be gained in the MTTKRP calculation is about 200 times, compared to the ALS algorithm. In practice, as long as the time used by our algorithm for one new batch of data is less than the data generation time, one can always use a smaller batch size to gain even greater efficiency.



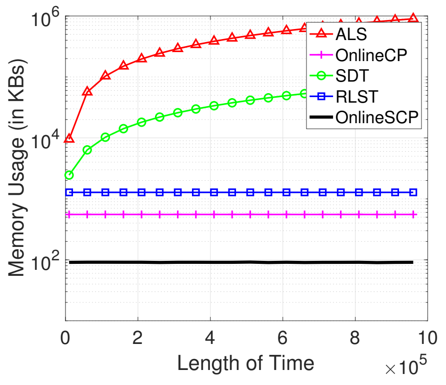
(a) $t \in [1 \times 10^4, 1 \times 10^6]$



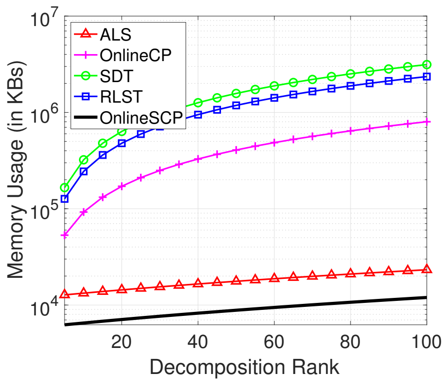
(b) $R \in [5, 100]$



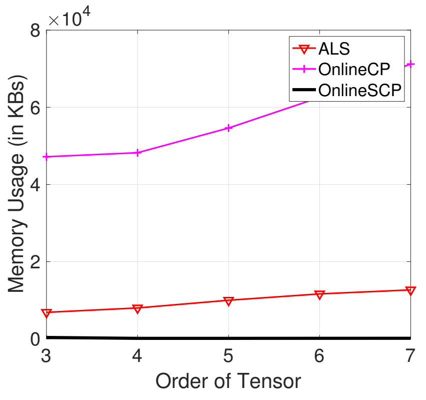
(c) $N \in [3, 7]$



(d) $t \in [1 \times 10^4, 1 \times 10^6]$



(e) $R \in [5, 100]$



(f) $N \in [3, 7]$

Figure 4.3: Scalability comparison w.r.t. different factors

4.4.3 Scalability

As demonstrated by results on real-world datasets, our proposed method, OnlineSCP, significantly improves on existing methods, especially for the efficiency aspect. To better show the merits of our method, we conduct a set of scalability experiments on a group of synthetic tensors that have been randomly generated. Three main factors are considered for scalability testing, including: the length of existing time, t, the decomposition rank, R, and the order of tensor, N.

As shown in the real-world experiments, existing online methods, OnlineCP, SDT and RLST, are specifically designed for dense tensors and do not scale well with large S. As a result, we set $S = 1000 \times 1000$ unless mentioned otherwise. Even though this is not a large-scale slice size, it is sufficient for us to see the general trends for both time and space consumption. The density of all synthetic tensors are fixed to 1×10^{-3} and we repeat all experiments 10 times and report the averaged results over these 10 runs.

Scalability v.s. t

In order to assess the scalability performance with respect to the length of existing time, t, we vary t from 10^4 to 10^6 and set $R = 5$ and $S = 100 \times 100$, since too large R and S will significantly slow down ALS. At each specific time length, we require all methods to process one slice of new data and report their running time and memory usage in Figure 4.3a and 4.3d. Both of them are presented in a log scale. It is clear that OnlineCP, RLST and our OnlineSCP are much better than ALS and SDT. Both the time and memory usage are independent of the existing time for the former, while a linear trend can be found for ALS and SDT. The time efficiency of our method is less than RLST and OnlineCP, since the size of data is not big enough to show the advantage of sparse operation compared to highly optimized matrix calculation. However, the memory usage of OnlineSCP is the least among all methods.

Scalability v.s. R

For scalability regarding to the decomposition rank, R, a $1000 \times 1000 \times 100$ tensor is used as the existing data and a new 1000×1000 slice is input to each method with R chosen from 5 to 100. The results can be found in Figure 4.3b and 4.3e (in log scale). As shown in Figure 4.3b, the running time of ALS, SDT and RLST grows much faster than OnlineCP and OnlineSCP. However, the high efficiency of OnlineCP is gained at the cost of a linear complexity for memory computation due to the intermediate data explosion issue. The growth rates of memory in ALS and OnlineSCP are considerably lower than the others since the sparse MTTKRP is calculated column by column and only a fixed amount of memory that depends on the number of non-zeros is used. The memory growth for these is mainly caused by the storage for loading matrices.

Scalability v.s. N

To validate the scalability of each algorithm with respect to the order of tensor, we conduct experiments on five tensors with orders ranging from 3 to 7, $t = 100$, $R = 5$ and the overall size of non-temporal mode $S \approx 1 \times 10^6$. It should be noted that SDT and RLST are omitted in this experiment since they work with 3^{rd}-order tensors only. Figure 4.3c and 4.3f demonstrate results for this part of experiment. Regarding to the running time, there is no surprise to see that all three methods are gradually increasing with the growth of tensor order, since more loading matrices need to be updated and the time-consuming MTTKRP calculation need to be done more frequently. In terms of memory usage, the space consumption of ALS and OnlineSCP is stable and an upwards trend can be found in OnlineCP, since it has a specific procedure to calculate all Khatri-Rao product series at a time, in order to speedup the decomposition for higher-order tensors.

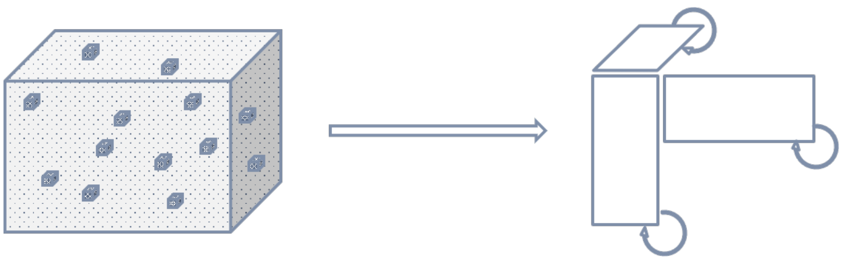
Overall, our method demonstrates excellent performance in terms of scalability w.r.t. different factors, compared to state-of-the-art methods. The time complexity of OnlineSCP is only linear w.r.t. the decomposition rank and the order of tensor, and independent of the existing time. Compared to other online methods, the growth of time in our method is less noticeable since the scaling factor in OnlineSCP is the number of non-zeros in the new data, while for other methods, they have to be scaled by the overall size of non-temporal modes. Additionally, our method is also the most memory-efficient and scalable method among all algorithms and it always consumes the least amount of memory, which is also linear w.r.t. the number of non-zeros in the new data.

4.5 Summary

To conclude, this chapter addressed the CP decomposition problem for online sparse tensors. We proposed an efficient algorithm, OnlineSCP, to incrementally track the up-to-date CP decomposition when a new batch of data is appended to the time mode of a sparse tensor. The complexity of our algorithm is only linear w.r.t. the number of non-zeros in the new data, which is significantly better than existing online approaches that are designed for dense tensors. As evaluated on both real-world and synthetic datasets, our algorithm demonstrated considerable improvements over state-of-the-art techniques, in terms of decomposition quality, time and space efficiency, and scalability.

Chapter 5

Sparse Tensor Decomposition with Constraints and Element-wise Dynamic Learning



This chapter discusses sparse tensor decomposition from three perspectives: *(i)* how to treat zero values in sparse data, *(ii)* how to impose constraints, *(iii)* how to dynamically learn the effect of a new element that will have on an existing decomposition. We propose a weighted decomposition formulation with an efficient learning strategy for dealing with all these issues as a whole. The content of this chapter is adapted from the following published paper:

Zhou, S., Erfani, S.M. and Bailey, J., 2017, November. SCED: a general framework for sparse tensor decomposition with constraints and element-wise dynamic learning. In *2017 IEEE International Conference on Data Mining (ICDM)* (pp. 675-684). IEEE.

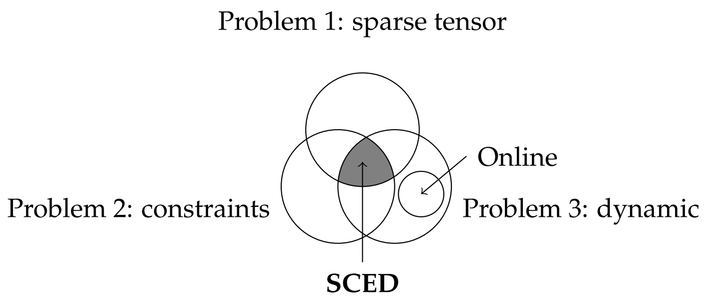


Figure 5.1: Research problem of SCED

5.1 Introduction

Multi-dimensional data is not uncommon to see nowadays, from video clips spread in people's daily life [Mu et al., 2014], to time-evolving graphs and networks such as social networks [Anandkumar et al., 2014], to spatio-temporal data like fMRI [Davidson et al., 2013, Ma et al., 2016]. Tensor is a natural representation for such data because of its ability to maintain the structure information. However, working with tensors is not easy due to the complex relationships among different dimensions. As a result, in order to simplify data, extract useful features and discovery meaningful knowledge, CP decomposition has been extensively studied and widely applied in recent years [Kolda and Bader, 2009, Cichocki et al., 2015, Sidiropoulos et al., 2017].

As shown in Fig. 5.1, the research question we focus in this chapter contains three parts: problem 1) we seek a unified formulation for sparse tensor decomposition and an efficient solving algorithm; problem 2) we look at constrained decompositions as well since they usually promote meaningful results; and problem 3) we also aim at designing an adaptive learning paradigm to tackle element-wise dynamic updates. It should be noted that the dynamic aspect we focus here is different from the online CP decomposition problem we discussed in last two chapters, where the tensor is growing slice by slice. While in this chapter, the data stream consists of new cell entries that can dynamically happen at any position.

A motivating example is context-aware recommender system, where ratings are modeled by a three-way tensor as $user \times item \times context$. Besides challenges imposed by its large scale and sparsity, some issues need to be underlined. First, the role of zeros might be different depending on rating type. For example, zero entries in a system with explicit ratings are usually treated as missing values and ignored [Acar et al., 2011a, Rai et al., 2014, Zhao et al., 2015], while for implicit feedback like user click logs, one cannot simply discard these valuable zeros as they represent hidden preferences such as dislike [Hidasi and Tikk, 2012]. However, existing methods are usually specifically designed for one of above cases. Thus, an algorithm that works for one type of data may not be applicable and easily adaptable to the others.

Second, there usually exists a rich body of domain knowledge in practice. Such useful information can be modeled as constraints that a decomposition needs to satisfy. With the help of constraints, the produced decomposition will be more meaningful and interpretable. One example is non-negativity [Welling and Weber, 2001, Lin, 2007, Cichocki and Anh-Huy, 2009, Hansen et al., 2015] where each user's preference is modeled by a set of non-negative coefficients, which stand for his/her favor to each latent item/context group. It is important to efficiently bring constraints into the decomposition, but many current techniques for this task only address dense tensors and have poor efficiency for sparse ones.

Finally, it is quite common to see that after learning a decomposition model from historical data, a large amount of new data, (user,item,context) tuples in this example, is generated at a high speed. A static model is less and less reliable with the growth of new data. However, it is too expensive to recompute a new CP decomposition due to the high time complexity. As a result, a dynamic learning model is desired.

Overall, existing methods have only partially addressed the above issues. How to handle them as a whole is still an open question. To close the gap, we propose a new algorithm, SCED, and summarize our contributions as follows:

- We propose a new formulation and an efficient algorithm to find the CP decomposition of sparse tensors in general.

- We enhance our method with advanced features such as the capability to incorporate constraints, and the ability to dynamically track a new decomposition on-the-fly.
- Via experiments on synthetic and real-world datasets, our approach demonstrates high performance in terms of effectiveness, efficiency and scalability, compared to the state-of-the-art.

5.2 Related Work

Most existing work for sparse CP decomposition specifically targets one of three special cases: 1) *True Observations* (TO) where zeros are treated the same as non-zero observations; 2) *Missing Values* (MV) where all zeros are ignored; and 3) *Implicit Information* (II) where zeros slightly contribute to the model, but are less emphasised than non-zeros. For the TO case, [Bader and Kolda, 2007] propose an algorithm based on ALS, tailoring it to sparse tensors, with support from special data structures and customized tensor-related operations. With advances in distributed computing, techniques like MapReduce have also been used to further speed up ALS for large-scale sparse tensors[Kang et al., 2012,Choi and Vishwanathan, 2014]. For MV, the Weighted CP Decomposition (WCPD) is a well suited framework proposed in [Acar et al., 2011a], and an optimization based algorithm, WOPT, is proposed to decompose sparse tensors with missing values. Bayesian methods have also been explored and can be found in [Rai et al., 2014,Zhao et al., 2015]. Compared to TO and MV, II has been less studied and the only work to our knowledge is [Hidasi and Tikk, 2012]. This is an extension of Matrix Factorization (MF) for implicit feedback [Hu et al., 2008,Pilászy et al., 2010,Devooght et al., 2015] to tensors, where a small uniform weight is assigned to zero entries.

In terms of constrained CP decomposition, non-negativity and sparseness are two popular constraints explored by researchers. An early approach [Welling and Weber, 2001] handle non-negativity based on Lee and Seung's multiplicative update rules [Lee and Seung, 2001]. Additionally,

Table 5.1: Comparison to related works

	General	Efficient	Scalable	Constraints	Dynamic
ALS	TO	✓	✓		
MU	TO	✓	✓	✓	
WOPT	MV				
iTALS	II				✓*
SCED	TO, MV, II	✓	✓	✓	✓

* not reported in original paper, but applicable with modifications

non-negative CP decomposition algorithms based on NALS [Lin, 2007], HALS [Cichocki and Anh-Huy, 2009] and Newton's method [Hansen et al., 2015] have also been proposed. Similar to MF, to promote sparse solutions, the l_1-norm is usually used as regularization to CP decomposition and an implementation of this idea can be found in [Liu et al., 2012a].

There is less existing work on online CP decomposition that is applicable for dynamic tensors. Works in [Nion and Sidiropoulos, 2009, Sun et al., 2008] assume new data is appended slice by slice, which does not match the dynamic element-wise scenario we address in this chapter. The most related papers are for MF [Devooght et al., 2015, He et al., 2016], where element-wise dynamic changes are handled by only refreshing corresponding rows related to the new entry.

Overall, existing work has only partially addressed our research question. This motivates us to develop a general framework that can address all aforementioned critical issues together. A comparison between our work and the most relevant and representative works is shown in Table 5.1.

5.3 Weighted CP Decomposition

In the MV case where zeros represent missing values, such zeros should be omitted since they carry no information and fitting them will only lead the model to a wrong direction. As a result, in order to handle missing values, Weighted CP Decomposition (WCPD) is introduced in [Acar et al., 2011a]

as

$$\mathcal{L}_{wcpd} = \frac{1}{2}\sum\nolimits_{\Omega} w_{i_1,\ldots,i_N}(x_{i_1,\ldots,i_N} - \hat{x}_{i_1,\ldots,i_N})^2, \tag{5.1}$$

$$w_{i_1,\ldots,i_N} = \begin{cases} 0 & \text{if } x_{i_1,\ldots,i_N} \text{ is missing,} \\ 1 & \text{otherwise,} \end{cases} \tag{5.2}$$

where $w_{i_1,\ldots,i_N}$ is the weight assigned to the $(i_1,\ldots,i_N)$-th entry's approximation, and Ω is a set contains all possible indices combinations such that $\{(i_1,\ldots,i_N) \in \Omega | \forall i_n \in [1, I_n], \forall n \in [1, N]\}$. Furthermore, let Ω^+ be the indices set corresponding to all *non-zero* entries and Ω^- be the indices set related to all *zero* entries, and we have $\Omega^+ \cup \Omega^- = \Omega$.

To minimize the loss function, [Acar et al., 2011a] treat this as an optimization problem and all parameters in loading matrices are stacked as a parameter vector, which can be simultaneously optimized by solvers such as Nonlinear Conjugate Gradient (NCG). Similar to ALS for sparse tensors, this Weighted OPTimization (WOPT) strategy has $\mathcal{O}(|\Omega^+|R)$ time complexity.

The II case is well studied in recommender systems with implicit feedback by MF [Hu et al., 2008, Pilászy et al., 2010, Devooght et al., 2015]. Under such circumstances, the zeros cannot be simply ignored as missing values, nor be equally treated as observations, since they somehow measure implicit information such as dislikes. One popular approach for this type of data is to assign different weights to zero and non-zero entries, so that the contribution of implicit information is leveraged.

Inspired by this, [Hidasi and Tikk, 2012] propose the iTALS algorithm, which extends the WCPD framework to the II case by modifying the weighting schema in Eq. (5.2) as

$$w_{i_1,\ldots,i_N} = \begin{cases} 1 & \text{if } x_{i_1,\ldots,i_N} = 0, \\ \alpha \cdot \#(i_1,\ldots,i_N) > 1 & \text{otherwise,} \end{cases}$$

where α is a parameter to control the difference of weights between non-zeros and zeros, and $\#(i_1,\ldots,i_N)$ is the number of event tuples corresponding to entry $(i_1, \dots, i_N)$.

Hidasi and Tikk optimize this implicit CP decomposition by row-wise ALS, which shares the same principle as typical ALS but just optimizes one row at a time as

$$\mathbf{a}^{(n)}_{i_n\cdot} \leftarrow \mathbf{x}_{(n)_{i_n\cdot}} \mathbf{W}^{(i_n)} \mathbf{B}^{(n)} (\mathbf{B}^{(n)\top} \mathbf{W}^{(i_n)} \mathbf{B}^{(n)})^{-1},$$

where $\mathbf{x}_{(n)_{i_n\cdot}}$ is the i_n-th row of the mode-n unfolding $\mathbf{X}_{(n)}$, $\mathbf{W}^{(i_n)}$ is a matrix with all weights related to $\mathbf{x}_{(n)_{i_n\cdot}}$ on its diagonal. The major disadvantage of iTALS is its inefficiency, as a $R \times R$ matrix, $\mathbf{B}^{(n)\top} \mathbf{W}^{(i_n)} \mathbf{B}^{(n)}$, has to be calculated for each individual rows at a cost of $\mathcal{O}(|\Omega^+_{i_n}|R^2)$, which results in an overall complexity as $\mathcal{O}(|\Omega^+|R^2)$.

5.4 SCED Algorithm

In this section, we introduce our general approach to find the CP decomposition of sparse tensors. We first show how to model all aforementioned special cases, TO, MV and II, under the WCPD framework by modifying the weighting schema. Then an efficient solving algorithm is proposed, which is also able to handle constraints such as non-negativity and sparseness. Lastly, we show how to extent our algorithm to dynamic environments where new cell entries arrive in a streaming fashion.

5.4.1 A General Weighting Schema

Initially, WCPD was proposed for the MV case. While it is easy to see that the TO case can be readily modeled in such formulation by assigning the weight of zero entries to 1. Additionally, we notice that the key concept for handling the II case is a uniform weight, which is always larger than 0, but smaller than the weight of non-zeros, is given to all zero entries. Based on these, we propose the following weighting schema that models all these three cases:

$$w_{i_1,\dots,i_N} = \begin{cases} \alpha \in [0,1] & \text{if } x_{i_1,\dots,i_N} = 0, \\ 1 & \text{otherwise,} \end{cases} \tag{5.3}$$

where α is the weight of zero entries. When $\alpha = 1$, this is equivalent to TO case; it reduces to the MV case by letting $\alpha = 0$; and the II case is also included in our proposed schema by choosing $\alpha \in (0,1)$.

5.4.2 Derivation of SCED

Our solving algorithm is a generalization of the Element-wise ALS (EALS, as discussed in §2.3.2) [He et al., 2016] to sparse tensors. Its principle is similar to standard ALS. The key difference is that in EALS, only one parameter (one cell in a loading matrix) is updated at a time, while in ALS, each time one loading matrix is estimated as a whole. The advantage of using a finer-grain update strategy is that it provides the freedom to choose the desired updating sequence without sacrificing effectiveness and efficiency. For example, one can decide to update a small fraction of cells in one loading matrix at first, then jump to the estimation of another part of parameters in other loading matrices. This is essential for dynamic updating, which will be discussed later in §5.4.4.

Specifically, let $j_n = (i_1, \dots, i_{n-1}, i_{n+1}, \dots, i_N)$, $b^{(n)}_{j_n r} = \prod_{\tilde{n} \neq n}^{N} a^{(\tilde{n})}_{i_{\tilde{n}} r}$, and recall that one element in the tensor can approximated by Eq. (2.16) as

$$\begin{aligned} \hat{x}_{i_n j_n} &= \sum_{r=1}^{R} a^{(n)}_{i_n r} b^{(n)}_{j_n r} = \sum_{\tilde{r} \neq r}^{R} a^{(n)}_{i_n \tilde{r}} b^{(n)}_{j_n \tilde{r}} + a^{(n)}_{i_n r} b^{(n)}_{j_n r} \\ &= \hat{x}^{r}_{i_n j_n} + a^{(n)}_{i_n r} b^{(n)}_{j_n r}, \end{aligned}$$

where $\hat{x}^{r}_{i_n j_n}$ is defined as the residual of $x_{i_n j_n}$, we can rewrite Eq. (5.1) as

$$\begin{aligned} \mathcal{L}_{wcpd} &= \frac{1}{2} \sum\nolimits_{\Omega} w_{i_n j_n} (x_{i_n j_n} - \hat{x}_{i_n j_n})^2 \\ &= \frac{1}{2} \sum\nolimits_{\Omega} w_{i_n j_n} (x_{i_n j_n} - \hat{x}^{r}_{i_n j_n} - a^{(n)}_{i_n r} b^{(n)}_{j_n r})^2. \end{aligned}$$

The partial derivative of $\mathcal{L}_{wcpd}$ w.r.t. the (i_n, r)-th parameter of loading matrix $\mathbf{A}^{(n)}$ is

$$\frac{\partial \mathcal{L}_{wcpd}}{\partial a_{i_n r}^{(n)}} = -\sum\nolimits_{\Omega_{i_n}} w_{i_n j_n}(x_{i_n j_n} - \hat{x}_{i_n j_n}^r - a_{i_n r}^{(n)} b_{j_n r}^{(n)}) b_{j_n r}^{(n)}.$$

By setting the derivative to 0, then we reach the closed form solution for $a_{i_n r}^{(n)}$ as

$$a_{i_n r}^{(n)} \leftarrow \frac{\sum_{\Omega_{i_n}} w_{i_n j_n}(x_{i_n j_n} - \hat{x}_{i_n j_n}^r) b_{j_n r}^{(n)}}{\sum_{\Omega_{i_n}} w_{i_n j_n}(b_{j_n r}^{(n)})^2}. \tag{5.4}$$

By applying the weighting schema Eq. (5.3) to Eq. (5.4) we have

$$a_{i_n r}^{(n)} \leftarrow \frac{\sum_{\Omega_{i_n}^+}(x_{i_n j_n} - \hat{x}_{i_n j_n}^r) b_{j_n r}^{(n)} - \alpha \sum_{\Omega_{i_n}^-} \hat{x}_{i_n j_n}^r b_{j_n r}^{(n)}}{\sum_{\Omega_{i_n}^+}(b_{j_n r}^{(n)})^2 + \alpha \sum_{\Omega_{i_n}^-}(b_{j_n r}^{(n)})^2}.$$

So far there are four summations for updating $a_{i_n r}^{(n)}$. The left two are related to non-zero entries only, which can be calculated efficiently as $|\Omega_{i_n}^+| \ll |\Omega_{i_n}|$. While the other two have to iterate over $\Omega_{i_n}^-$, which contains indices for all zeros in the i_n-th row of $\mathbf{X}_{(n)}$. In the following we show that such access to all zero entries is unnecessary and avoidable.

Since $\Omega_{i_n}^+ \cup \Omega_{i_n}^- = \Omega_{i_n}$ we can transform the zero entries related summations as

$$\sum\nolimits_{\Omega_{i_n}^-} \hat{x}_{i_n j_n}^r b_{j_n r}^{(n)} = \sum\nolimits_{\Omega_{i_n}} \hat{x}_{i_n j_n}^r b_{j_n r}^{(n)} - \sum\nolimits_{\Omega_{i_n}^+} \hat{x}_{i_n j_n}^r b_{j_n r}^{(n)},$$
$$\sum\nolimits_{\Omega_{i_n}^-} (b_{j_n r}^{(n)})^2 = \sum\nolimits_{\Omega_{i_n}} (b_{j_n r}^{(n)})^2 - \sum\nolimits_{\Omega_{i_n}^+} (b_{j_n r}^{(n)})^2.$$

Again, the non-zero related summations can be readily obtained, and the major concern is how to efficiently get the terms related to Ω_{i_n}. Let $\mathbf{Q}^{(n)} = \circledast_{i \neq n}^N (\mathbf{A}^{(i)\top} \mathbf{A}^{(i)})$, one can easily verify that $\sum_{\Omega_{i_n}} (b_{j_n r}^{(n)})^2 = q_{rr}^{(n)}$. Additionally, recall that $\hat{x}_{i_n j_n}^r = x_{i_n j_n} - a_{i_n r}^{(n)} b_{j_n r}^{(n)}$, $\sum_{\Omega_{i_n}} \hat{x}_{i_n j_n}^r b_{j_n r}^{(n)}$ can be rewritten

as

$$\begin{aligned}
&\sum\nolimits_{\Omega_{i_n}} (\sum\nolimits_{r=1}^{R} a_{i_n r}^{(n)} b_{j_n r}^{(n)} - a_{i_n r}^{(n)} b_{j_n r}^{(n)}) b_{j_n r}^{(n)} \\
&= \sum\nolimits_{r=1}^{R} a_{i_n r}^{(n)} \sum\nolimits_{\Omega_{i_n}} (b_{j_n r}^{(n)})^2 - a_{i_n r}^{(n)} \sum\nolimits_{\Omega_{i_n}} (b_{j_n r}^{(n)})^2 \\
&= \mathbf{a}_{i_n \cdot}^{(n)} \mathbf{q}_{\cdot r}^{(n)} - a_{i_n r}^{(n)} q_{rr}^{(n)},
\end{aligned} \tag{5.5}$$

which can be efficiently computed in $\mathcal{O}(R)$ time.

Therefore, if $\mathbf{Q}^{(n)}$ is known, the complexity to update $a_{i_n r}^{(n)}$ is only related to $|\Omega_{i_n}^{+}|$. In addition, to speed up the computing of $\mathbf{Q}^{(n)}$, a list of auxiliary matrices $\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}$ where $\mathbf{U}^{(n)} = \mathbf{A}^{(n)\top} \mathbf{A}^{(n)}, n \in [1, N]$ can be calculated and cached in advance, which results in $\mathcal{O}(NR^2)$ time. And only the n-th auxiliary matrix $\mathbf{U}^{(n)}$ need to be re-computed after updating $\mathbf{A}^{(n)}$.

Overall, by putting everything together, we reach the final update rule as,

$$a_{i_n r}^{(n)} \leftarrow \frac{\sum_{\Omega_{i_n}^{+}} (x_{i_n j_n} - (1-\alpha)\hat{x}_{i_n j_n}^{r}) b_{j_n r}^{(n)} - \alpha(\mathbf{a}_{i_n \cdot}^{(n)} \mathbf{q}_{\cdot r}^{(n)} - a_{i_n r}^{(n)} q_{rr}^{(n)})}{(1-\alpha)\sum_{\Omega_{i_n}^{+}} (b_{j_n r}^{(n)})^2 + \alpha q_{rr}^{(n)}}, \tag{5.6}$$

and our proposed SCED algorithm is summarized in Algorithm 5.1.

5.4.3 Incorporating Constraints

Here we mainly focus on two types of constraints: non-negativity and regularizations, due to their popularity.

Non-negativity

Non-negativity is a widely used constraint in CP decomposition to enforce interpretable solutions. We handle this by applying a simple "half-wave rectifying" [Cichocki and Anh-Huy, 2009] projection to each updating. Specifically, after getting an updated value by Eq. (5.6), positive value is kept while negative one is replaced by 0 (or a small number such 1×10^{-9}

Algorithm 5.1: SCED Algorithm

```
Input: Input tensor $\mathcal{X}$, decomposition rank $R$, weight for zeros $\alpha$
Output: Loading matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$
Randomly initialize $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$
for $(i_1, \ldots, i_N) \in \Omega^+$ do $\hat{x}_{i_1,\ldots,i_N} \leftarrow$ Eq. (2.16)
for $n \leftarrow 1$ to $N$ do $\mathbf{U}^{(n)} = \mathbf{A}^{(n)\top}\mathbf{A}^{(n)}$
while stopping criteria is not met do
    for $n \leftarrow 1$ to $N$ do
        $\mathbf{Q}^{(n)} \leftarrow \circledast_{i \neq n}^{N} \mathbf{U}^{(i)}$
        for $i_n \leftarrow 1$ to $I_n$ do
            for $(i_n, j_n) \in \Omega_{i_n}^+$ do $b_{j_n \cdot}^{(n)} = \prod_{\tilde{n} \neq n}^{N} a_{i_{\tilde{n}} \cdot}^{(\tilde{n})}$
            for $r \leftarrow 1$ to $R$ do
                $\hat{\mathbf{x}}_{i_n \cdot}^{r} = \hat{\mathbf{x}}_{i_n \cdot} - a_{i_n r}^{(n)} \mathbf{b}_{\cdot r}^{(n)\top}$
                $a_{i_n r}^{(n)} \leftarrow$ Eq. (5.6)
                $\hat{\mathbf{x}}_{i_n \cdot} = \hat{\mathbf{x}}_{i_n \cdot}^{r} + a_{i_n r}^{(n)} \mathbf{b}_{\cdot r}^{(n)\top}$
            end
        end
        $\mathbf{U}^{(n)} = \mathbf{A}^{(n)\top}\mathbf{A}^{(n)}$
    end
end
```

for numerical stableness). The correctness of this is shown as follows.

Theorem 1. *The minimization problem*

$$\min_{a_{i_n r}^{(n)} \geq 0} \mathcal{L}_{wcpd}$$

has the unique solution as

$$a_{i_n r}^{(n)} = \left[\frac{\sum_{\Omega_{i_n}} w_{i_n j_n} (x_{i_n j_n} - \hat{x}_{i_n j_n}^{r}) b_{j_n r}^{(n)}}{\sum_{\Omega_{i_n}} w_{i_n j_n} (b_{j_n r}^{(n)})^2} \right]_+$$

where $[z]_+ = \mathtt{max}(0, z)$.

Proof. This can be proved in similar way as [Kim et al., 2014]. We can organize the derivative of $\mathcal{L}_{wcpd}$ w.r.t. $a_{i_n r}^{(n)}$ as

$$\begin{aligned}\frac{\partial \mathcal{L}_{wcpd}}{\partial a_{i_n r}^{(n)}} &= a_{i_n r}^{(n)} \cdot slope + intercept,\\ slope &= \textstyle\sum_{\Omega_{i_n}} w_{i_n j_n} (b_{j_n r}^{(n)})^2,\\ intercept &= -\textstyle\sum_{\Omega_{i_n}} w_{i_n j_n} (x_{i_n j_n} - \hat{x}_{i_n j_n}^{r}) b_{j_n r}^{(n)}.\end{aligned}$$

If $intercept \leq 0$, it is clear that $\mathcal{L}_{wcpd}$ reaches its minimum at $a_{i_n r}^{(n)} = -\frac{intercept}{slope}$, where the derivative is 0; if $intercept > 0$, the loss, $\mathcal{L}_{wcpd}$, increases as $a_{i_n r}^{(n)}$ become larger than 0. Thus, the minimum is attained at $a_{i_n r}^{(n)} = 0$. As a result, combining both cases, the solution can be expressed as $a_{i_n r}^{(n)} = [-\frac{intercept}{slope}]_+$ □

Regularization

A common strategy to take constraints into consideration is to treat them as regularizations to the original optimization problem as

$$\min_{\mathbf{A}^{(1)} \dots \mathbf{A}^{(N)}} \mathcal{L}_{wcpd} + \sum_{n=1}^{N} \lambda_n \phi_n(\mathbf{A}^{(n)}), \tag{5.7}$$

where λ_n is a non-negative regularization parameter and ϕ_n is the regularizing function applied to the n-th loading matrix. Depending on constraints, different ϕ can be used. For example, if $\phi_n(\mathbf{A}^{(n)}) = \frac{1}{2}||\mathbf{A}^{(n)}||^2$, Tikhonov regularization is used to prevent overfitting; and $\phi_n(\mathbf{A}^{(n)}) = \sum_{i_n=1}^{I_n} ||\mathbf{a}_{i_n \cdot}^{(n)}||_1$ is another widely used regularization to promote sparseness in solution.

The above regularized optimization problem can be easily solved by our SCED algorithm using a similar derivation in §5.4.2. Due to the page limit, here we directly give the closed form solutions based on Eq. (5.4), and of course similar efficient versions to Eq. (5.6) can be derived

$$a_{i_n r}^{(n)} \leftarrow \begin{cases} \frac{\sum_{\Omega_{i_n}} w_{i_n j_n}(x_{i_n j_n} - \hat{x}^r_{i_n j_n}) b_{j_n r}^{(n)}}{\sum_{\Omega_{i_n}} w_{i_n j_n}(b_{j_n r}^{(n)})^2 + \lambda_n} & \ell_2\text{-norm,} \\ \frac{\sum_{\Omega_{i_n}} w_{i_n j_n}(x_{i_n j_n} - \hat{x}^r_{i_n j_n}) b_{j_n r}^{(n)} + \lambda_n \cdot \mathtt{sign}(a_{i_n r}^{(n)})}{\sum_{\Omega_{i_n}} w_{i_n j_n}(b_{j_n r}^{(n)})^2} & \ell_1\text{-norm.} \end{cases}$$

5.4.4 Dynamic Learning

In real-world applications, it is not uncommon to see that after decomposing a tensor, new data will keep arriving at a high speed. A method that can efficiently track the new decompositions in such dynamic scenario is desired, since the static model may not perform well because it is not up-to-date.

This problem has been extensively studied for the matrix case and a common assumption is that the new data will only have considerable impact for local features, while the global model will not be affected significantly. For example, given a matrix $\mathbf{X} \in \mathbb{R}^{M \times N}$ and its decomposition $\mathbf{W} \in \mathbb{R}^{M \times R}$ and $\mathbf{H} \in \mathbb{R}^{N \times R}$, in order to learn a new interaction x_{mn}, only the m-th and n-th rows of $\mathbf{W}$ and $\mathbf{H}$ will be updated, while other parameters remain unchanged. Since CP decomposition is a generalization of MF for multi-way data, a similar vector retaining strategy can be used for dynamically learning new incoming interactions on tensorial data and we summarize the dynamic version of SCED in Algorithm 5.2.

5.4.5 Complexity

Here we briefly give a time complexity analysis and the comparison with existing methods can be found in Table 5.2. ALS, WOPT and iTALS are chosen baselines that specifically target the TO, MV and II cases, respectively.

For SCED, as shown in Algorithm 5.1, to update one loading matrix, $\mathbf{Q}^{(n)}$ can be calculated in $(N-1)R^2$ operations (line 6) and for each row

Algorithm 5.2: SCED for Dynamic Learning

```
Input: Existing loading matrices $\mathring{\mathbf{A}}^{(1)}, \ldots, \mathring{\mathbf{A}}^{(N)}$, new interaction $x_{i_1,\ldots,i_N}$
Output: Updated matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$
1  for $n \leftarrow 1$ to $N$ do
2      $\mathbf{A}^{(n)} \leftarrow \mathring{\mathbf{A}}^{(n)}$
3      if $\mathbf{a}^{(n)}_{i_n\cdot}$ not exists then random initialize $\mathbf{a}^{(n)}_{i_n\cdot}$
4  end
5  $\hat{x}_{i_1,\ldots,i_N} \leftarrow$ Eq. (2.16)
6  while stopping criteria is not met do
7      for $n \leftarrow 1$ to $N$ do
8          $\mathring{\mathbf{U}} \leftarrow \mathbf{a}^{(n)\top}_{i_n\cdot} \mathbf{a}^{(n)}_{i_n\cdot}$
9          update $\mathbf{a}^{(n)}_{i_n\cdot}$                  /* line 8-13 of Algorithm 5.1 */
10         $\mathbf{U} \leftarrow \mathbf{a}^{(n)\top}_{i_n\cdot} \mathbf{a}^{(n)}_{i_n\cdot}$
11         $\mathbf{U}^{(n)} = \mathbf{U}^{(n)} - \mathring{\mathbf{U}} + \mathbf{U}$          /* update cache */
12     end
13 end
```

in $\mathbf{A}^{(n)}$, a $|\Omega^+_{i_n}| \times R$ matrix $\mathbf{B}^{(n)}$ is generated (line 8). Then each element $a^{(n)}_{i_n r}$ is updated at a cost of $\mathcal{O}(|\Omega^+_{i_n}| + R)$ (line 10-12). In total, the time cost for updating $\mathbf{A}^{(n)}$ is $\mathcal{O}(|\Omega^+|R + I_n R^2)$. This procedure is repeated for all loading matrices and takes $\mathcal{O}(|\Omega^+|NR + \sum_{n=1}^{N} I_n R^2)$ operations for one iteration, which is dominated by $\mathcal{O}(|\Omega^+|R)$. In summary, for the TO and MV cases, our method shares the same efficiency as the state-of-the-art, while our method is R times faster than iTALS for the II case.

Similar to the static case, the complexity of dynamic SCED algorithm is dominated by $\mathcal{O}(|\Omega^+_{i_n}|R)$. While in general, using our method for updating existing CP decomposition is around I times faster than batch methods, where $I = \texttt{mean}(I_1, \ldots, I_N)$, since there is only one row need to be updated for each loading matrix.

Table 5.2: Time complexity

	SCED	Baseline
TO ($\alpha = 1$) & MV ($\alpha = 0$)	$\mathcal{O}(\|\Omega^+\|R)$	
II ($\alpha \in (0,1)$)	$\mathcal{O}(\|\Omega^+\|R)$	$\mathcal{O}(\|\Omega^+\|R^2)$

5.5 Empirical Analysis

In this section, we compare the performance of our SCED algorithm with the state-of-the-art, in terms of effectiveness and efficiency. The performance is evaluated on both synthetic and real-world datasets. For each dataset, both static and dynamic settings are tested. After that, based on the investigation on large-scale synthetic tensors, we further analyze the scalability of our approach.

5.5.1 Experiment Specifications

Datasets

The first half of the experiments are conducted on three $200 \times 200 \times 200$ synthetic datasets: SYN-TO, SYN-MV and SYN-II. All of them have rank as 20 and density as 1%. The key difference among them is the role of zero entries. SYN-TO is generated by sparse loading matrices such that its zeros are true observations. In contrast, the zeros in SYN-MV are missing values that are randomly sampled from a dense tensor, which is generated by random loading matrices. Since it is non-trivial to simulate the implicit feedback, while the weight of zeros in such situation lies between the weights of TO and MV cases, we create SYN-II by mixing the data generation protocols of SYN-TO and SYN-MV. Specifically, less sparse loading matrices are generated to form a tensor with around 50% non-zeros, from which 1% values are randomly sampled as the SYN-II.

In addition, to better evaluate the performance of our algorithm in real-world applications, three datasets of varying characteristics have been used:

MovieLens [Harper and Konstan, 2016], LastFM [Celma, 2010] and MathOverflow [Paranjape et al., 2017]. Their detail can be found in Table 5.3.

Table 5.3: Detail of real-world datasets

datasets	Size	nnz^*	Density
MovieLens	$6040 \times 3952 \times 1040$	1×10^6	4.03×10^{-5}
LastFM	$991 \times 1000 \times 168$	2.9×10^6	1.73×10^{-2}
MathOverflow	$24818 \times 24818 \times 2351$	4×10^5	2.75×10^{-7}

* number of non-zeros

Baselines

Four baselines have been selected as the competitors to evaluate the performance in our experiment:

(i) ALS: an implementation for sparse tensors from Tensor Toolbox [Bader et al., 2015].

(ii) MU: a multiplicative update rule based algorithm for non-negative CP decomposition from the Tensor Toolbox [Bader et al., 2015].

(iii) WOPT [Acar et al., 2011a]: an algorithm for decomposing incomplete tensors based on weighted optimization.

(iv) iTALS [Hidasi and Tikk, 2012]: an approach that decomposes sparse tensors that represent implicit feedbacks.

It should be noted that all these baselines are batch methods and there is no existing work that can be directly used for element-wise dynamic updating on sparse tensors. However, since iTALS has a row-wise update rule, we modify it under the same vector-retaining model as our method, as a baseline that is able to perform dynamic learning.

Evaluation Metrics

The empirical performance is measured from both effectiveness and efficiency aspects.

In terms of effectiveness, for synthetic datasets, because the ground truth loading matrices are known already, *fitness* is used and defined as

$$\textit{fitness} \triangleq \left(1 - \frac{\left\|\hat{\mathcal{X}} - \mathcal{X}\right\|}{\left\|\mathcal{X}\right\|}\right),$$

where $\mathcal{X}$ is the tensor formed by ground truth, $\hat{\mathcal{X}}$ is the estimation and $\left\|\bullet\right\|$ denotes the Frobenius norm. The closer the fitness to 1, the better the decomposition is. However, for each real-world dataset, a test set (10%) is sampled and *Root Mean Square Error* (RMSE) is used for measuring the decomposition quality, since the ground truth is not given. The lower the RMSE, the better the result.

In addition, with respect to efficiency, for static decomposition, the average *running time* for one iteration, measured in seconds, is reported in order to validate the time efficiency of each algorithm. On the other hand, the running time for processing one new entry is recorded to compare the efficiency under the dynamic setting.

Experimental Setup

For both synthetic and real-world datasets, there are two types of experiments that have been conducted for performance evaluation: static and dynamic settings.

Static setting: given a data tensor, it is decomposed by each algorithm with the same random initialization. It should be noted that for real-world tensors, 10% of observations are randomly sampled and held out as a test set for RMSE calculation. In terms of experimental parameters, 20 has been used as the decomposition rank over all experiments, since we are not aiming at finding the best decomposition, but are more interested in the relative performance between our proposal and the baselines. For synthetic datasets, the maximum number of iterations is set to 50. While for real-world datasets, this has been set to 10 due to the low efficiency of iTALS and WOPT.

Dynamic setting: 10% of observations are randomly sampled as the dynamic set, such that each dataset is divided into different parts: 90% training, 10% dynamic for synthetic datasets, and 80% training, 10% dynamic and 10% testing for real-world datasets. The training set is decomposed by the batch algorithm with different random initializations, and the best result is chosen as the base for dynamic learning. In the dynamic updating phase, at each time stamp, an entry from the dynamic set is randomly selected and processed by both our SCED algorithm and the baselines. Specifically, for batch baselines (ALS and WOPT), the previous decomposition is used as a hot start and only one iteration is performed to process the new data. After that, effectiveness (fitness for synthetic, RMSE for real-world datasets) and efficiency (running time in seconds) metrics are recorded for performance.

Another difference between synthetic and real-world experiments is the choice of baselines. For synthetic datasets, since the role of zeros are known, only the corresponding algorithms are used for comparison. For example, ALS and MU are selected as baselines for SYN-TO, compared to our SCED variants, SCED-t [1] and SCED-nn [2]; while the comparison is only conducted between WOPT and SCED-m [3] for SYN-MV. Conversely, all algorithms are used for SYN-II, in order to show the merits of assigning small weight to zero entries under such circumstance. For real-world datasets, all baselines are used for comparison in the static experiment. While under the dynamic setting, only SCED-i [4] and iTALS are used for dynamic learning, because of their relative good performance in the static case, and the low efficiency of other batch baselines.

In terms of algorithm-specified parameters, apart from the aforementioned parameters, no parameter needs to be tuned for ALS and MU. For WOPT, default parameters have been used for its internal line search procedure. For SCED-i and iTALS, the weight of zeros (α) is set as the density of each dataset, in order to balance the contribution of non-zero and zero

[1]SCED for the true observation case

[2]SCED for true observation case with non-negative constraint

[3]SCED for the missing value case

[4]SCED for the implicit information case

entries. This is a heuristic and it could instead be fine tuned by cross validation. Lastly, no regularization has been used in SCED, for a fair comparison, i.e., $\lambda_n = 0, n \in [1, N]$.

All experiments are replicated 10 times on a desktop with Intel i7 processors, 16 GB RAM and Matlab 2016b. The reported results are averaged over these 10 runs.

5.5.2 On Synthetic Datasets

Static Results

The effectiveness of decomposing synthetic tensors under the static setting is presented in Fig. 5.2. Since the efficiency is only related to the number of non-zeros and has no linkage to the types of data, we summarize all efficiency result in Fig. 5.3.

In most of cases, our proposal, SCED (denoted by solid lines), shows better decomposition quality, compared to baselines (denoted by dashed lines). Specifically, for SYN-TO and SYN-MV datasets, although baselines yield acceptable results, significant improvements can be found in our method. For SYN-II, all methods that treat zeros as equally important to observations show poor performance. In contrast, slightly better performance can be found in WOPT, which ignores zero entries. Even so, it is still outperformed by our approach by a large margin, where SCED-m achieves around 0.5 in fitness score, while the fitness of WOPT is lower than 0.1. Both SCED-i and iTALS show the best performance. However, it can be seen that iTALS is much slower than our proposal.

In terms of efficiency, ALS and MU share similar performance to each other, while they are nearly twice as slow as our algorithm. Significant time consumption can be observed in WOPT, due to its internal line search procedure.

Dynamic Results

In this part of experiment, the training set is decomposed by the batch algorithm for initialization: ALS for SYN-TO, WOPT for SYN-MV and batch SCED-i for SYN-II datasets, respectively. With the best seed, the dynamic set is learned and the results are reported in Fig. 5.4.

It is clear that for the SYN-TO dataset, our proposed method achieves perfect overlapping with ALS, which is a batch algorithm that does optimization on all observations to time. This verifies the usefulness of the proposed dynamic updating schema. In terms of efficiency, on average, our method speeds up the processing time for one new entry by around 170 times (~ 0.09 second in ALS v.s. $\sim 5 \times 10^{-4}$ second in SCED-t).

Even greater speed-up can be observed for SYN-MV, where SCED-m ($\sim 5.5 \times 10^{-4}$ second) is more than 1400 times faster than WOPT (~ 0.8 second). More importantly, better decomposition quality is also shown in our algorithm, which confirmed its superior performance compared to WOPT, in both static and dynamic settings.

With respect to SYN-II dataset, both SCED-i and iTALS are capable to dynamically track the new CP decomposition when new entries fed in. Similar to SYN-TO, perfect overlapping can be found while our method is three times faster than iTALS.

5.5.3 On Real-world Datasets

Static Results

The performance comparison on real-world datasets can be found in Fig. 5.5. Among all datasets, the smallest RMSEs are always obtained by SCED-i and iTALS, which means that giving small weight to zero entries is more likely to yield better understanding of the data, compared to the other two extreme cases that either totally ignore them or treat them equally to observations. TO-based methods ($\alpha = 1$) produce poor results in general. However, within this group, we still see better results from our algorithm.

For example, one can find that SCED-nn shows much better convergence, compared to its competitor for the same task, MU. Similar to the results on synthetic datasets, significant improvement can be seen between SCED-m and WOPT in MovieLens and LastFM. On the other hand, we notice a considerable performance drop by them on MathOverflow. One reason behind this might be that their optimizations focus on non-zeros only, while MathOverflow is much more sparse than other two datasets. As a result, their CP decompositions produced may be biased to known entries, which could be considered as overfitting. One possible solution to address this issue is to add regularizations into the objectives, which can be easily handled by our proposal (e.g., set λ to 0.1 with ϕ as l_2-norm), but there is no clear way to adopt WOPT for this case.

Dynamic Results

As mentioned in experiment setup, only SCED-i and iTALS have been chosen for dynamic updating for real-world datasets, due to their good static performance and the inefficiency of other batch methods for large-scale datasets. We report the performance comparison in Fig. 5.6 and Table 5.4. Additionally, to validate whether such dynamic learning is truly effective, we also decompose each tensor with batch SCED-i algorithm before (80% training) and after (80% training + 10% dynamic) the dynamic learning phase, as reference points. Specifically, the maximum number of iterations for batch method is set to 200 and the decomposition generated before feeding the dynamic set is used as the seed for hot start.

As can be seen from Fig. 5.6, both SCED-i and iTALS can effectively track the decompositions when new data arrives, while our algorithm is around 3 times faster than iTALS. In terms of efficiency of the batch method, it takes 320, 475 and 600 seconds to decompose the training sets of MovieLens, LastFM and MathOverflow, respectively. And roughly speaking, an extra 30 seconds is needed for processing the additional dynamic sets. Such high expense makes the batch method infeasible to be applied to a highly dynamic system where new data is arriving at high velocity. Unlike the batch algorithm that can only process one state of data at high time cost, dynamic algorithms can keep tracking decompositions of all intermediate states at significantly lower cost. This means the proposed algorithm can efficiently and easily refresh the decomposition model to date, to provide better service than batch techniques, where the most up-to-date decomposition is not always available.

5.5.4 Scalability

As shown in §5.4.5, theoretically, SCED is R times faster than iTALS. Previous experiments partially confirm this analysis. However, the observed speed-up is only around 3 to 5 times. This is because that R has been set to 20, which is too small to see the trend. As a result, to evaluate the performance of SCED and iTALS on large-scale datasets, a scalability test is performed w.r.t. three key features: number of non-zeros, decomposition rank and tensor size.

Specifically, random tensors of size $1000 \times 1000 \times 1000$ with $R = 20$ are decomposed and the result can is in Fig. 5.7a. The number of non-zeros varies from 1×10^4 to 1×10^8. Similar evaluation has been done for analyzing the scalability w.r.t. R (Fig. 5.7b, R varies from 2 to 128, by fixing $nnz = 1 \times 10^4$ and size as $1000 \times 1000 \times 1000$) and the tensor size (Fig. 5.7c, cardinality of a mode varies from 100 to 1000, with $R = 20$ and $nnz = 1 \times 10^4$). The reported results are average running time for one iteration under the batch setting, while it is clear that similar trends can be seen for the dynamic case.

5.5.5 Highlights of Results

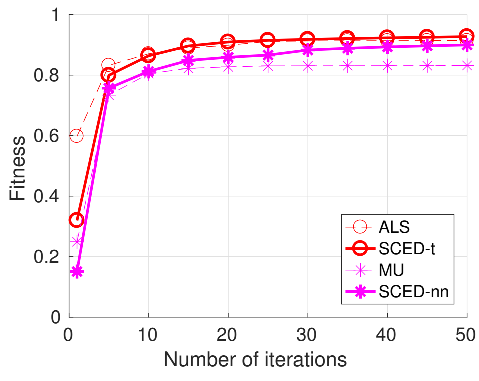
We highlight some key findings as follows.

- The proposed SCED algorithm uniformly produces best or close-to-best quality decompositions on different types of data, across both static or dynamic settings. Two major improvements can be found in SCED-m v.s. WOPT, and SCED-nn v.s. MU.

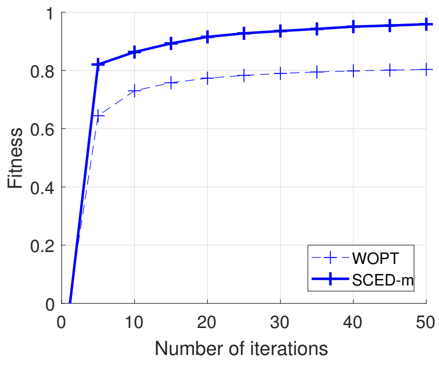
- The efficiency of SCED is similar to ALS and MU. While our method is significantly faster than WOPT and iTALS, spanning all types of data and settings.
- The proposed dynamic learning method is highly effective and usually demonstrates comparable or even better results to batch methods. While our method reduces time cost by orders of magnitude.
- Compared to iTALS, our method is more suitable to be applied to large-scale data since better scalability is shown w.r.t. number of non-zeros, decomposition rank and tensor size.

5.6 Summary

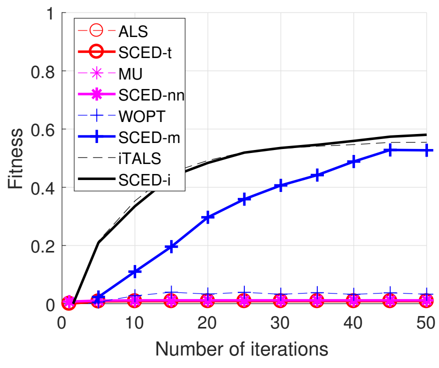
To conclude, in this chapter, we addressed the problem of finding the CP decomposition of sparse tensors. An efficient algorithm, SCED, is proposed, which has linear time complexity w.r.t. the number of non-zeros and decomposition rank. In addition, our framework is also flexible to handle constraints such as non-negativity and regularizations like ℓ_1 and ℓ_2 norms. Finally, a dynamic learning algorithm is also proposed to tackle dynamic tensors that have new data being updated at the element-level. As evaluated on both synthetic and real-world datasets, under both static and dynamic settings, our algorithm demonstrates high performance in terms of effectiveness, efficiency and scalability, compared to state-of-the-art approaches.



(a) TO



(b) MV



(c) II

Figure 5.2: Effectiveness of static decomposition on synthetic datasets

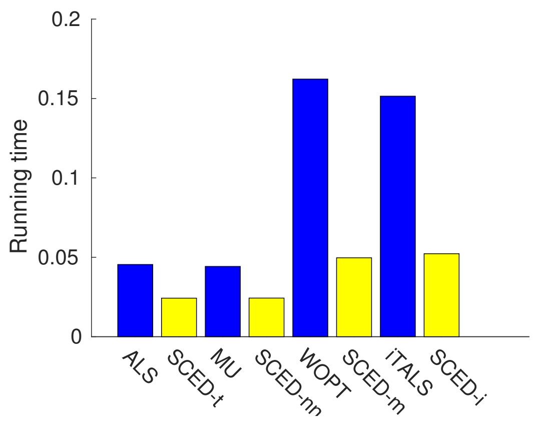
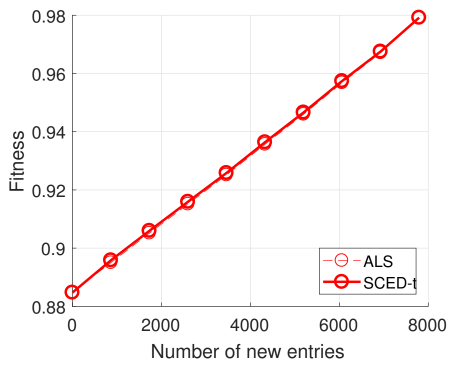
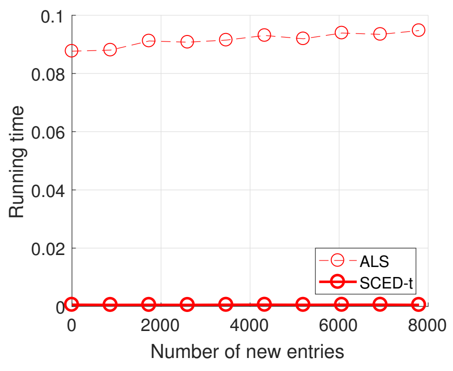


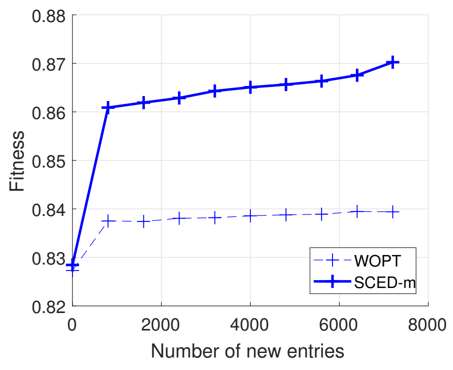
Figure 5.3: Efficiency of static decomposition on synthetic datasets



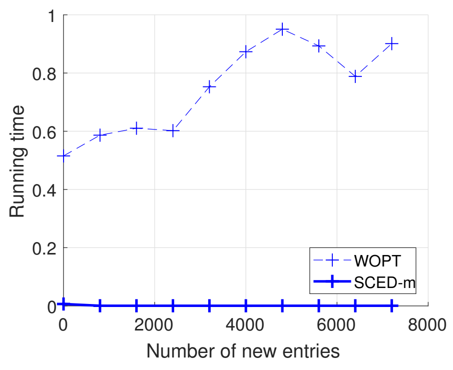
(a) TO-Effectiveness



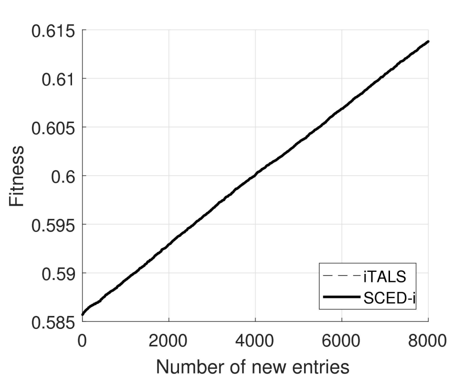
(b) TO-Efficiency



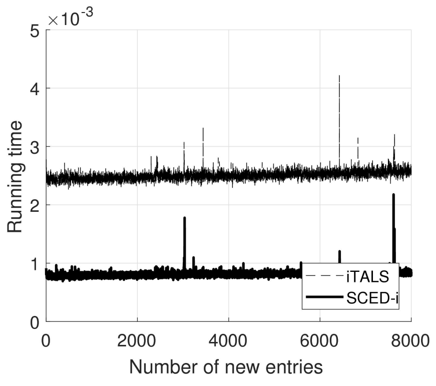
(c) MV-Effectiveness



(d) MV-Efficiency

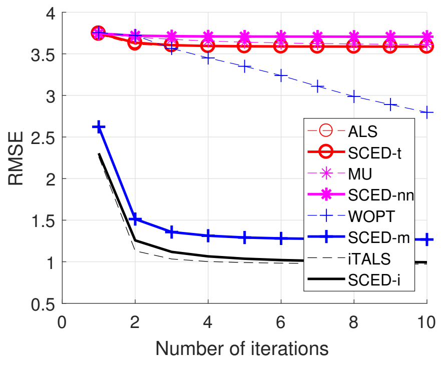


(e) II-Effectiveness

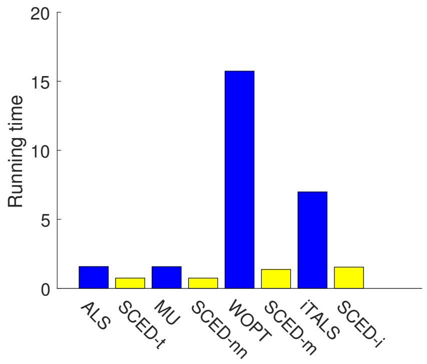


(f) II-Efficiency

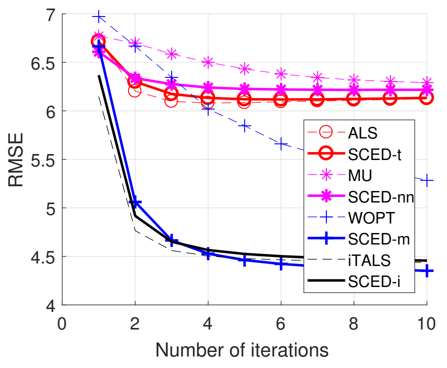
Figure 5.4: Dynamic decomposition on synthetic datasets



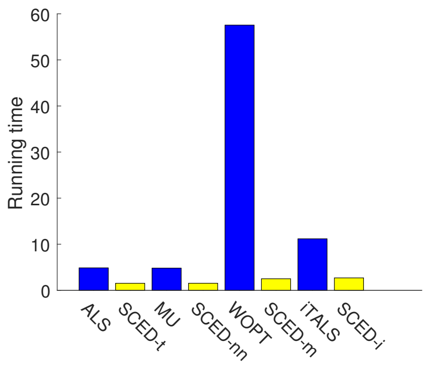
(a) MovieLens-Effectiveness



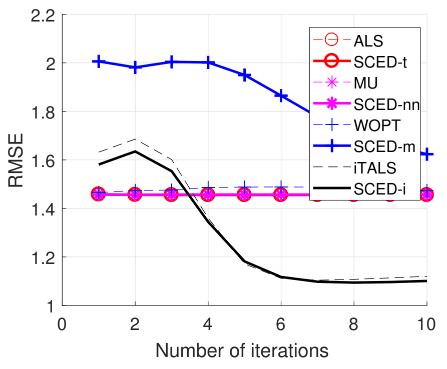
(b) MovieLens-Efficiency



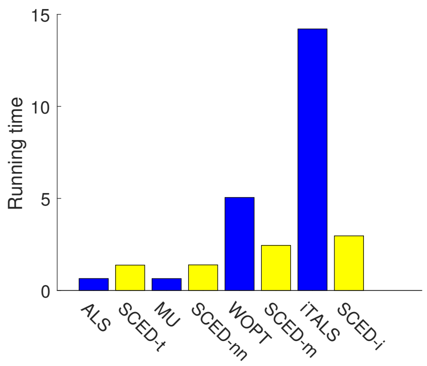
(c) LastFM-Effectiveness



(d) LastFM-Efficiency

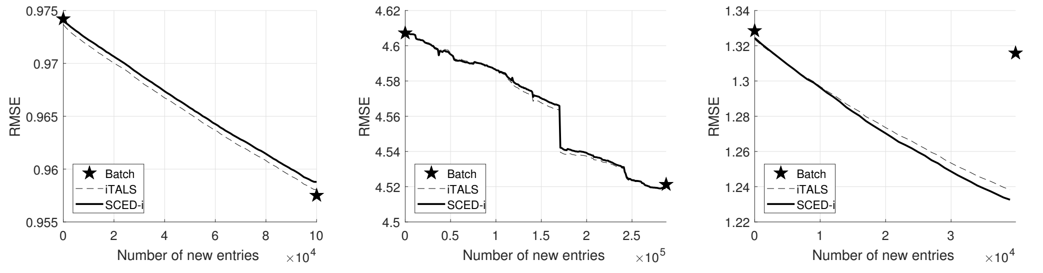


(e) MathOverflow-Effectiveness



(f) MathOverflow-Efficiency

Figure 5.5: Static decomposition on real-world datasets

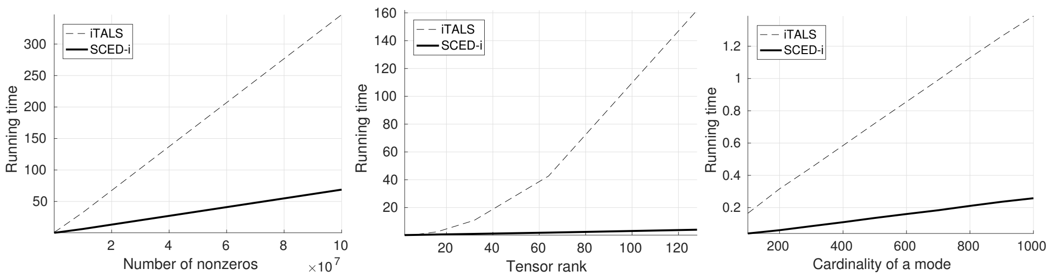


(a) MovieLens (b) LastFM (c) MathOverflow

Figure 5.6: Dynamic decomposition on real-world datasets

Table 5.4: Average running time to process one new entry on real-world datasets. Values in parentheses are number of entries can be processed per minutes

dataset	iTALS	SCED-i
MovieLens	0.0255 (2356)	0.0094 (6359)
LastFM	0.0484 (1241)	0.0185 (3244)
MathOverflow	0.0091 (6581)	0.0081 (7408)



(a) $nnz \in [1e4, 1e8]$ (b) $R \in [2, 128]$ (c) $I \in [100, 1000]$

Figure 5.7: Scalability comparison of SCED to iTALS

Chapter 6

Conclusion

In this thesis, we focused on developing efficient and scalable algorithms for decomposing dynamic tensors that are constantly changing over time. In particular, we studied three types of dynamic tensors: 1) dense dynamic tensors with slice-wise updates, 2) sparse dynamic tensors with slice-wise updates, and 3) sparse dynamic tensors with element-wise updates. We summarize the contributions of this thesis, discuss the limitations of our work and outline some potential directions of future works as follows.

6.1 Summary of Contributions

In Chapter 3, we investigated on effectively refreshing the existing decomposition when new data slices are appended to a dense tensor on its time mode. Based on our observation of the structure of online tensors, we suggested an incremental updating schema that dividing the computation into historical and new data parts, where historical information was stored by complimentary matrices. This reduced the computational cost to a constant that only depends on the size of the new data chunk. Furthermore, as the first adaptively method that is applicable to higher-order tensors, we designed a dynamic programming algorithm to avoid duplicated calculations by caching intermediate results. As validated by experiment on real-world and synthetic datasets, our algorithm, OnlineCP, demonstrated comparable results with the most accurate algorithm, ALS, whilst being computationally much more efficient. Specifically, on small and moderate datasets, our approach was tens to hundreds of times faster than ALS, while for large-scale datasets, the speedup can be more than 3,000 times. Compared to other state-of-the-art online approaches, our method showed not only significantly better decomposition quality, but also better performance in terms of stability, efficiency and scalability.

In Chapter 4, we focused on addressing the efficiency issue of previous algorithm on sparse data. Specifically, we developed a new algorithm that is fully optimized with keeping data sparsity in mind and successively reduced the time complexity from linear w.r.t. the size of new data, to linear w.r.t. the number of non-zeros in the new data. In addition, to further control the memory-usage, we redesigned the complimentary matrices. Overall, Experiments on nine real-world datasets showed that when most of the existing online methods failed to give results given a limited amount of resources, the decomposition produced by our algorithm is of high quality, whilst at the same time achieving speed improvements of up to 250 times and 100 times less memory, compared to ALS.

In Chapter 5, we studied sparse tensor decomposition from three aspects: *(i)* the roles of zero entries under different application scenarios, *(ii)* the constrained CP decomposition, and *(iii)* the element-wise dynamic updates observed at arbitrary positions in the tensors. We showed that this can be modeled in a generalized weighted decomposition formulation and solved by a parameter by parameter learning schema, with the flexibility to incorporate constraints and to incrementally learn new data entries in an efficient manner. Extensive experiment had been conducted to show the merits of our algorithm, compared to current approaches.

6.2 Future Directions

Our studies presented in this thesis addressed the dynamic learning aspect of several typical types of dynamic tensors. However, there are still many interesting and potential extensions for future research.

1. Chapters 3 and 4 only discussed the dynamic aspect of online tensors, while there was no constrained decomposition has been explored. Imposing Tikhonov regularization (*i.e.*, Frobenius norms of loading matrices) is readily easy by appending it as penalty terms to the objective and this is also easy to solve since the regularization is differentiable. However, non-negativity is not a simple constraint to incorporate under the current ALS-like algorithms. Even though solvers and algorithms like multiplicative update rule exist for static cases, whether they applicable to the dynamic setting is still a problem that deserved further research. In addition, there are more constraints like orthogonality [Kolda, 2001], graph regularization [Cai et al., 2011], and smoothness [Yokota et al., 2016] that are commonly used in real-world problems but not discussed in the thesis. It is of great interest and potential to study these constraints under the incremental learning framework proposed in this thesis.

2. For the incremental algorithms proposed for dynamic tensors with slice-wise updates, the main speedup we have gained is by decomposing the computation of MTTKRP into historical and new data parts, which limited the complexities of the proposed algorithm to linear w.r.t. the new data. In fact, this can be further improved by more advanced computing technologies such as distributed computing with MapReduce, or parallel accelerations with GPUs. However, it should be noticed that whether these approaches useful depends on the scale of the data and technical implementation, since the improvement gained by these methods will be discounted by the extra computation overhead involved, like the communication cost for distributed computing. Additionally, methods proposed for enhancing the efficiency of static ALS algorithms (*e.g.*, line-search, subsampling, random projection, as introduced in §2.3.2) are potentially applicable to our algorithms as well, which can be beneficial.

3. Two types of dynamic updates have been studied in this thesis: slice-

wise and element-wise updates, since they have been commonly seen in real-world problems. However, more types of dynamic patterns can be found in various datasets and applications. For example, the dynamic tensors may expanded in multiple modes simultaneously, elements in existing tensors may be removed or updated, not just as the addition case discussed in Chapter 5. Another example is that in many applications such as recommendation system, sometimes we may have element-wise (*e.g.* new ratings) and slice-wise updates (*e.g.* new users and items) simultaneously. Towards this end, it would be interesting to see that how to apply and adapter the insights we have gained throughout the studies in this thesis to those cases. Furthermore, we consider the parameter by parameter learning rules proposed in Chapter 5 as a more flexible learning algorithm that is of great potential to be applied to dynamic tensors in general. For example, slice-wise updates can be viewed as appending the elements in a tensor slice one by one in a short time period, removing or updating the an existing value can be seen as adding a negative value or the difference between the previous and current values in the same position. As a result, It would be interesting to extend our algorithm to a more general framework for more dynamic cases, while it should be noted that the choice of the updating order may be a factor that needs to be considered, in order to achieve good efficiency.

4. In this thesis, we focused on only the tensor decomposition problem. While in real-world applications, tensor decomposition is usually just one step of the whole analysis pipeline and practitioners often apply tensor decomposition as a pre-processing tool for tensor data, and further analyses such as clustering, classification and forecasting, are performed based on the produced loading matrices. Contrast to this step by step approach, there are some approaches in the literature that modeled the decomposition and learning phases into a unified objective functions and obtained better performance [Xu et al., 2016, Xu et al., 2018]. As a result, it would be useful to explore the ideas presented in this thesis in such problem formulation in order to better

efficiency and scalability.

5. Finally, in this thesis we assumed that all dynamic update in a tensor are equally important and the current algorithm have not taken the freshness of the data into consideration. However, in real-world applications, the recent data can play more important roles than the data received a while ago, since the new data contains more useful information with the latest status of the system. This can be modeled by introducing a forgetting factors to the proposed methods in our studies, while how to design such forgetting strategy requires additional investigations.

In conclusion, we studied dynamic tensor learning problem in this thesis. We hope that the ideas and insights we have presented will contribute the advance of the field and lay fruitful foundations for future researches.

Bibliography

[Abdallah et al., 2007] Abdallah, E. E., Hamza, A. B., and Bhattacharya, P. (2007). Mpeg video watermarking using tensor singular value decomposition. In *International Conference Image Analysis and Recognition*, pages 772–783. Springer.

[Abdi and Williams, 2010] Abdi, H. and Williams, L. J. (2010). Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459.

[Acar et al., 2007] Acar, E., Aykut-Bingol, C., Bingol, H., Bro, R., and Yener, B. (2007). Multiway analysis of epilepsy tensors. *Bioinformatics*, 23(13):i10–i18.

[Acar et al., 2015] Acar, E., Bro, R., and Smilde, A. K. (2015). Data fusion in metabolomics using coupled matrix and tensor factorizations. *Proceedings of the IEEE*, 103(9):1602–1620.

[Acar et al., 2005] Acar, E., Çamtepe, S. A., Krishnamoorthy, M. S., and Yener, B. (2005). Modeling and multiway analysis of chatroom tensors. In *International Conference on Intelligence and Security Informatics*, pages 256–268. Springer.

[Acar et al., 2011a] Acar, E., Dunlavy, D. M., Kolda, T. G., and Mørup, M. (2011a). Scalable tensor factorizations for incomplete data. *Chemometrics and Intelligent Laboratory Systems*, 106(1):41–56.

[Acar et al., 2011b] Acar, E., Kolda, T. G., and Dunlavy, D. M. (2011b). All-at-once optimization for coupled matrix and tensor factorizations. *arXiv preprint arXiv:1105.3422*.

[Acar et al., 2013] Acar, E., Rasmussen, M. A., Savorani, F., Næs, T., and Bro, R. (2013). Understanding data fusion within the framework of coupled matrix and tensor factorizations. *Chemometrics and Intelligent Laboratory Systems*, 129:53–63.

[Agrawal et al., 2015a] Agrawal, R., Golshan, B., and Papalexakis, E. (2015a). A study of distinctiveness in web results of two search engines. In *Proceedings of the 24th International Conference on World Wide Web*, pages 267–273. ACM.

[Agrawal et al., 2015b] Agrawal, R., Golshan, B., and Papalexakis, E. (2015b). Whither social networks for web search? In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1661–1670. ACM.

[Altun et al., 2010] Altun, K., Barshan, B., and Tunçel, O. (2010). Comparative study on classifying human activities with miniature inertial and magnetic sensors. *Pattern Recognition*, 43(10):3605–3620.

[Anandkumar et al., 2014] Anandkumar, A., Ge, R., Hsu, D. J., and Kakade, S. M. (2014). A tensor approach to learning mixed membership community models. *Journal of Machine Learning Research*, 15(1):2239–2312.

[Andersen and Rayens, 2004] Andersen, A. H. and Rayens, W. S. (2004). Structure-seeking multilinear methods for the analysis of fmri data. *NeuroImage*, 22(2):728–739.

[Andersen and Bro, 2003] Andersen, C. M. and Bro, R. (2003). Practical aspects of parafac modeling of fluorescence excitation-emission data. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 17(4):200–215.

[Appellof and Davidson, 1981] Appellof, C. J. and Davidson, E. R. (1981). Strategies for analyzing data from video fluorometric monitoring of liquid chromatographic effluents. *Analytical Chemistry*, 53(13):2053–2056.

[Araujo et al., 2014] Araujo, M., Papadimitriou, S., Günnemann, S., Faloutsos, C., Basu, P., Swami, A., Papalexakis, E. E., and Koutra, D. (2014). Com2: fast automatic discovery of temporal (comet) communities. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 271–283. Springer.

[Bachlin et al., 2010] Bachlin, M., Plotnik, M., Roggen, D., Maidan, I., Hausdorff, J. M., Giladi, N., and Troster, G. (2010). Wearable assistant for parkinsons disease patients with the freezing of gait symptom. *IEEE Transactions on Information Technology in Biomedicine*, 14(2):436–446.

[Bader et al., 2008] Bader, B. W., Berry, M. W., and Browne, M. (2008). Discussion tracking in enron email using parafac. In *Survey of Text Mining II*, pages 147–163. Springer.

[Bader et al., 2007] Bader, B. W., Harshman, R. A., and Kolda, T. G. (2007). Temporal analysis of semantic graphs using asalsan. In *icdm*, pages 33–42. IEEE.

[Bader and Kolda, 2006] Bader, B. W. and Kolda, T. G. (2006). Algorithm 862: Matlab tensor classes for fast algorithm prototyping. *ACM Transactions on Mathematical Software (TOMS)*, 32(4):635–653.

[Bader and Kolda, 2007] Bader, B. W. and Kolda, T. G. (2007). Efficient matlab computations with sparse and factored tensors. *SIAM Journal on Scientific Computing*, 30(1):205–231.

[Bader et al., 2015] Bader, B. W., Kolda, T. G., et al. (2015). Matlab tensor toolbox version 2.6. Available online.

[Ballani et al., 2013] Ballani, J., Grasedyck, L., and Kluge, M. (2013). Black box approximation of tensors in hierarchical tucker format. *Linear algebra and its applications*, 438(2):639–657.

[Battaglino et al., 2018] Battaglino, C., Ballard, G., and Kolda, T. G. (2018). A practical randomized cp tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, 39(2):876–901.

[Ben-Israel and Greville, 2003] Ben-Israel, A. and Greville, T. N. (2003). *Generalized inverses: theory and applications*, volume 15. Springer Science & Business Media.

[Beutel et al., 2014] Beutel, A., Talukdar, P. P., Kumar, A., Faloutsos, C., Papalexakis, E. E., and Xing, E. P. (2014). Flexifact: Scalable flexible factorization of coupled tensors on hadoop. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 109–117. SIAM.

[Bro, 1997] Bro, R. (1997). Parafac. tutorial and applications. *Chemometrics and intelligent laboratory systems*, 38(2):149–171.

[Bro, 1998] Bro, R. (1998). Multi-way analysis in the food industry. *Models, Algorithms, and Applications. Academish proefschrift. Dinamarca.*

[Bro et al., 1999] Bro, R., Andersson, C. A., and Kiers, H. A. (1999). Parafac2part ii. modeling chromatographic data with retention time shifts. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 13(3-4):295–309.

[Bro and De Jong, 1997] Bro, R. and De Jong, S. (1997). A fast non-negativity-constrained least squares algorithm. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 11(5):393–401.

[Cai et al., 2011] Cai, D., He, X., Han, J., and Huang, T. S. (2011). Graph regularized nonnegative matrix factorization for data representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1548–1560.

[Cai et al., 2015] Cai, Y., Tong, H., Fan, W., Ji, P., and He, Q. (2015). Facets: Fast comprehensive mining of coevolving high-order time series. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 79–88. ACM.

[Carlson et al., 2010] Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka Jr, E. R., and Mitchell, T. M. (2010). Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3. Atlanta.

[Carroll and Chang, 1970] Carroll, J. D. and Chang, J.-J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. *Psychometrika*, 35(3):283–319.

[Celma, 2010] Celma, O. (2010). *Music Recommendation and Discovery in the Long Tail*. Springer.

[Chang et al., 2013] Chang, K.-W., Yih, W.-t., and Meek, C. (2013). Multi-relational latent semantic analysis. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1602–1612.

[Chang et al., 2014] Chang, K.-W., Yih, W.-t., Yang, B., and Meek, C. (2014). Typed tensor decomposition of knowledge bases for relation extraction. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1568–1579.

[Cheng et al., 2016] Cheng, D., Peng, R., Liu, Y., and Perros, I. (2016). Spals: Fast alternating least squares via implicit leverage scores sampling. In *Advances In Neural Information Processing Systems*, pages 721–729.

[Chew et al., 2007] Chew, P. A., Bader, B. W., Kolda, T. G., and Abdelali, A. (2007). Cross-language information retrieval using parafac2. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 143–152. ACM.

[Chi and Kolda, 2012] Chi, E. C. and Kolda, T. G. (2012). On tensors, sparsity, and nonnegative factorizations. *SIAM Journal on Matrix Analysis and Applications*, 33(4):1272–1299.

[Choi et al., 2017] Choi, D., Jang, J.-G., and Kang, U. (2017). Fast, accurate, and scalable method for sparse coupled matrix-tensor factorization. *arXiv preprint arXiv:1708.08640*.

[Choi and Vishwanathan, 2014] Choi, J. H. and Vishwanathan, S. (2014). Dfacto: Distributed factorization of tensors. In *Advances in Neural Information Processing Systems*, pages 1296–1304.

[Cichocki, 2014] Cichocki, A. (2014). Era of big data processing: A new approach via tensor networks and tensor decompositions. *arXiv preprint arXiv:1403.2048*.

[Cichocki and Anh-Huy, 2009] Cichocki, A. and Anh-Huy, P. (2009). Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 92(3):708–721.

[Cichocki et al., 2015] Cichocki, A., Mandic, D., De Lathauwer, L., Zhou, G., Zhao, Q., Caiafa, C., and Phan, H. A. (2015). Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *IEEE Signal Processing Magazine*, 32(2):145–163.

[Comon, 2014] Comon, P. (2014). Tensors: a brief introduction. *IEEE Signal Processing Magazine*, 31(3):44–53.

[Comon et al., 2009] Comon, P., Luciani, X., and De Almeida, A. L. (2009). Tensor decompositions, alternating least squares and other tales. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 23(7-8):393–405.

[Coppi and Bolasco, 1988] Coppi, R. and Bolasco, S. (1988). Rank decomposition and uniqueness for 3-way and n-way arrays. In *Multiway data analysis*, pages 7–18. North-Holland.

[Davidson et al., 2013] Davidson, I., Gilpin, S., Carmichael, O., and Walker, P. (2013). Network discovery via constrained tensor analysis of fmri data. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 194–202. ACM.

[De Lathauwer, 2008] De Lathauwer, L. (2008). Decompositions of a higher-order tensor in block termspart ii: Definitions and uniqueness. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1033–1066.

[De Lathauwer et al., 2000] De Lathauwer, L., De Moor, B., and Vandewalle, J. (2000). On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors. *SIAM journal on Matrix Analysis and Applications*, 21(4):1324–1342.

[De Lathauwer and Vandewalle, 2004] De Lathauwer, L. and Vandewalle, J. (2004). Dimensionality reduction in higher-order signal processing and rank-(r1, r2,, rn) reduction in multilinear algebra. *Linear Algebra and its Applications*, 391:31–55.

[Deerwester et al., 1990] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407.

[Devooght et al., 2015] Devooght, R., Kourtellis, N., and Mantrach, A. (2015). Dynamic matrix factorization with priors on unknown values. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 189–198. ACM.

[Du et al., 2018] Du, Y., Zheng, Y., Lee, K.-c., and Zhe, S. (2018). Probabilistic streaming tensor decomposition. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 99–108. IEEE.

[Dunlavy et al., 2011] Dunlavy, D. M., Kolda, T. G., and Acar, E. (2011). Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(2):10.

[Erdős and Miettinen, 2013] Erdős, D. and Miettinen, P. (2013). Scalable boolean tensor factorizations using random walks. *arXiv preprint arXiv:1310.4843*.

[Ermiş et al., 2015] Ermiş, B., Acar, E., and Cemgil, A. T. (2015). Link prediction in heterogeneous data via generalized coupled tensor factorization. *Data Mining and Knowledge Discovery*, 29(1):203–236.

[Field and Graupe, 1991] Field, A. S. and Graupe, D. (1991). Topographic component (parallel factor) analysis of multichannel evoked potentials:

practical issues in trilinear spatiotemporal decomposition. *Brain topography*, 3(4):407–423.

[FitzGerald et al., 2005] FitzGerald, D., Cranitch, M., and Coyle, E. (2005). Non-negative tensor factorisation for sound source separation.

[Fonollosa et al., 2015] Fonollosa, J., Sheik, S., Huerta, R., and Marco, S. (2015). Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring. *Sensors and Actuators B: Chemical*, 215:618–629.

[Friedlander and Hatz, 2008] Friedlander, M. P. and Hatz, K. (2008). Computing non-negative tensor factorizations. *Optimisation Methods and Software*, 23(4):631–647.

[Frolov and Oseledets, 2017] Frolov, E. and Oseledets, I. (2017). Tensor methods and recommender systems. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(3):e1201.

[Globerson et al., 2007] Globerson, A., Chechik, G., Pereira, F., and Tishby, N. (2007). Euclidean Embedding of Co-occurrence Data. *The Journal of Machine Learning Research*, 8:2265–2295.

[Grasedyck, 2010] Grasedyck, L. (2010). Hierarchical singular value decomposition of tensors. *SIAM Journal on Matrix Analysis and Applications*, 31(4):2029–2054.

[Grasedyck et al., 2013] Grasedyck, L., Kressner, D., and Tobler, C. (2013). A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78.

[Gujral et al., 2018a] Gujral, E., Pasricha, R., and Papalexakis, E. E. (2018a). Sambaten: Sampling-based batch incremental tensor decomposition. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 387–395. SIAM.

[Gujral et al., 2018b] Gujral, E., Pasricha, R., Yang, T., and Papalexakis, E. E. (2018b). Octen: Online compression-based tensor decomposition. *arXiv preprint arXiv:1807.01350*.

[Hackbusch, 2012] Hackbusch, W. (2012). *Tensor spaces and numerical tensor calculus*, volume 42. Springer Science & Business Media.

[Hackbusch and Kühn, 2009] Hackbusch, W. and Kühn, S. (2009). A new scheme for the tensor representation. *Journal of Fourier analysis and applications*, 15(5):706–722.

[Hansen et al., 2015] Hansen, S., Plantenga, T., and Kolda, T. G. (2015). Newton-based optimization for kullback–leibler nonnegative tensor factorizations. *Optimization Methods and Software*, 30(5):1002–1029.

[Harper and Konstan, 2016] Harper, F. M. and Konstan, J. A. (2016). The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19.

[Harshman and Lundy, 1992] Harshman, R. and Lundy, M. (1992). Three-way dedicom: Analyzing multiple matrices of asymmetric relationships. In *Annual Meeting of the North American Psychometric Society*.

[Harshman, 1970] Harshman, R. A. (1970). Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis.

[Harshman, 1972] Harshman, R. A. (1972). Parafac2: Mathematical and technical notes. *UCLA working papers in phonetics*, 22(3044):122215.

[Harshman, 1978] Harshman, R. A. (1978). Models for analysis of asymmetrical relationships among n objects or stimuli. In *First Joint Meeting of the Psychometric Society and the Society of Mathematical Psychology, Hamilton, Ontario, 1978*.

[Håstad, 1990] Håstad, J. (1990). Tensor rank is np-complete. *Journal of Algorithms*, 11(4):644–654.

[He et al., 2014] He, L., Kong, X., Yu, P. S., Yang, X., Ragin, A. B., and Hao, Z. (2014). Dusk: A dual structure-preserving kernel for supervised tensor learning with applications to neuroimages. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 127–135. SIAM.

[He et al., 2016] He, X., Zhang, H., Kan, M.-Y., and Chua, T.-S. (2016). Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 549–558. ACM.

[Henrion, 1994] Henrion, R. (1994). N-way principal component analysis theory, algorithms and applications. *Chemometrics and intelligent laboratory systems*, 25(1):1–23.

[Herrada, 2009] Herrada, O. C. (2009). *Music recommendation and discovery in the long tail*. PhD thesis, Universitat Pompeu Fabra.

[Hidasi and Tikk, 2012] Hidasi, B. and Tikk, D. (2012). Fast als-based tensor factorization for context-aware recommendation from implicit feedback. *Machine Learning and Knowledge Discovery in Databases*, pages 67–82.

[Hitchcock, 1927] Hitchcock, F. L. (1927). The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189.

[Ho et al., 2014] Ho, J. C., Ghosh, J., and Sun, J. (2014). Marble: high-throughput phenotyping from electronic health records via sparse non-negative tensor factorization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 115–124. ACM.

[Hu et al., 2011] Hu, W., Li, X., Zhang, X., Shi, X., Maybank, S., and Zhang, Z. (2011). Incremental tensor subspace learning and its applications to foreground segmentation and tracking. *International Journal of Computer Vision*, 91(3):303–327.

[Hu et al., 2008] Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. Ieee.

[Huang et al., 2016] Huang, K., Sidiropoulos, N. D., and Liavas, A. P. (2016). A flexible and efficient algorithmic framework for constrained matrix and tensor factorization. *IEEE Transactions on Signal Processing*, 64(19):5052–5065.

[Jeon et al., 2016] Jeon, B., Jeon, I., Sael, L., and Kang, U. (2016). Scout: Scalable coupled matrix-tensor factorization-algorithm and discoveries. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 811–822. IEEE.

[Jeon et al., 2015] Jeon, I., Papalexakis, E. E., Kang, U., and Faloutsos, C. (2015). Haten2: Billion-scale tensor decompositions. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 1047–1058. IEEE.

[Jiang et al., 2014] Jiang, M., Cui, P., Wang, F., Xu, X., Zhu, W., and Yang, S. (2014). Fema: flexible evolutionary multi-faceted analysis for dynamic behavioral pattern discovery. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1186–1195. ACM.

[Kamstrup-Nielsen et al., 2013] Kamstrup-Nielsen, M. H., Johnsen, L. G., and Bro, R. (2013). Core consistency diagnostic in parafac2. *Journal of Chemometrics*, 27(5):99–105.

[Kang et al., 2012] Kang, U., Papalexakis, E., Harpale, A., and Faloutsos, C. (2012). Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 316–324. ACM.

[Karatzoglou et al., 2010] Karatzoglou, A., Amatriain, X., Baltrunas, L., and Oliver, N. (2010). Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 79–86. ACM.

[Khan et al., 2016] Khan, S. A., Leppäaho, E., and Kaski, S. (2016). Bayesian multi-tensor factorization. *Machine Learning*, 105(2):233–253.

[Kiers and Mechelen, 2001] Kiers, H. A. and Mechelen, I. V. (2001). Three-way component analysis: Principles and illustrative application. *Psychological methods*, 6(1):84.

[Kim et al., 2014] Kim, J., He, Y., and Park, H. (2014). Algorithms for nonnegative matrix and tensor factorizations: A unified view based on block coordinate descent framework. *Journal of Global Optimization*, 58(2):285–319.

[Kleinberg, 1999] Kleinberg, J. M. (1999). Hubs, authorities, and communities. *ACM computing surveys (CSUR)*, 31(4es):5.

[Kolda, 2001] Kolda, T. G. (2001). Orthogonal tensor decompositions. *SIAM Journal on Matrix Analysis and Applications*, 23(1):243–255.

[Kolda and Bader, 2009] Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM review*, 51(3):455–500.

[Kolda et al., 2005] Kolda, T. G., Bader, B. W., and Kenny, J. P. (2005). Higher-order web link analysis using multilinear algebra. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE.

[Kolda and Sun, 2008] Kolda, T. G. and Sun, J. (2008). Scalable tensor decompositions for multi-aspect data mining. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 363–372. IEEE.

[Koren, 2008] Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM.

[Kressner and Tobler, 2012] Kressner, D. and Tobler, C. (2012). htuckera matlab toolbox for tensors in hierarchical tucker format. *Mathicse, EPF Lausanne*.

[Kruskal, 1977] Kruskal, J. B. (1977). Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear algebra and its applications*, 18(2):95–138.

[Kunegis, 2013] Kunegis, J. (2013). Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350. ACM.

[Lafayette and Wiebelt, 2017] Lafayette, L. and Wiebelt, B. (2017). Spartan and nemo: Two hpc-cloud hybrid implementations. In *e-Science (e-Science), 2017 IEEE 13th International Conference on*, pages 458–459. IEEE.

[Lee and Seung, 2001] Lee, D. D. and Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562.

[Lee et al., 2007] Lee, H., Battle, A., Raina, R., and Ng, A. Y. (2007). Efficient sparse coding algorithms. In *Advances in neural information processing systems*, pages 801–808.

[Lee, 1998] Lee, T.-W. (1998). Independent component analysis. In *Independent component analysis*, pages 27–66. Springer.

[Li et al., 2017] Li, J., Choi, J., Perros, I., Sun, J., and Vuduc, R. (2017). Model-driven sparse cp decomposition for higher-order tensors. In *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*, pages 1048–1057. IEEE.

[Lin, 2007] Lin, C.-J. (2007). Projected gradient methods for nonnegative matrix factorization. *Neural computation*, 19(10):2756–2779.

[Liu et al., 2017] Liu, B., Wen, C., Sarwate, A. D., and Dehnavi, M. M. (2017). A unified optimization approach for sparse tensor operations on gpus. In *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*, pages 47–57. IEEE.

[Liu et al., 2012a] Liu, J., Liu, J., Wonka, P., and Ye, J. (2012a). Sparse nonnegative tensor factorization using columnwise coordinate descent. *Pattern Recognition*, 45(1):649–656.

[Liu et al., 2005] Liu, N., Zhang, B., Yan, J., Chen, Z., Liu, W., Bai, F., and Chien, L. (2005). Text representation: From vector to tensor. In *Data Mining, Fifth IEEE International Conference on*, pages 4–pp. IEEE.

[Liu et al., 2012b] Liu, W., Chan, J., Bailey, J., Leckie, C., and Kotagiri, R. (2012b). Utilizing common substructures to speedup tensor factorization for mining dynamic graphs. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 435–444. ACM.

[Lu and Yang, 2015] Lu, Y. and Yang, J. (2015). Notes on low-rank matrix factorization. *arXiv preprint arXiv:1507.00333*.

[Lundy et al., 2003] Lundy, M. E., Harshman, R. A., Paatero, P., and Swartzman, L. C. (2003). Application of the 3-way dedicom model to skew-symmetric data for paired preference ratings of treatments for chronic back pain. In *TRICAP 2003 Meeting, Lexington, Kentucky*. Citeseer.

[Ma et al., 2016] Ma, G., He, L., Lu, C.-T., Yu, P. S., Shen, L., and Ragin, A. B. (2016). Spatio-temporal tensor analysis for whole-brain fmri classification. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 819–827. SIAM.

[Ma et al., 2009] Ma, X., Schonfeld, D., and Khokhar, A. (2009). Dynamic updating and downdating matrix svd and tensor hosvd for adaptive indexing and retrieval of motion trajectories. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 1129–1132. IEEE.

[Mairal et al., 2010] Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2010). Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11(Jan):19–60.

[Mao et al., 2014] Mao, H.-H., Wu, C.-J., Papalexakis, E. E., Faloutsos, C., Lee, K.-C., and Kao, T.-C. (2014). Malspot: Multi 2 malicious network behavior patterns analysis. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 1–14. Springer.

[Maruhashi et al., 2011] Maruhashi, K., Guo, F., and Faloutsos, C. (2011). Multiaspectforensics: Pattern mining on large-scale heterogeneous networks with tensor analysis. In *Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on*, pages 203–210. IEEE.

[Miettinen, 2011] Miettinen, P. (2011). Boolean tensor factorizations. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 447–456. IEEE.

[Mislove, 2009] Mislove, A. E. (2009). *Online social networks: measurement, analysis, and applications to distributed information systems*. PhD thesis, Rice University.

[Mitchell et al., 2018] Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Yang, B., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., et al. (2018). Never-ending learning. *Communications of the ACM*, 61(5):103–115.

[Mnih and Salakhutdinov, 2008] Mnih, A. and Salakhutdinov, R. R. (2008). Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264.

[Mocks, 1988] Mocks, J. (1988). Topographic components model for event-related potentials and some biophysical considerations. *IEEE transactions on biomedical engineering*, 35(6):482–484.

[Mørup et al., 2008] Mørup, M., Hansen, L. K., Arnfred, S. M., Lim, L.-H., and Madsen, K. H. (2008). Shift-invariant multilinear decomposition of neuroimaging data. *NeuroImage*, 42(4):1439–1450.

[Mørup et al., 2006] Mørup, M., Hansen, L. K., Herrmann, C. S., Parnas, J., and Arnfred, S. M. (2006). Parallel factor analysis as an exploratory tool for wavelet transformed event-related eeg. *NeuroImage*, 29(3):938–947.

[Mørup et al., 2011] Mørup, M., Hansen, L. K., and Madsen, K. H. (2011). Modeling latency and shape changes in trial based neuroimaging data. In *ACSCC*, pages 439–443.

[Mu et al., 2014] Mu, C., Huang, B., Wright, J., and Goldfarb, D. (2014). Square deal: Lower bounds and improved relaxations for tensor recovery. In *ICML*, pages 73–81.

[Mu et al., 2011] Mu, Y., Ding, W., Morabito, M., and Tao, D. (2011). Empirical discriminative tensor analysis for crime forecasting. In *International Conference on Knowledge Science, Engineering and Management*, pages 293–304. Springer.

[Muti and Bourennane, 2005] Muti, D. and Bourennane, S. (2005). Multidimensional filtering based on a tensor approach. *Signal Processing*, 85(12):2338–2353.

[Nagy and Kilmer, 2006] Nagy, J. G. and Kilmer, M. E. (2006). Kronecker product approximation for preconditioning in three-dimensional imaging applications. *IEEE Transactions on Image Processing*, 15(3):604–613.

[Nene et al., 1996] Nene, S. A., Nayar, S. K., Murase, H., et al. (1996). Columbia object image library (coil-20).

[Nickel et al., 2011] Nickel, M., Tresp, V., and Kriegel, H.-P. (2011). A three-way model for collective learning on multi-relational data. In *ICML*, volume 11, pages 809–816.

[Nickel et al., 2012] Nickel, M., Tresp, V., and Kriegel, H.-P. (2012). Factorizing yago: scalable machine learning for linked data. In *Proceedings of the 21st international conference on World Wide Web*, pages 271–280. ACM.

[Nion and De Lathauwer, 2008] Nion, D. and De Lathauwer, L. (2008). An enhanced line search scheme for complex-valued tensor decompositions. application in ds-cdma. *Signal Processing*, 88(3):749–755.

[Nion et al., 2010] Nion, D., Mokios, K. N., Sidiropoulos, N. D., and Potamianos, A. (2010). Batch and adaptive parafac-based blind separation of convolutive speech mixtures. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1193–1207.

[Nion and Sidiropoulos, 2009] Nion, D. and Sidiropoulos, N. D. (2009). Adaptive algorithms to track the parafac decomposition of a third-order tensor. *IEEE Transactions on Signal Processing*, 57(6):2299–2310.

[Olshausen and Field, 1997] Olshausen, B. A. and Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325.

[Panagakis and Kotropoulos, 2011] Panagakis, Y. and Kotropoulos, C. (2011). Automatic music tagging via parafac2. In *acoustics, speech and signal processing (ICASSP), 2011 IEEE international conference on*, pages 481–484. IEEE.

[Pantraki and Kotropoulos, 2015] Pantraki, E. and Kotropoulos, C. (2015). Automatic image tagging and recommendation via parafac2. In *Machine Learning for Signal Processing (MLSP), 2015 IEEE 25th International Workshop on*, pages 1–6. IEEE.

[Papadimitriou et al., 2005] Papadimitriou, S., Sun, J., and Faloutsos, C. (2005). Streaming pattern discovery in multiple time-series. In *Proceedings of the 31st international conference on Very large data bases*, pages 697–708. VLDB Endowment.

[Papalexakis et al., 2013] Papalexakis, E. E., Akoglu, L., and Ienco, D. (2013). Do more views of a graph help? community detection and clustering in multi-graphs. In *FUSION*, pages 899–905. Citeseer.

[Papalexakis et al., 2014] Papalexakis, E. E., Faloutsos, C., Mitchell, T. M., Talukdar, P. P., Sidiropoulos, N. D., and Murphy, B. (2014). Turbosmt: Accelerating coupled sparse matrix-tensor factorizations by 200x. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 118–126. SIAM.

[Papalexakis et al., 2012] Papalexakis, E. E., Faloutsos, C., and Sidiropoulos, N. D. (2012). Parcube: Sparse parallelizable tensor decompositions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 521–536. Springer.

[Papalexakis et al., 2017] Papalexakis, E. E., Faloutsos, C., and Sidiropoulos, N. D. (2017). Tensors for data mining and data fusion: Models, applications, and scalable algorithms. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):16.

[Paranjape et al., 2017] Paranjape, A., Benson, A. R., and Leskovec, J. (2017). Motifs in temporal networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 601–610, New York, NY, USA. ACM.

[Perros et al., 2015] Perros, I., Chen, R., Vuduc, R., and Sun, J. (2015). Sparse hierarchical tucker factorization and its application to healthcare. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pages 943–948. IEEE.

[Perros et al., 2017] Perros, I., Papalexakis, E. E., Wang, F., Vuduc, R., Searles, E., Thompson, M., and Sun, J. (2017). Spartan: Scalable parafac2 for large & sparse data. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 375–384. ACM.

[Phan and Cichocki, 2011] Phan, A. H. and Cichocki, A. (2011). Parafac algorithms for large-scale problems. *Neurocomputing*, 74(11):1970–1984.

[Phan et al., 2013a] Phan, A.-H., Tichavskỳ, P., and Cichocki, A. (2013a). Fast alternating ls algorithms for high order candecomp/parafac tensor factorizations. *IEEE Transactions on Signal Processing*, 61(19):4834–4846.

[Phan et al., 2013b] Phan, A.-H., Tichavsky, P., and Cichocki, A. (2013b). Low complexity damped gauss–newton algorithms for candecomp/parafac. *SIAM Journal on Matrix Analysis and Applications*, 34(1):126–147.

[Pilászy et al., 2010] Pilászy, I., Zibriczky, D., and Tikk, D. (2010). Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 71–78. ACM.

[Rai et al., 2014] Rai, P., Wang, Y., Guo, S., Chen, G., Dunson, D. B., and Carin, L. (2014). Scalable bayesian low-rank decomposition of incomplete multiway tensors. In *ICML*, pages 1800–1808.

[Rajih et al., 2008] Rajih, M., Comon, P., and Harshman, R. A. (2008). Enhanced line search: A novel method to accelerate parafac. *SIAM journal on matrix analysis and applications*, 30(3):1128–1147.

[Rendle, 2010] Rendle, S. (2010). Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE.

[Reynolds et al., 2016] Reynolds, M. J., Doostan, A., and Beylkin, G. (2016). Randomized alternating least squares for canonical tensor decompositions: Application to a pde with random data. *SIAM Journal on Scientific Computing*, 38(5):A2634–A2664.

[Samaria and Harter, 1994] Samaria, F. S. and Harter, A. C. (1994). Parameterisation of a stochastic model for human face identification. In *Appli-*

cations of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on, pages 138–142. IEEE.

[Savas and Eldén, 2007] Savas, B. and Eldén, L. (2007). Handwritten digit classification using higher order singular value decomposition. *Pattern recognition*, 40(3):993–1003.

[Schimbinschi et al., 2015] Schimbinschi, F., Nguyen, X. V., Bailey, J., Leckie, C., Vu, H., and Kotagiri, R. (2015). Traffic forecasting in complex urban networks: Leveraging big data and machine learning. In *Big data (big data), 2015 IEEE international conference on*, pages 1019–1024. IEEE.

[Shetty and Adibi, 2004] Shetty, J. and Adibi, J. (2004). The enron email dataset database schema and brief statistical report. *Information sciences institute technical report, University of Southern California*, 4.

[Shi et al., 2015] Shi, L., Gangopadhyay, A., and Janeja, V. P. (2015). Stensr: Spatio-temporal tensor streams for anomaly detection and pattern discovery. *Knowledge and Information Systems*, 43(2):333–353.

[Shin et al., 2017] Shin, K., Hooi, B., Kim, J., and Faloutsos, C. (2017). Densealert: Incremental dense-subtensor detection in tensor streams. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1057–1066. ACM.

[Sidiropoulos and Bro, 2000] Sidiropoulos, N. D. and Bro, R. (2000). On the uniqueness of multilinear decomposition of n-way arrays. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 14(3):229–239.

[Sidiropoulos et al., 2017] Sidiropoulos, N. D., De Lathauwer, L., Fu, X., Huang, K., Papalexakis, E. E., and Faloutsos, C. (2017). Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582.

[Şimşekli et al., 2015] Şimşekli, U., Cemgil, A. T., and Ermiş, B. (2015). Learning mixed divergences in coupled matrix and tensor factorization models. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 2120–2124. IEEE.

[Smilde et al., 2005] Smilde, A., Bro, R., and Geladi, P. (2005). *Multi-way analysis: applications in the chemical sciences*. John Wiley & Sons.

[Smith et al., 2017] Smith, S., Choi, J. W., Li, J., Vuduc, R., Park, J., Liu, X., and Karypis, G. (2017). FROSTT: The formidable repository of open sparse tensors and tools.

[Smith et al., 2018] Smith, S., Huang, K., Sidiropoulos, N. D., and Karypis, G. (2018). Streaming tensor factorization for infinite data sources. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 81–89. SIAM.

[Smith et al., 2015] Smith, S., Ravindran, N., Sidiropoulos, N. D., and Karypis, G. (2015). Splatt: Efficient and parallel sparse tensor-matrix multiplication. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 61–70. IEEE.

[Sobral et al., 2014] Sobral, A., Baker, C. G., Bouwmans, T., and Zahzah, E.-h. (2014). Incremental and multi-feature tensor subspace learning applied for background modeling and subtraction. In *International Conference Image Analysis and Recognition*, pages 94–103. Springer.

[Song et al., 2017] Song, Q., Huang, X., Ge, H., Caverlee, J., and Hu, X. (2017). Multi-aspect streaming tensor completion. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 435–443. ACM.

[Sun et al., 2006] Sun, J., Tao, D., and Faloutsos, C. (2006). Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 374–383. ACM.

[Sun et al., 2008] Sun, J., Tao, D., Papadimitriou, S., Yu, P. S., and Faloutsos, C. (2008). Incremental tensor analysis: Theory and applications. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2(3):11.

[Sun et al., 2005] Sun, J.-T., Zeng, H.-J., Liu, H., Lu, Y., and Chen, Z. (2005). Cubesvd: a novel approach to personalized web search. In *Proceedings of the 14th international conference on World Wide Web*, pages 382–390. ACM.

[Sun et al., 2016] Sun, Y., Yuan, N. J., Wang, Y., Xie, X., McDonald, K., and Zhang, R. (2016). Contextual intent tracking for personal assistants. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 273–282. ACM.

[Tao et al., 2008] Tao, D., Song, M., Li, X., Shen, J., Sun, J., Wu, X., Faloutsos, C., and Maybank, S. J. (2008). Bayesian tensor approach for 3-d face modeling. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(10):1397.

[Tomasi and Bro, 2005] Tomasi, G. and Bro, R. (2005). Parafac and missing values. *Chemometrics and Intelligent Laboratory Systems*, 75(2):163–180.

[Vasilescu and Terzopoulos, 2002a] Vasilescu, M. A. O. and Terzopoulos, D. (2002a). Multilinear analysis of image ensembles: Tensorfaces. In *European Conference on Computer Vision*, pages 447–460. Springer.

[Vasilescu and Terzopoulos, 2002b] Vasilescu, M. A. O. and Terzopoulos, D. (2002b). Multilinear image analysis for facial recognition. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 2, pages 511–514. IEEE.

[Vasilescu and Terzopoulos, 2004] Vasilescu, M. A. O. and Terzopoulos, D. (2004). Tensortextures: Multilinear image-based rendering. *ACM Transactions on Graphics (TOG)*, 23(3):336–342.

[Vervliet et al., 2014] Vervliet, N., Debals, O., Sorber, L., and De Lathauwer, L. (2014). Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis. *IEEE Signal Processing Magazine*, 31(5):71–79.

[Viswanath et al., 2009] Viswanath, B., Mislove, A., Cha, M., and Gummadi, K. P. (2009). On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM workshop on Online social networks*, pages 37–42. ACM.

[Wang and Ahuja, 2004] Wang, H. and Ahuja, N. (2004). Compact representation of multidimensional data using tensor rank-one decomposition. *vectors*, 1:5.

[Wang et al., 2003] Wang, H. et al. (2003). Facial expression decomposition. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 958–965. IEEE.

[Wang and Zhang, 2013] Wang, Y.-X. and Zhang, Y.-J. (2013). Nonnegative matrix factorization: A comprehensive review. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1336–1353.

[Welling and Weber, 2001] Welling, M. and Weber, M. (2001). Positive tensor factorization. *Pattern Recognition Letters*, 22(12):1255–1261.

[Wise et al., 2001] Wise, B. M., Gallagher, N. B., and Martin, E. B. (2001). Application of parafac2 to fault detection and diagnosis in semiconductor etch. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 15(4):285–298.

[Xiong et al., 2010] Xiong, L., Chen, X., Huang, T.-K., Schneider, J., and Carbonell, J. G. (2010). Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 211–222. SIAM.

[Xu et al., 2018] Xu, J., Liu, X., Wilson, T., Tan, P.-N., Hatami, P., and Luo, L. (2018). Muscat: Multi-scale spatio-temporal learning with application to climate modeling. In *IJCAI*, pages 2912–2918.

[Xu et al., 2016] Xu, J., Zhou, J., Tan, P.-N., Liu, X., and Luo, L. (2016). Wisdom: Weighted incremental spatio-temporal multi-task learning via tensor decomposition. In *Big Data (Big Data), 2016 IEEE International Conference on*, pages 522–531. IEEE.

[Xu and Yin, 2013] Xu, Y. and Yin, W. (2013). A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on imaging sciences*, 6(3):1758–1789.

[Yang et al., 2017] Yang, K., Li, X., Liu, H., Mei, J., Xie, G., Zhao, J., Xie, B., and Wang, F. (2017). Tagited: Predictive task guided tensor decomposition for representation learning from electronic health records. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence.*

[Yılmaz et al., 2011] Yılmaz, K. Y., Cemgil, A. T., and Simsekli, U. (2011). Generalised coupled tensor factorisation. In *Advances in neural information processing systems*, pages 2151–2159.

[Yokota et al., 2016] Yokota, T., Zhao, Q., and Cichocki, A. (2016). Smooth parafac decomposition for tensor completion. *IEEE Transactions on Signal Processing*, 64(20):5423–5436.

[Zhang and Sawchuk, 2012] Zhang, M. and Sawchuk, A. A. (2012). Usc-had: a daily activity dataset for ubiquitous activity recognition using wearable sensors. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 1036–1043. ACM.

[Zhao et al., 2015] Zhao, Q., Zhang, L., and Cichocki, A. (2015). Bayesian cp factorization of incomplete tensors with automatic rank determination. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1751–1763.

[Zhao et al., 2016] Zhao, Q., Zhou, G., Zhang, L., Cichocki, A., and Amari, S.-I. (2016). Bayesian robust tensor factorization for incomplete multiway data. *IEEE transactions on neural networks and learning systems*, 27(4):736–748.

[Zhou et al., 2017] Zhou, Z., Fang, J., Yang, L., Li, H., Chen, Z., and Blum, R. S. (2017). Low-rank tensor decomposition-aided channel estimation for millimeter wave mimo-ofdm systems. *IEEE Journal on Selected Areas in Communications*, 35(7):1524–1538.