PLOS ONE

RESEARCH ARTICLE

S^3CMTF: Fast, accurate, and scalable method for incomplete coupled matrix-tensor factorization

Dongjin Choi[1], **Jun-Gi Jang**[2], **U Kang**[2]*

1 School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, Georgia, United States of America, **2** Department of Computer Science and Engineering, Seoul National University, Seoul, Republic of Korea

* ukang@snu.ac.kr

Citation: Choi D, Jang J-G, Kang U (2019) S^3CMTF: Fast, accurate, and scalable method for incomplete coupled matrix-tensor factorization. PLoS ONE 14(6): e0217316. https://doi.org/10.1371/journal.pone.0217316

Editor: Junwen Wang, Mayo Clinic Arizona, UNITED STATES

Received: February 13, 2019

Accepted: May 8, 2019

Published: June 28, 2019

Data Availability Statement: All data files are available from the web page: https://datalab.snu.ac.kr/S3CMTF/.

Funding: This work was supported by the National Research Foundation of Korea (NRF) funded by MSIT (2019R1A2C2004990, and NRF-016M3C4A7952587, PF Class Heterogeneous High Performance Computer Development). The Institute of Engineering Research at Seoul National University provided research facilities for this work. The ICT at Seoul National University provides research facilities for this study. The funders had

Abstract

How can we extract hidden relations from a tensor and a matrix data simultaneously in a fast, accurate, and scalable way? Coupled matrix-tensor factorization (CMTF) is an important tool for this purpose. Designing an accurate and efficient CMTF method has become more crucial as the size and dimension of real-world data are growing explosively. However, existing methods for CMTF suffer from lack of accuracy, slow running time, and limited scalability. In this paper, we propose S^3CMTF, a fast, accurate, and scalable CMTF method. In contrast to previous methods which do not handle large sparse tensors and are not parallelizable, S^3CMTF provides parallel sparse CMTF by carefully deriving gradient update rules. S^3CMTF asynchronously updates partial gradients without expensive locking. We show that our method is guaranteed to converge to a quality solution theoretically and empirically. S^3CMTF further boosts the performance by carefully storing intermediate computation and reusing them. We theoretically and empirically show that S^3CMTF is the fastest, outperforming existing methods. Experimental results show that S^3CMTF is up to 930× faster than existing methods while providing the best accuracy. S^3CMTF shows linear scalability on the number of data entries and the number of cores. In addition, we apply S^3CMTF to Yelp rating tensor data coupled with 3 additional matrices to discover interesting patterns.

Introduction

Given a tensor data, and related matrix data, how can we analyze them efficiently? Tensors (i.e., multi-dimensional arrays) and matrices are natural representations for various real world high-order data [1, 2, 3]. For instance, an online review site Yelp provides rich information about users (name, friends, reviews, etc.), or businesses (name, city, Wi-Fi, etc.). One popular representation of such data includes a 3-way rating tensor with (user ID, business ID, time) triplets and an additional friendship matrix with (user ID, user ID) pairs. Coupled matrix-tensor factorization (CMTF) is an effective tool for joint analysis of coupled matrices and a tensor. The main purpose of CMTF is to integrate matrix factorization [4] and tensor factorization [5]

no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

to efficiently extract the factor matrices of each mode. The extracted factors have many useful applications such as latent semantic analysis [6, 7, 8], recommendation systems [9, 10], network traffic analysis [11], and completion of missing values [12, 13, 14].

However, existing CMTF methods do not provide good performance in terms of time, accuracy, and scalability. CMTF-Tucker-ALS [15], a method based on Tucker decomposition [16], has a limitation that it is only applicable for dense data and not parallelizable. For sparse real-world data, it assumes empty entries as zero and outputs highly skewed results which lead to high reconstruction error. Moreover, CMTF-Tucker-ALS does not scale to large data because it suffers from high memory requirement caused by *M-bottleneck problem* [17]. CMTF-OPT [12] is a CMTF method based on CANDECOMP/PARAFAC (CP) decomposition [18]. SDF [19] provided Quasi-Newton and nonlinear least squares optimization techniques for general coupled factorization problems where factors may have certain structures as Toeplitz, orthogonal and nonnegative. CMTF-Tucker-ALS and CMTF-OPT undergo high reconstruction error since the former is not applicable to sparse data, and the latter focuses only on CP model and thus cannot be generalized to the Tucker model. Furthermore, both methods are sequential and hard to take benefit of multi-core parallelization.

In this paper, we propose S^3CMTF (**S**parse, lock-free **S**GD based, and **S**calable CMTF), a CMTF method which resolves the problems of previous methods. S^3CMTF provides parallel, sparse CMTF based on Tucker factorization unlike previous methods which do not support sparse tensors or cannot be parallelized. We also show that asynchronously parallel stochastic gradient descent (SGD) is useful for S^3CMTF in multi-core shared memory systems without expensive locking. S^3CMTF further boosts the performance by storing intermediate computation and reusing them. Table 1 shows the comparison of S^3CMTF and other existing methods. The main contributions of our study are as follows:

- **Algorithm**: We propose S^3CMTF, a coupled tensor-matrix factorization algorithm for matrix-tensor joint datasets. S^3CMTF is designed to efficiently extract factors from the joint datasets by taking advantage of sparsity, exploiting intermediate data. We propose a method which resolves conflicts of parallelization and leads to a solution with guaranteed convergence.
- **Performance**: S^3CMTF shows the best performance on accuracy, speed, and scalability. S^3CMTF runs up to **930× faster** and is more scalable than existing methods as shown in Fig 1A. For real-world datasets, S^3CMTF converges faster to the better optimum as shown in Fig 1B.
- **Discovery**: Applying S^3CMTF on Yelp review dataset with a 3-mode tensor (user, business, time) coupled with 3 additional matrices ((user, user), (business, category), and (business, city)), we observe interesting patterns and clusters of businesses and suggest a process for personal recommendation.

Table 1. Comparison of our proposed S^3CMTF and the existing CMTF methods. S^3CMTF outperforms all other methods in terms of time, accuracy, scalability, memory usage, and parallelizability.

Method	Time	Accuracy	Scalability	Memory	Parallel
CMTF-Tucker-ALS	slow	low	low	high	no
CMTF-OPT	slow	low	low	high	no
S^3CMTF-base	fast	**high**	**high**	**lower**	**yes**
S^3CMTF-opt	**faster**	**high**	**high**	low	**yes**

https://doi.org/10.1371/journal.pone.0217316.t001

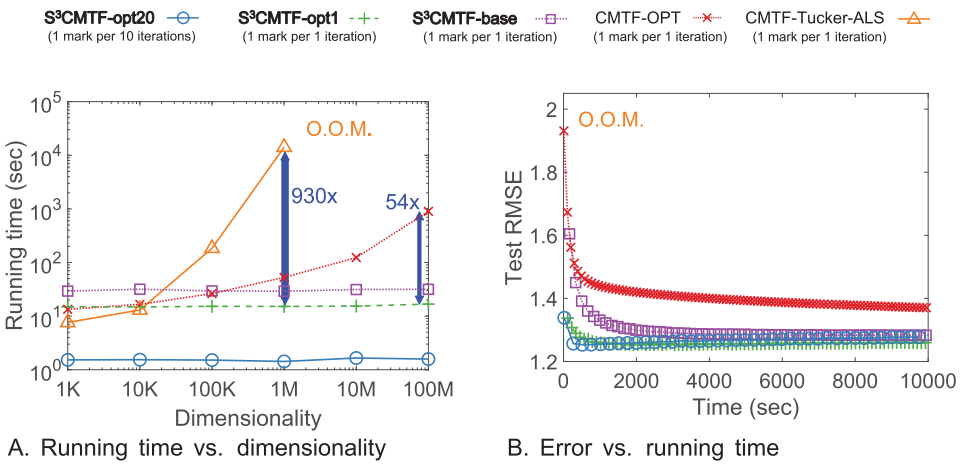

Fig 1. Comparison of our proposed S^3CMTF and the existing methods. (a) For a fixed number of nonzeros, S^3CMTF takes constant time as dimensionality grows, while existing methods become slower. Our sequential method S^3CMTF-opt1 is 930× and 54× faster than CMTF-OPT and CMTF-Tucker ALS, respectively. (b) S^3CMTF-opt20 shows the best convergence rate and accuracy on real world Yelp dataset. CMTF-Tucker-ALS shows O.O.M. in both experiments. (O.O.M.: out of memory error).

https://doi.org/10.1371/journal.pone.0217316.g001

Preliminaries and related works

In this section, we describe preliminaries for tensor and coupled matrix-tensor factorization. We list all symbols used in this paper in Table 2.

Tensor

A tensor is a multi-dimensional array. Each 'dimension' of a tensor is called *mode* or *way*. The length of each mode is called 'dimensionality' and denoted by $I_1, \cdots, I_N$. In this paper, an N-mode or N-way tensor is denoted by the boldface Euler script capital (e.g. $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$), and matrices are denoted by boldface capitals (e.g. $\mathbf{A}$). x_α and a_β denote the entry of $\mathcal{X}$ and $\mathbf{A}$ with indices α and β, respectively.

We describe tensor operations used in this paper. A mode-n fiber is a vector which has fixed indices except for the n-th index in a tensor. The mode-n matrix product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ with a matrix $\mathbf{A} \in \mathbb{R}^{J \times I_n}$ is denoted by $\mathcal{X} \times_n \mathbf{A}$ and has the size of $I_1 \times \cdots I_{n-1} \times J \times I_{n+1} \cdots \times I_N$. It is defined as:

$$(\mathcal{X} \times_n \mathbf{A})_{i_1 \cdots i_{n-1} j i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \cdots i_N} a_{j i_n}, \tag{1}$$

where a_{ji_n} is the (j, i_n)-th entry of $\mathbf{A}$. For brevity, we use the following shorthand notation for multiplication on every mode as in [20]:

$$\mathcal{X} \times \{\mathbf{A}\} \coloneqq \mathcal{X} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)}, \tag{2}$$

where $\{\mathbf{A}\}$ denotes the ordered set $\{\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(N)}\}$.

We use the following notation for multiplication on every mode except n-th mode.

$$\mathcal{X} \times_{-n} \{\mathbf{A}\} \coloneqq \mathcal{X} \times_1 \mathbf{A}^{(1)} \cdots \times_{n-1} \mathbf{A}^{(n-1)} \times_{n+1} \mathbf{A}^{(n+1)} \cdots \times_N \mathbf{A}^{(N)}.$$

We examine the case that an ordered set of row vectors $\{\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \cdots, \mathbf{a}^{(N)}\}$, denoted by $\{\mathbf{a}\}$, is multiplied to a tensor $\mathcal{X}$. First, consider the multiplication for every corresponding mode. By

Table 2. Table of symbols.

Symbol	Definition
$\mathcal{X}$	input tensor
$\mathcal{G}$	core tensor
N	order (number of modes) of the input tensor
I_n	dimensionality of n-th mode of input tensor $\mathcal{X}$
J_n	dimensionality of n-th mode of core tensor $\mathcal{G}$
α	a tensor index $(i_1\, i_2 \cdots i_N)$
x_α	the entry of $\mathcal{X}$ with index α
$\mathbf{X}_{(n)}$	mode-n matricization of a tensor
$\mathbf{U}^{(n)}$	n-th factor matrix of $\mathcal{X}$
$\{\mathbf{U}\}$	set of all factor matrices of $\mathcal{X}$
$\mathbf{u}_i^{(n)}$	the i-th row vector of $\mathbf{U}^{(n)}$
$\{\mathbf{u}\}_\alpha$	ordered set of row vectors $\{\mathbf{u}_{i_1}^{(1)}, \mathbf{u}_{i_2}^{(2)}, \ldots, \mathbf{u}_{i_N}^{(N)}\}$
$\{\mathbf{u}\}_\alpha^\top$	ordered set of column vectors $\{\mathbf{u}_{i_1}^{(1)\top}, \mathbf{u}_{i_2}^{(2)\top}, \ldots, \mathbf{u}_{i_N}^{(N)\top}\}$
$u_{ij}^{(n)}$	entry of $\mathbf{U}^{(n)}$ with index (i, j)
$\mathbf{Y}$	coupled matrix
β	a matrix index $k_1 k_2$
y_β	the entry of $\mathbf{Y}$ with index β
$\mathbf{V}$	factor matrix for the coupled matrix $\mathbf{Y}$
$\mathbf{v}_k$	the k-th row vector of $\mathbf{V}$
$\Omega_\mathcal{X}$	observed index set of $\mathcal{X}$
$\Omega_\mathcal{X}^{n,i}$	subset of $\Omega_\mathcal{X}$ having i as the n-th index

https://doi.org/10.1371/journal.pone.0217316.t002

Eq (1),

$$\mathcal{X} \times \{\mathbf{a}\} = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \cdots i_N} a_{i_1}^{(1)} a_{i_2}^{(2)} \cdots a_{i_N}^{(N)},$$

where $a_k^{(m)}$ denotes the k-th element of $\mathbf{a}^{(m)}$. Then, consider the multiplication for every mode except n-th mode. Such multiplication results in a vector of length I_n. The k-th entry of the vector is

$$[\mathcal{X} \times_{-n} \{\mathbf{a}\}]_k = \sum_{\forall \alpha \in \Omega_\mathcal{X}^{n,k}} x_\alpha a_{i_1}^{(1)} \cdots a_{i_{n-1}}^{(n-1)} a_{i_{n+1}}^{(n+1)} \cdots a_{i_N}^{(N)}, \tag{3}$$

where $\Omega_\mathcal{X}^{n,k}$ denotes the index set of $\mathcal{X}$ having its n-th index as k. $\alpha = (i_1\, i_2 \cdots i_N)$ denotes the index for an entry.

Tucker decomposition

Tucker decomposition is one of the most popular tensor factorization models. Tucker decomposition factorizes an N-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ into a core tensor $\mathcal{G} \in \mathbb{R}^{J_1 \times J_2 \times \ldots \times J_N}$ and factor matrices $\mathbf{U}^{(1)} \in \mathbb{R}^{I_1 \times J_1}, \mathbf{U}^{(2)} \in \mathbb{R}^{I_2 \times J_2}, \ldots, \mathbf{U}^{(N)} \in \mathbb{R}^{I_N \times J_N}$ satisfying

$$\mathcal{X} \approx \tilde{\mathcal{X}} = \mathcal{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \cdots \times_N \mathbf{U}^{(N)} = \mathcal{G} \times \{\mathbf{U}\}.$$

Element-wise formulation of Tucker model is

$$\begin{aligned} x_\alpha \approx \tilde{x}_\alpha &= \sum_{j_1=1}^{J_1}\sum_{j_2=1}^{J_2}\cdots\sum_{j_N=1}^{J_N} g_{j_1 j_2 \cdots j_N} u^{(1)}_{i_1 j_1} u^{(2)}_{i_2 j_2} \cdots u^{(N)}_{i_N j_N} \\ &= \mathcal{G} \times_1 \mathbf{u}^{(1)}_{i_1} \times_2 \mathbf{u}^{(2)}_{i_2} \cdots \times_N \mathbf{u}^{(N)}_{i_N} := \mathcal{G} \times \{\mathbf{u}\}_\alpha, \end{aligned} \tag{4}$$

where α is a tensor index ($i_1 i_2 \cdots i_N$), and $\mathbf{u}^{(n)}_{i_n}$ denotes the i_n-th row of factor matrix $\mathbf{U}^{(n)}$. $\{\mathbf{u}\}_\alpha$ denotes the set of factor rows $\{\mathbf{u}^{(1)}_{i_1}, \mathbf{u}^{(2)}_{i_2}, \ldots, \mathbf{u}^{(N)}_{i_N}\}$. The core tensor $\mathcal{G}$ indicates the relation between the factors in Tucker formulation. When the core tensor size is restricted as $J_1 = J_2 = \cdots = J_N$ and the core tensor structure is hyper-diagonal, it is equivalent to CANDECOMP/PARAFAC (CP) decomposition. Orthogonality constraint can optionally be imposed to the Tucker decomposition by forcing the factor matrices to have orthonormal columns (e.g. $\mathbf{U}^{(n)T}\mathbf{U}^{(n)} = \mathbf{I}$ for $n = 1, \cdots, N$ where $\mathbf{I}$ is an identity matrix).

Coupled matrix-tensor factorization

Coupled matrix-tensor factorization (CMTF) is proposed for joint factorization of a tensor and matrices. CMTF integrates matrix factorization and tensor factorization.

Definition 1. ***(Coupled Matrix-Tensor Factorization)*** *Given an N-mode tensor* $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ *and a matrix* $\mathbf{Y} \in \mathbb{R}^{I_c \times K}$ *where c is the coupled mode,* $\mathcal{X} \approx \tilde{\mathcal{X}} = \mathcal{G} \times \{\mathbf{U}\}$, *and* $\mathbf{Y} \approx \tilde{\mathbf{Y}} = \mathbf{U}^{(c)}\mathbf{V}^\top$ *are the coupled matrix-tensor factorization.* $\mathbf{U}^{(c)} \in \mathbb{R}^{I_c \times J_c}$ *is the c-th mode factor matrix, and* $\mathbf{V} \in \mathbb{R}^{K \times J_c}$ *denotes the factor matrix for the coupled matrix. Finding the factor matrices and core tensor for CMTF is equivalent to solving*

$$\underset{\mathbf{U}^{(1)},\cdots,\mathbf{U}^{(N)},\mathbf{V},\mathcal{G}}{\arg\min} \|\mathcal{X} - \mathcal{G} \times \{\mathbf{U}\}\|^2 + \|\mathbf{Y} - \mathbf{U}^{(c)}\mathbf{V}^\top\|^2, \tag{5}$$

where $\|\bullet\|$ *denotes the Frobenius norm.*

Various methods have been proposed to efficiently solve the CMTF problem. An alternating least squares (ALS) method CMTF-Tucker-ALS [15] was proposed. CMTF-Tucker-ALS is based on Tucker-ALS (HOOI) [21] which is a popular method for fitting the Tucker model. Tucker-ALS suffers from a crucial intermediate memory-bottleneck problem known as *M-bottleneck problem* [17] that arises from materialization of a large dense tensor $\mathcal{X} \times_{-n} \{\mathbf{U}\}^\top$ as intermediate data where $\{\mathbf{U}\}^\top = \{\mathbf{U}^{(1)\top}, \mathbf{U}^{(2)\top}, \ldots, \mathbf{U}^{(N)\top}\}$. Generalized coupled tensor factorization frameworks [22, 23] have been proposed, and they propose multiplicative methods for non-negative factorization. SDF [19] provided Quasi-Newton and nonlinear least squares optimization techniques for general coupled factorization problems where factors may have certain structures as Toeplitz, orthogonal and nonnegative. A Bayesian method [24] has been proposed. It suggests a generative model for tensor factorization and gets parameters with Gibbs sampling method. Most methods for CMTF use CP decomposition model for $\tilde{\mathcal{X}}$ where $J_1 = J_2 = \cdots = J_N$ and the core tensor $\mathcal{G}$ is hyper-diagonal [12, 25, 26, 27, 28, 19]. CMTF-OPT [12] is a representative algorithm for this problem which uses nonlinear conjugate gradient descent method to find factors. HaTen2 [26, 29], and SCouT [25] propose distributed methods for CMTF using CP decomposition model based on the MapReduce framework. Turbo-SMT [27] provides a time-boosting technique for CP-based CMTF methods.

Note that Eq (5) requires all data entries of $\mathcal{X}$ and $\mathbf{Y}$ to be observed. Unobserved values are set to zeros when $\mathcal{X}$ and $\mathbf{Y}$ are sparse, which results in low accuracy. However, most real world data set shows high sparsity. For example, the density of real world tensor we use for

experiments vary from 10^{-7} to 10^{-4}. For this reason above methods show low accuracy for real-world sparse data; what we focus on this paper is solving CMTF for sparse data.

Definition 2. ***(Sparse CMTF)*** *When $\mathcal{X}$ and $\mathbf{Y}$ are sparse, sparse CMTF aims to find factors only considering the observed entries. Let $\mathcal{W}^{(1)}$ indicates the observed entries of $\mathcal{X}$ such that*

$$w_{\alpha}^{(1)} = \begin{cases} 1 & \text{if } x_{\alpha} \text{ is observed} \\ 0 & \text{if } x_{\alpha} \text{ is unobserved} \end{cases}, \text{ for } \forall \alpha \in \Omega_{\mathcal{X}}.$$

Let $\mathbf{W}^{(2)}$ indicates the observed entries of $\mathbf{Y}$ analogously. We modify Eq (5) as

$$\underset{\mathbf{U}^{(1)},\cdots,\mathbf{U}^{(N)},\mathbf{V},\mathcal{G}}{\arg\min} \|\mathcal{W}^{(1)} * (\mathcal{X} - \mathcal{G} \times \{\mathbf{U}\})\|^2 + \|\mathbf{W}^{(2)} * (\mathbf{Y} - \mathbf{U}^{(c)}\mathbf{V}^{\top})\|^2, \quad (6)$$

where $*$ denotes the Hadamard product (element-wise product).

CMTF-Tucker-ALS does not support sparse CMTF since it calculates a singular vector of full and dense matrix. CMTF-OPT provides single machine approach for sparse CMTF for CP model, and CDTF [30] and FlexiFaCT [28] provide distributed methods for sparse CMTF for CP model. Note that all existing methods are based on CP model. Our method is for more general setting, Tucker decomposition, and also easily applied to CP model.

Proposed method

Overview

S^3CMTF provides an algorithm for the joint factorization of Tucker decomposition. The major challenge of parallel Tucker decomposition is to avoid the race condition, and design an efficient algorithm for updating factors.

In this section, we describe S^3CMTF (Sparse, lock-free SGD based, and Scalable CMTF), our proposed method for fast, accurate, and scalable CMTF. Our purpose is to minimize the number of race conditions with probabilistic guarantee by exploiting problem characteristic and minimize calculations by exploiting intermediate data.

We first propose a lock-free parallel method S^3CMTF-base; then, we propose a time-improved version S^3CMTF-opt. Fig 2 shows the overall scheme of S^3CMTF. S^3CMTF-base employs asynchronous parallel SGD for the parallel update with proper workload distribution, and S^3CMTF-opt further improves the speed of S^3CMTF-base by exploiting intermediate data and reusing them.

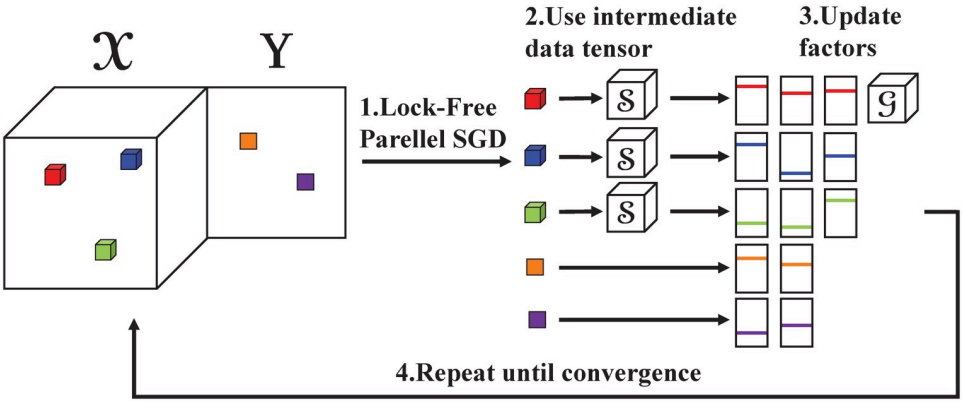

Fig 2. The scheme for S^3CMTF.

https://doi.org/10.1371/journal.pone.0217316.g002

Objective function & gradient

We discuss the improved formulation of the sparse CMTF problem defined in Definition 2. For simplicity, we consider the case that one matrix $\mathbf{Y} \in \mathbb{R}^{I_c \times K}$ is coupled to the c-th mode of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$. Naive calculation of Eq (6) takes excessive time and memory since it includes materialization of dense tensor $\mathcal{G} \times \{\mathrm{U}\}$. Therefore, we re-formulate the new CMTF objective function f to exploit the sparsity of data and add regularization. f is the weighted sum of two functions f_t and f_m which are element-wise sums of squared reconstruction error and regularization terms of tensor $\mathcal{X}$ and matrix $\mathbf{Y}$, respectively.

$$f = \frac{1}{2} f_t + \frac{\lambda_m}{2} f_m, \tag{7}$$

where λ_m is a balancing factor of the two functions.

$$f_t = \left[\sum_{\forall \alpha \in \Omega_{\mathcal{X}}} (x_\alpha - (\mathcal{G} \times \{\mathbf{u}\}_\alpha))^2 \right] + \lambda_{reg} \left(\|\mathcal{G}\|^2 + \sum_{n=1}^{N} \|\mathbf{U}^{(n)}\|^2 \right),$$

where $\alpha = (i_1 \cdots i_N)$, $\Omega_{\mathcal{X}}$ is the observable index set of $\mathcal{X}$, and λ_{reg} denotes the regularization parameter for factors. We rewrite the equation so that it is amenable to SGD update:

$$f_t = \sum_{\forall \alpha \in \Omega_{\mathcal{X}}} \left[(x_\alpha - (\mathcal{G} \times \{\mathbf{u}\}_\alpha))^2 + \frac{\lambda_{reg}}{|\Omega_{\mathcal{X}}|} \|\mathcal{G}\|^2 + \lambda_{reg} \sum_{n=1}^{N} \frac{\|\mathbf{u}_{i_n}^{(n)}\|^2}{|\Omega_{\mathcal{X}}^{n,i_n}|} \right],$$

where $\alpha = (i_1 \cdots i_N)$. Note that $\Omega_{\mathcal{X}}^{n,i_n}$ is the subset of $\Omega_{\mathcal{X}}$ having i_n as the n-th index. Now we formulate f_m, the sum of squared errors of coupled matrix and regularization term corresponding to the coupled matrix.

$$f_m = \sum_{\forall \beta = (j_1 j_2) \in \Omega_{\mathbf{Y}}} \left[(y_\beta - \mathbf{u}_{j_1}^{(c)} \mathbf{v}_{j_2}^\top)^2 + \frac{\lambda_{reg}}{|\Omega_{\mathbf{Y}}^{2,j_2}|} \|\mathbf{v}_{j_2}\|^2 \right].$$

We calculate the gradient of f (Eq (7)) with respect to factors and core for stochastic gradient descent update. Consider that we pick one index $\alpha = (i_1 \ldots i_N) \in \Omega_{\mathcal{X}}$ and matrix index $\beta = (j_1 j_2) \in \Omega_{\mathbf{Y}}$. We calculate the corresponding partial derivatives of f with respect to the factors and the core tensor as follows.

$$\left. \frac{\partial f}{\partial \mathbf{u}_{i_n}^{(n)}} \right|_\alpha = -(x_\alpha - (\mathcal{G} \times \{\mathbf{u}\}_\alpha))[(\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha)_{(n)}]^\top + \frac{\lambda_{reg}}{|\Omega_{\mathcal{X}}^{n,i_n}|} \mathbf{u}_{i_n}^{(n)}, \tag{8a}$$

$$\left. \frac{\partial f}{\partial \mathcal{G}} \right|_\alpha = -(x_\alpha - (\mathcal{G} \times \{\mathbf{u}\}_\alpha)) \times \{\mathbf{u}\}_\alpha^\top + \frac{\lambda_{reg}}{|\Omega_{\mathcal{X}}|} \mathcal{G}, \tag{8b}$$

$$\left. \frac{\partial f}{\partial \mathbf{u}_{j_1}^{(c)}} \right|_\beta = -\lambda_m (y_\beta - \mathbf{u}_{j_1}^{(c)} \mathbf{v}_{j_2}^\top) \mathbf{v}_{j_2}, \tag{8c}$$

$$\left. \frac{\partial f}{\partial \mathbf{v}_{j_2}} \right|_\beta = -\lambda_m (y_\beta - \mathbf{u}_{j_1}^{(c)} \mathbf{v}_{j_2}^\top) \mathbf{u}_{j_1}^{(c)} + \frac{\lambda_m \lambda_{reg}}{|\Omega_{\mathbf{Y}}^{2,j_2}|} \mathbf{v}_{j_2}. \tag{8d}$$

Note that our formulated coupled matrix-tensor factorization model is easily generalized to the case that multiple matrices are coupled to a tensor. We couple multiple matrices to a tensor for experiments in Sections for experiments and discovery.

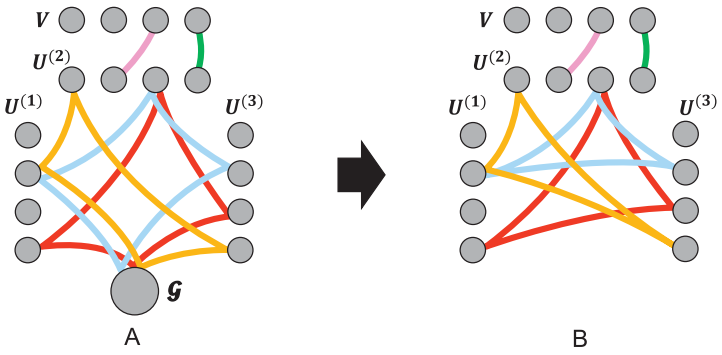

Fig 3. Example hypergraphs induced by S^3CMTF objective function (Eq (7)). A matrix $\mathbf{Y}$ is coupled to the second mode of $\mathcal{X}$ with a coupled factor matrix $\mathbf{V}$. Each node represents a factor row or the core tensor. Each hyperedge includes corresponding factors to an SGD update. (a) Induced hypergraph with the core tensor. Every hyperedge corresponding to tensor entries includes $\mathcal{G}$. (b) Induced hypergraph without core tensor. The graph has sparse structure as every node is shared by only few hyperedges.

https://doi.org/10.1371/journal.pone.0217316.g003

Multi-core parallelization

How can we parallelize the SGD updates for CMTF in multiple cores? In CMTF, SGD is hard to be parallelized without conflicts since each update may suffer from memory conflicts by attempting to write the core tensor $\mathcal{G}$ to memory concurrently [31]. One solution for this problem is memory locking and synchronization. However, there are lots of overhead associated with locking. Therefore, we use lock-free strategy to parallelize S^3CMTF. We develop a parallel update scheme for S^3CMTF by adopting HOGWILD! update scheme [32]. For any SGD problem, a hypergraph is induced where its nodes represent parameters and edges represent the set of parameters related to a data point.

Definition 3. ***(Induced Hypergraph)*** *The objective function in* Eq (7) *induces a hypergraph* $G = (V, E)$ *whose nodes represent factor rows and the core tensor. Each entry of* $\mathcal{X}$ *and* $\mathbf{Y}$ *induces a hyperedge* $e \in E$ *consisting of corresponding factor rows or core tensor.* Fig 3A *shows an example induced graph of* S^3CMTF.

Lock-free parallel updates often converge nearly linearly for a sparse SGD problem in which conflicts between different updates rarely occur [32]. However, in CMTF with Tucker formulation, every update of tensor entries includes the core tensor $\mathcal{G}$ as shown in Fig 3A. We allocate the update of the core tensor $\mathcal{G}$ to one dedicated CPU core and increase the step size by the number to keep the expected step size unchanged, which leads to line 7 of Algorithm 1 described in the next section. Then we obtain a new induced hypergraph in Fig 3B. Previous induced hypergraph (Fig 3A) implies that every factor update (red, blue, and orange hyperedges) is in conflict with each other on updating the core tensor, resulting to unexpected behaviors. In contrast, the new induced hypergraph shows that the update of factors is independent of that of the core tensor.

Note that our problem with this induced hypergraph is a general case of matrix completion problem in [32] which provides convergence guarantee of lock-free parallelism; each edge in our hypergraph entails N vertices, while that in [32] entails only 2 vertices.

Algorithm 1 S^3CMTF-base

Require: Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, rank $(J_1, \cdots, J_N)$, number of parallel cores P, initial learning rate η_0, decay rate μ, coupled mode c, and coupled matrix $\mathbf{Y} \in \mathbb{R}^{I_c \times K}$

Emsure: Core tensor $\mathcal{G} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$, factor matrices $\mathbf{U}^{(1)}, \cdots, \mathbf{U}^{(N)}$, $\mathbf{V}$

1: Initialize $\mathcal{G}$, $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times J_n}$ for $n = 1, \cdots, N$, and $\mathbf{V}$ randomly

2: **repeat**

3: **for** $\forall\alpha = (i_1 \dots i_N) \in \Omega_{\mathcal{X}}$, $\forall\beta = (j_1 j_2) \in \Omega_{\mathbf{Y}}$ in random order **do in parallel**
4: **if** α is picked **then**
5: $(\frac{\partial f}{\partial \mathbf{u}^{(1)}_{i_1}}, \cdots, \frac{\partial f}{\partial \mathbf{u}^{(N)}_{i_N}}, \frac{\partial f}{\partial \mathcal{G}}) \leftarrow$ *compute_gradient*$(\alpha, x_\alpha, \mathcal{G})$
6: $\mathbf{u}^{(n)}_{i_n} \leftarrow \mathbf{u}^{(n)}_{i_n} - \eta_t \frac{\partial f}{\partial \mathbf{u}^{(n)}_{i_n}}$, (for $n = 1, \cdots, N$)
7: $\mathcal{G} \leftarrow \mathcal{G} - \eta_t P \frac{\partial f}{\partial \mathcal{G}}$ (executed by one dedicated CPU core)
8: **end if**
9: **if** β is picked **then**
10: $\tilde{y}_\beta \leftarrow \mathbf{u}^c_{j_1} \mathbf{v}^\top_{j_2}$, $\frac{\partial f}{\partial \mathbf{u}^{(c)}_{j_1}} \leftarrow -\lambda_m (y_\beta - \tilde{y}_\beta)\mathbf{v}_{j_2}$
11: $\frac{\partial f}{\partial \mathbf{v}_{j_2}} \leftarrow -\lambda_m (y_\beta - \tilde{y}_\beta)\mathbf{u}^{(c)}_{j_1} + \frac{\lambda_m \lambda_{reg}}{|\Omega_{\mathbf{Y}_{2,j_2}}|}\mathbf{v}_{j_2}$
12: $\mathbf{u}^{(c)}_{j_1} \leftarrow \mathbf{u}^{(c)}_{j_1} - \eta_t \frac{\partial f}{\partial \mathbf{u}^{(c)}_{j_1}}$, $\mathbf{v}_{j_2} \leftarrow \mathbf{v}_{j_2} - \eta_t \frac{\partial f}{\partial \mathbf{v}_{j_2}}$
13: **end if**
14: **end for**
15: $\eta_t = \eta_0 (1 + \mu t)^{-1}$
16: **until** convergence conditions are satisfied
17: **for** $n = 1, \dots, N$ **do**
18: $\mathbf{Q}^{(n)}, \mathbf{R}^{(n)} \leftarrow$ QR decomposition of $\mathbf{U}^{(n)}$
19: $\mathbf{U}^{(n)} \leftarrow \mathbf{Q}^{(n)}$, $\mathcal{G} \leftarrow \mathcal{G} \times_n \mathbf{R}^{(n)}$
20: **end for**
21: $\mathbf{V} \leftarrow \mathbf{V}\mathbf{R}^{(c)\top}$
22: **return** $\mathcal{G}$, $\mathbf{U}^{(1)}, \cdots, \mathbf{U}^{(N)}$, $\mathbf{V}$

S^3CMTF-base

We present our method, S^3CMTF-base, combination of the aforementioned techniques. S^3CMTF-base solves the sparse CMTF problem by parallel SGD techniques explained above. Algorithm 1 shows the procedure of S^3CMTF-base. In the beginning, S^3CMTF-base initializes factor matrices and the core tensor randomly (line 1 of Algorithm 1). The outer loop (lines 2-16) repeats until the factor variables converge. The inner loop (lines 3-14) is performed by several cores in parallel. In each inner loop, S^3CMTF-base selects an index which belongs to $\Omega_{\mathcal{X}}$ or $\Omega_{\mathbf{Y}}$ in random order (line 3). If a tensor index α is picked, then the algorithm calculates the partial gradients of corresponding factor rows using *compute_gradient* (Algorithm 2) in line 5, and updates factor row vectors (line 6). Core tensor $\mathcal{G}$ is updated by one dedicated CPU core (line 7). Note that if line 7 is run by multiple cores, a core may interrupt another core's update of $\mathcal{G}$ by overwriting the gradient $\frac{\partial f}{\partial \mathcal{G}}$, which leads to unexpected update of $\mathcal{G}$ and hinders convergence; thus, we eliminate the possibility of such conflict by allocating update of $\mathcal{G}$ to the dedicated CPU core. The update of line 7 is done independently by the dedicated CPU core, but concurrently with gradient calculation (line 5) and factor updates (line 6) of other CPU cores. The number P of cores is multiplied to the gradient to compensate for the one-core update so that SGD uses the same expected learning rate for all the parameters. If a coupled matrix index β is picked, then the gradient update is performed on corresponding factor row vectors (lines 9-13). At the end of the outer loop, the learning rate η_t of the t-th iteration is monotonically decreased [33]. (line 15). QR decomposition is applied on factors to satisfy orthogonality constraint of factor matrices (lines 17-20). QR decomposition of $\mathbf{U}^{(n)}$ generates $\mathbf{Q}^{(n)}$, an orthogonal matrix of the same size as $\mathbf{U}^{(n)}$, and a square matrix $\mathbf{R}^{(n)} \in \mathbb{R}^{J_n \times J_n}$. Substituting $\mathbf{U}^{(n)}$ by $\mathbf{Q}^{(n)}$ (line 19) and $\mathcal{G}$ by $\mathcal{G} \times_1 \mathbf{R}^{(1)} \dots \times_N \mathbf{R}^{(N)}$ (after N-th execution of line 19) result in orthogonal factors with equivalent factorization quality [5]. In the same manner, we substitute $\mathbf{V}$ by $\mathbf{V}\mathbf{R}^{(c)\top}$ (line 21) since $\tilde{\mathbf{Y}} = \mathbf{U}^{(c)}\mathbf{V}^\top = \mathbf{Q}^{(c)}\mathbf{R}^{(c)}\mathbf{V}^\top = \mathbf{Q}^{(c)}(\mathbf{V}\mathbf{R}^{(c)\top})^\top$.

Algorithm 2 *compute_gradient*(α,x_α,$\mathcal{G}$)

Require: Tensor entry x_α, $\alpha = (i_1 \cdots i_N) \in \Omega_\mathcal{X}$, core tensor $\mathcal{G}$

Ensure: Gradients $\frac{\partial f}{\partial \mathbf{u}^{(1)}_{i_1}}, \frac{\partial f}{\partial \mathbf{u}^{(2)}_{i_2}}, \cdots, \frac{\partial f}{\partial \mathbf{u}^{(N)}_{i_N}}, \frac{\partial f}{\partial \mathcal{G}}$

1: $\tilde{x}_\alpha \leftarrow \mathcal{G} \times \{\mathbf{u}\}_\alpha$

2: **for** $n = 1, \cdots, N$ **do**

3: $\quad \frac{\partial f}{\partial \mathbf{u}^{(n)}_{i_n}} \leftarrow -(x_\alpha - \tilde{x}_\alpha)[(\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha)_{(n)}]^\top + \frac{\lambda_{reg}}{|\Omega^{n,i_n}_\mathcal{X}|}\mathbf{u}^{(n)}_{i_n}$

4: **end for**

5: $\frac{\partial f}{\partial \mathcal{G}} \leftarrow -(x_\alpha - \tilde{x}_\alpha) \times \{\mathbf{u}\}^\top_\alpha + \frac{\lambda_{reg}}{|\Omega_\mathcal{X}|}\mathcal{G}$

6: **return** $\frac{\partial f}{\partial \mathbf{u}^{(1)}_{i_1}}, \frac{\partial f}{\partial \mathbf{u}^{(2)}_{i_2}}, \cdots, \frac{\partial f}{\partial \mathbf{u}^{(N)}_{i_N}}, \frac{\partial f}{\partial \mathcal{G}}$

Algorithm 3 *compute_gradient_opt*(α,x_α,$\mathcal{G}$)

Require: Tensor entry x_α, $\alpha = (i_1 \cdots i_N) \in \Omega_\mathcal{X}$, core tensor $\mathcal{G}$

Ensure: Gradients $\frac{\partial f}{\partial \mathbf{u}^{(1)}_{i_1}}, \frac{\partial f}{\partial \mathbf{u}^{(2)}_{i_2}}, \cdots, \frac{\partial f}{\partial \mathbf{u}^{(N)}_{i_N}}, \frac{\partial f}{\partial \mathcal{G}}$

1: $\tilde{x}_\alpha \leftarrow 0$

2: **for** $\forall (j_1 j_2 \ldots j_N) \in \Omega_\mathcal{G}$ **do**

3: $\quad s_{j_1 j_2 \ldots j_N} \leftarrow g_{j_1 j_2 \ldots j_N} u^{(1)}_{i_1 j_1} u^{(2)}_{i_2 j_2} \ldots u^{(N)}_{i_N j_N}$

4: $\quad \tilde{x}_\alpha \leftarrow \tilde{x}_\alpha + s_{j_1 j_2 \ldots j_N}$

5: **end for**

6: **for** $n = 1, \ldots, N$ **do**

7: $\quad \frac{\partial f}{\partial \mathbf{u}^{(n)}_{i_n}} \leftarrow -(x_\alpha - \tilde{x}_\alpha) \cdot \mathit{Collapse}(\mathcal{S}, n) \oslash \mathbf{u}^{(n)}_{i_n} + \frac{\lambda_{reg}}{|\Omega^{n,i_n}_\mathcal{X}|}\mathbf{u}^{(n)}_{i_n}$

8: **end for**

9: $\frac{\partial f}{\partial \mathcal{G}} \leftarrow -(x_\alpha - \tilde{x}_\alpha) \cdot \mathcal{S} \oslash \mathcal{G} + \lambda_{reg}\mathcal{G}$

10: **return** $\frac{\partial f}{\partial \mathbf{u}^{(1)}_{i_1}}, \frac{\partial f}{\partial \mathbf{u}^{(2)}_{i_2}}, \ldots, \frac{\partial f}{\partial \mathbf{u}^{(N)}_{i_N}}, \frac{\partial f}{\partial \mathcal{G}}$

S^{3}CMTF-opt

There is much room for improvement in calculations of S^3CMTF-base. The computational bottleneck of S^3CMTF-base is *compute_gradient*. There are implicitly redundant calculations during multiple tensor-matrix products. For example, calculation of $\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha$ is repeated N times for every execution of *compute_gradient* (Algorithm 2) in line 5 of Algorithm 1. The calculation of $\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha$ for the n-th mode is equivalent to a special case of a well-studied operation, matricized tensor times Khatri-Rao product (MTTKRP). MTTKRP is an operation to compute $\mathbf{X}_{(n)} \odot_{\forall k \neq n} \mathbf{A}^{(k)}$ where $\mathbf{X}_{(n)}$ is a matricized tensor along the n-th mode, and $\odot$ denotes the Khatri-Rao product [34]. $\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha$ is equivalent to an MTTKRP $\mathbf{G}_{(n)} \odot_{\forall k \neq n} \mathbf{u}^{(k)}$ where the matrix $\mathbf{A}^{(k)}$ is replaced by the vector $\mathbf{u}^{(k)}$.

Calculating MTTKRP along all modes is known as the CP gradient problem. In *compute_gradient*, we need to calculate $\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha$ for all N modes (line 3 of Algorithm 2), raising the special case of the CP gradient problem. To solve the particular CP gradient problem faster, we propose a method to avoid redundant computations by reusing the intermediate calculations in previous steps. Calculation of $\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha$ is equivalent to a summation of $g_{j_1 j_2 \ldots j_N} u^{(1)}_{i_1 j_1} \ldots u^{(n-1)}_{i_{n-1} j_{n-1}} u^{(n+1)}_{i_{n+1} j_{n+1}} \ldots u^{(N)}_{i_N j_N}$ (Eq 3) which is a product of the core value $g_{j_1 j_2 \ldots j_N}$ and $N-1$ related factor values. Before the calculation of the CP gradient, $\tilde{x}_\alpha = \mathcal{G} \times \{\mathbf{u}\}_\alpha$ is calculated in line 1 of Algorithm 2. We exploit the fact that $\mathcal{G} \times \{\mathbf{u}\}_\alpha$ is the summation of the product $\sum_{j_1=1}^{J_1} \ldots \sum_{j_N=1}^{J_N} g_{j_1 \ldots j_N} u^{(1)}_{i_1 j_1} \ldots u^{(N)}_{i_N j_N}$ (Eq 4), the product of a core value and all N related factor values. In S^3CMTF-opt, we save time by storing the intermediate calculations for $\tilde{x}_\alpha$ and reusing them.

Definition 4. ***(Intermediate Data)*** *When updating the factor rows for a tensor entry* $x_{\alpha=(i_1\cdots i_N)}$, *we define* $(j_1 j_2 \cdots j_N)$*-th element of intermediate data* $\mathcal{S}$:

$$s_{j_1 j_2 \cdots j_N} \leftarrow g_{j_1 j_2 \cdots j_N} u^{(1)}_{i_1 j_1} u^{(2)}_{i_2 j_2} \cdots u^{(N)}_{i_N j_N}.$$

There is no extra time required for calculating $\mathcal{S}$ because $\mathcal{S}$ is generated while calculating $\tilde{x}_\alpha$. Lemma 1 shows that $\tilde{x}_\alpha$ is calculated by summing all entries of $\mathcal{S}$.

Lemma 1. *For a given tensor index* α, *the estimated tensor entry* $\tilde{x}_\alpha = \sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} \cdots \sum_{j_N=1}^{J_N} s_{j_1 j_2 \cdots j_N}$.

Proof. The proof is straightforward by Eq (4).

We use $\mathcal{S}$ with following *Collapse* operation to calculate gradients efficiently.

Definition 5. ***(Collapse)*** *The Collapse operation of the intermediate tensor* $\mathcal{S}$ *on the n-th mode outputs a row vector defined as*:

$$Collapse(\mathcal{S}, n) = [\textstyle\sum_{\forall \delta \in \Omega_{\mathcal{S}}^{n,1}} s_\delta, \sum_{\forall \delta \in \Omega_{\mathcal{S}}^{n,2}} s_\delta, \cdots, \sum_{\forall \delta \in \Omega_{\mathcal{S}}^{n,J_n}} s_\delta].$$

Collapse operation aggregates the elements of intermediate tensor $\mathcal{S}$ with respect to a fixed mode. We re-express the calculation of gradients for tensor factors in Eqs (8a)–(8d) in an efficient manner.

Lemma 2. ***(Efficient Gradient Calculation)*** *The following statements are equivalent calculations of the gradients as in* Eqs (8a)–(8d).

$$\tilde{x}_\alpha \leftarrow \sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} \cdots \sum_{j_N=1}^{J_N} s_{j_1 j_2 \cdots j_N}, \tag{9a}$$

$$\frac{\partial f}{\partial \mathbf{u}^{(n)}_{i_n}} \leftarrow -(x_\alpha - \tilde{x}_\alpha) \cdot Collapse(\mathcal{S}, n) \oslash \mathbf{u}^{(n)}_{i_n} + \frac{\lambda_{reg}}{|\Omega_{\mathcal{X}}^{n,i_n}|} \mathbf{u}^{(n)}_{i_n}, \tag{9b}$$

$$\frac{\partial f}{\partial \mathcal{G}} \leftarrow -(x_\alpha - \tilde{x}_\alpha) \cdot \mathcal{S} \oslash \mathcal{G} + \lambda_{reg} \mathcal{G}. \tag{9c}$$

where $\alpha = (i_1\, i_2 \cdots i_N)$, *and* $\oslash$ *denotes element-wise division*.

Proof. In Lemma 1, Eq (9a) is proved. To prove the equivalence of Eq (9b) and the Eq (8a), it suffices to show $[(\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha)_{(n)}]^\top = Collapse(\mathcal{S}, n) \oslash \mathbf{u}^{(n)}_{i_n}$ We use Eq (3) for the proof. $\alpha = (i_1 \ldots i_N) \in \Omega_{\mathcal{X}}$ and $\delta = (j_1 \ldots j_N) \in \Omega_{\mathcal{G}}^{n,k}$.

$$\begin{aligned}
[(\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha)_{(n)}]^\top_k &= \sum_{\forall \delta \in \Omega_{\mathcal{G}}^{n,k}} g_\delta u^{(1)}_{i_1 j_1} \cdots u^{(n-1)}_{i_{n-1} j_{n-1}} u^{(n+1)}_{i_{n+1} j_{n+1}} \cdots u^{(N)}_{i_N j_N} \\
&= \sum_{\forall \delta \in \Omega_{\mathcal{G}}^{n,k}} g_\delta u^{(1)}_{i_1 j_1} \cdots u^{(n-1)}_{i_{n-1} j_{n-1}} u^{(n)}_{i_n k} u^{(n+1)}_{i_{n+1} j_{n+1}} \cdots u^{(N)}_{i_N j_N} / u^{(n)}_{i_n k} \\
&= \sum_{\forall \delta \in \Omega_{\mathcal{G}}^{n,k}} s_\delta / u^{(n)}_{i_n k} \\
&= \frac{[Collapse(\mathcal{S}, n)]_k}{u^{(n)}_{i_n k}} \\
&= [Collapse(\mathcal{S}, n) \oslash \mathbf{u}^{(n)}_{i_n}]_k.
\end{aligned}$$

Next, to show the equivalence of Eq (9c) and the second equation of Eq (8), it suffices to show $1 \times \{\mathbf{u}\}_\alpha^\top = \mathcal{S} \oslash \mathcal{G}$.

$$\begin{aligned}[1 \times \{\mathbf{u}\}_\alpha^\top]_{\gamma=(l_1 l_2 \cdots l_N)} &= u^{(1)}_{i_1 l_1} u^{(2)}_{i_2 l_2} \cdots u^{(N)}_{i_N l_N} \\ &= g_\gamma u^{(1)}_{i_1 l_1} \cdots u^{(N)}_{i_N l_N} / g_\gamma \\ &= s_\gamma / g_\gamma \\ &= [\mathcal{S} \oslash \mathcal{G}]_\gamma.\end{aligned}$$

S^3CMTF-opt replaces *compute_gradient* (Algorithm 2) of S^3CMTF-base with *compute_gradient_opt* (Algorithm 3), the time-optimized alternative. We prove that the new calculation scheme is faster than the previous one.

Lemma 3. *compute_gradient_opt is faster than compute_gradient. The theoretical time complexity of compute_gradient is* $\mathcal{O}(N^2 J^N)$ *and the time complexity of compute_gradient_opt is* $\mathcal{O}(NJ^N)$ *where* $J_1 = J_2 = \cdots = J_N = J$.

Proof. We assume that $I_1 = I_2 = \cdots = I_N = I$ for brevity. First, we calculate the time complexity of *compute_gradient* (Algorithm 2). Given a tensor index α, computing $\tilde{x}_\alpha$ (line 1 of Algorithm 2) takes $\mathcal{O}(NJ^N)$. Computing $(\mathcal{G} \times_{-n} \{\mathbf{u}\}_\alpha)$ (line 3) takes $\mathcal{O}(NJ^N)$. Thus, aggregate time for calculating the row gradient for all modes (lines 2-4) takes $\mathcal{O}(N^2 J^N)$. Calculating $(x_\alpha - \tilde{x}_\alpha) \times \{\mathbf{u}\}_\alpha^\top$ (line 5) takes $\mathcal{O}(NJ^N)$. In total, *compute_gradient* takes $\mathcal{O}(N^2 J^N)$ time. Next, we calculate the time complexity of *compute_gradient_opt* (Algorithm 3). Computing an entry of intermediate data $\mathcal{S}$ (line 3 of Algorithm 3) takes $\mathcal{O}(N)$. Aggregate time for getting $\mathcal{S}$ (lines 2-5) is $\mathcal{O}(NJ^N)$ since $|\Omega_\mathcal{G}| = \mathcal{O}(J^N)$. Calculating row gradient for all modes (lines 6-8) takes $\mathcal{O}(NJ^N)$ since *Collapse* operation takes $\mathcal{O}(J^N)$. Calculating gradient for core tensor (line 9) takes $\mathcal{O}(J^N)$. In total, *compute_gradient_opt* takes $\mathcal{O}(NJ^N)$ time.

Analysis

We analyze the proposed method in terms of time complexity per iteration. For simplicity, we assume that $I_1 = I_2 = \cdots = I_N = I$, and $J_1 = J_2 = \cdots = J_N = J$. Table 3 summarizes the time complexity (per iteration) and memory usage of S^3CMTF and other methods. Note that the memory usage refers to the auxiliary space for temporary variables used by a method.

Lemma 4. *The time complexity (per iteration) of* S^3CMTF-*base is* $\mathcal{O}(|\Omega|N^2 J^N/P + |\Omega_\mathbf{Y}|J/P)$ *and the time complexity (per iteration) of* S^3CMTF-*opt is* $\mathcal{O}(|\Omega|NJ^N/P + |\Omega_\mathbf{Y}|J/P)$ *where P denotes the number of parallel cores*.

Proof. First, we check the time complexity of S^3CMTF-base. When a tensor index α is picked in the inner loop (line 4 of Algorithm 1), calculating gradients with respect to tensor factors (line 5) takes $\mathcal{O}(N^2 J^N)$ as shown in Lemma 3. Updating factor rows (line 6) takes

Table 3. Comparison of time complexity (per iteration) and memory usage of our proposed S^3CMTF and other CMTF algorithms. S^3CMTF-opt shows the lowest time complexity and S^3CMTF-base shows the lowest memory usage. For simplicity, we assume that all modes are of size I, of rank J, and an $I \times K$ matrix is coupled to one mode. P is the number of parallel cores. (* indicates the lowest time or memory).

	Time complexity (per iter.)	Memory usage
S^3CMTF-base	$\mathcal{O}(\|\Omega_\mathbf{X}\|N^2 J^N/P + \|\Omega_\mathbf{Y}\|J/P)$	$\mathcal{O}(PJ)^*$
S^3CMTF-opt	$\mathcal{O}(\|\Omega_\mathbf{X}\|NJ^N/P + \|\Omega_\mathbf{Y}\|J/P)^*$	$\mathcal{O}(PJ^N)$
CMTF-Tucker-ALS	$\mathcal{O}(NI^{N-1}J^2 + NI^2 J^{N-1} + I^2 K)$	$\mathcal{O}(IJ^{N-1})$
CMTF-OPT	$\mathcal{O}(\|\Omega_\mathbf{X}\|NJ + NI^{N-1}J + IJK)$	$\mathcal{O}(I^{N-1}J + JK)$

https://doi.org/10.1371/journal.pone.0217316.t003

$\mathcal{O}(NJ)$, and updating core tensor (line 7) takes $\mathcal{O}(J^N)$. If a coupled matrix index β is picked (line 9), calculating $\tilde{y}_\beta$ (line 10) takes $\mathcal{O}(J)$. Calculating and updating the factor rows corresponding to coupled matrix entry (lines 10-12) take $\mathcal{O}(J)$. All calculations except updating core tensor (line 7) are conducted in parallel. Finally, for all $\alpha \in \Omega_\mathcal{X}$ and $\beta \in \Omega_\mathbf{Y}$, S^3CMTF-base takes $\mathcal{O}(|\Omega_\mathcal{X}|N^2J^N/P + |\Omega_\mathbf{Y}|J/P)$ for one iteration. S^3CMTF-opt uses *compute_gradient_opt* instead of *compute_gradient* in line 5 of Algorithm 1, whose time complexity is shown in Lemma 3. Overall running time per iteration for S^3CMTF-opt is $\mathcal{O}(|\Omega_\mathcal{X}|NJ^N/P + |\Omega_\mathbf{Y}|J/P)$.

Experiments

In this and the next sections, we experimentally evaluate S^3CMTF. Especially, we answer the following questions.

Q1: **Performance** How accurate and fast is S^3CMTF compared to competitors?

Q2: **Scalability** How do S^3CMTF and other methods scale in terms of dimensionality, the number of observed entries, and the number of cores?

Q3: **Discovery** What are the discoveries of applying S^3CMTF on real-world data?

The source codes of our method and datasets used in this paper are available at https://datalab.snu.ac.kr/S3CMTF.

Experimental settings

Data. Table 4 shows the data we used in our experiments. We use three real-world datasets, MovieLens (http://grouplens.org/datasets/movielens/10m), Netflix (http://www.netflixprize.com), and Yelp (http://www.yelp.com/dataset_challenge), as well as synthetic data to evaluate S^3CMTF. Each entry of the real-world datasets represents a rating, which consists of (user, 'item', time; rating) where 'item' indicates 'movie' for MovieLens and Netflix, and 'business' for Yelp. We use (movie, genre) and (movie, year) as coupled matrices for MovieLens and Netflix, respectively. We use (user, user) friendship matrix, (business, category) and (business, city) matrices for Yelp. Particularly for scalability experiments, we generate 3-mode synthetic random tensors with dimensionality I and corresponding coupled matrices to observe speed property while size is varying. We vary I in the range of 1K $\sim$ 100M and the number of tensor entries in the range of 1K $\sim$ 100M. We set the number of entries as $|\Omega_\mathbf{Y}| = \frac{1}{10}|\Omega_\mathcal{X}|$ for synthetic coupled matrices. We generated observed indices randomly, and their entries to follow uniform random distribution between 0 and 1.

Table 4. Summary of the data used for experiments. 'K' means thousand, and 'M' million. Tensors and matrices of density 1 are fully observed.

Name	Data	Dimensionality	# entries	Density
MovieLens	User-Movie-Time	71K-11K-157	10M	$\sim 10^{-4}$
	Movie-Genre	20	214K	1
Netflix	User-Movie-Time	480K-18K-74	100M	$\sim 10^{-4}$
	Movie-Yearmonth	110	2M	1
Yelp	User-Business-Time	1M-144K-149	4M	$\sim 10^{-7}$
	User-User	1M	7M	$\sim 10^{-4}$
	Business-Category	1K	172M	1
	Business-City	1K	126M	1
Synthetic	3-mode tensor	1K $\sim$ 100M	1K $\sim$ 100M	10^{-20} *to* $^{-3}$
	Matrix	1K $\sim$ 100M	1K $\sim$ 100M	10^{-11} *to* $^{-4}$

https://doi.org/10.1371/journal.pone.0217316.t004

Measure. We use test RMSE as the measure for tensor reconstruction error.

$$\text{test RMSE} = \sqrt{\frac{1}{|\Omega_{test}|}\sum_{\forall \alpha \in \Omega_{test}} (x_\alpha - \tilde{x}_\alpha)^2}$$

where Ω_{test} is the index set of the test data tensor, x_α stands for each test tensor entry, and $\tilde{x}_\alpha$ is the corresponding reconstructed value.

Methods. For fair comparison, we compare single core run of S^3CMTF-base and S^3CMTF-opt with other single machine CMTF methods: CMTF-Tucker-ALS and CMTF-OPT (described in Section). To examine multi-core performance, we run two versions of S^3CMTF-opt: S^3CMTF-opt1 (1 core), and S^3CMTF-opt20 (20 cores). We exclude distributed CMTF methods [25, 26, 28] since they are designed for Hadoop with multiple machines, and thus take too much time for single machine environment. For example, [17] reported that HaTen2 [26] takes 10,700s to decompose 4-way tensor with $I = 10K$ and $|\Omega_{\mathcal{X}}| = 100K$, which is almost 7,000× slower than our single machine implementation of S^3CMTF-opt. For CMTF-Tucker-ALS, we implemented a C++ version based on Tucker-MET [20], and for CMTF-OPT, we implemented a C++ version of CMTF-OPT [12]. Our implementation for CMTF-OPT solves Eq (6) by sparse matrix operations. We implement S^3CMTF with C++. For all of our C++ implementations, we used C++11 with O2 flag. We used Armadillo 7.700 with LAPACK 3.7.0 and BLAS 3.7.0 for matrix operations such as eigenvector calculations. We used OpenMP 4.0 library for multi-core parallelization of S^3CMTF.

We conduct all experiments on a machine equipped with Intel Xeon E5-2630 v4 2.2GHz CPU and 256GB RAM. We mark out-of-memory (O.O.M.) error when the memory usage exceeds the limit.

Hyperparameters. We set pre-defined hyperparameters that resulted in the best reconstruction error on a 10% validation set by random grid search: tensor rank J, regularization factor λ_{reg}, λ_m, the initial learning rate η_0, and decay rate μ. We set λ_{reg} to 0.1, $\lambda_m = 10$, and $\mu = 0.1$ for all datasets. For rank and initial learning rate, MovieLens: $J = 12$, $\eta = 0.001$, Netflix: $J = 11$, $\eta = 0.001$, and Yelp: $J = 10$, $\eta = 0.0005$. For synthetic datasets, we use $J = 10$ for all experiments.

Performance of S^3CMTF

We observe the performance of S^3CMTF to answer Q1. As seen in Figs 1B and 4, S^3CMTF converges faster to the optimum with the lowest test error than existing methods with the following details.

Accuracy. We divide each data tensor into 80%/20% for train/test sets. Specifically, 80% of the tensor entries are regarded as the train set and remaining 20% as the test set. The lower error for a same elapsed time implies the better accuracy and faster convergence. Figs 1B and 4 show the changes of test RMSE of each method on three datasets over elapsed time which are the answers for Q1. S^3CMTF achieves the lowest error compared to others for the same elapsed time. For Yelp, CMTF-Tucker-ALS yielded an O.O.M. error. S^3CMTF-opt20 achieves the lowest error 1.253, 0.9147, and 0.8037 while the best competing method, CMFT-OPT, gives the error 1.370, 1.018, and 0.8125 for Yelp, Netflix, and MovieLens datasets, respectively. Note that the competing method CMFT-Tucker-ALS gives either an out of memory error or results in the highest error rate.

Running time. We compare our method with the multi-core version of SALS-single [30], a parallel CP decomposition algorithm, to demonstrate the high performance of S^3CMTF compared to the state-of-the-art decomposition algorithms. We used non-coupled CP version of our method, S^3CMTF-CP-opt, by setting $\mathcal{G}$ to be hyper-diagonal and not coupling any

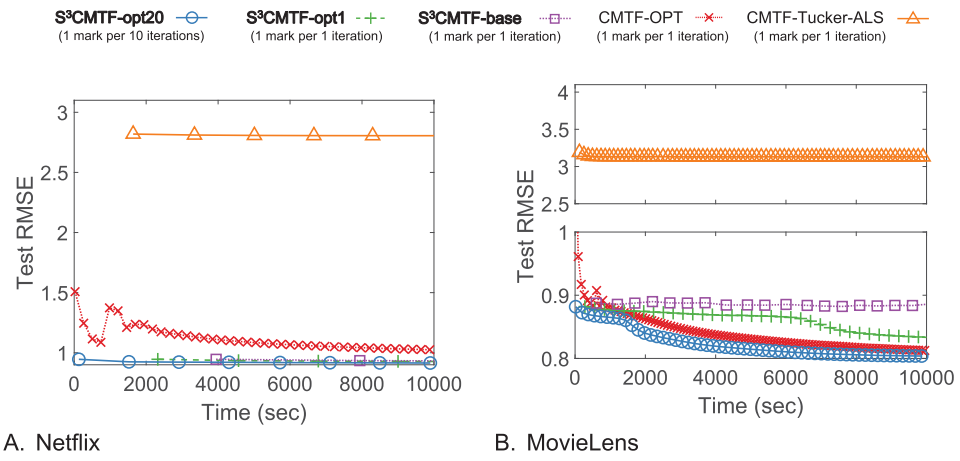

Fig 4. Test RMSE of S^3CMTF and other CMTF methods over iterations. S^3CMTF-opt20 shows the best convergence rate and accuracy.

https://doi.org/10.1371/journal.pone.0217316.g004

matrices. Fig 5 shows that S^3CMTF is better than SALS-single in terms of both error and time for MovieLens dataset. S^3CMTF-TUCKER explicitly denotes S^3CMTF-opt for Tucker model.

Scalability analysis

We present scalability of our proposed S^3CMTF and competitors to answer Q2, in terms of two aspects: data scalability and parallel scalability. We use synthetic data of varying size for evaluation. As a result, we show the running time (for one iteration) of S^3CMTF follows our theoretical analysis in Section.

Data scalability. The time complexity of CMTF-Tucker-ALS and CMTF-OPT have $\mathcal{O}(NI^{N-1}J^2)$ and $\mathcal{O}(NI^{N-1}J)$ as their dominant terms, respectively. In contrast, S^3CMTF exploits the sparsity of input data, and has the time complexity linear to the number of entries ($|\Omega_{\mathcal{X}}|$, $|\Omega_{\mathbf{Y}}|$) and is independent of the dimensionality (I) as shown in Lemma 4. Figs 1A and 6A show that the running time (for one iteration) of S^3CMTF on real world data sets follows our theoretical analysis in Section. First, we fix $|\Omega_{\mathcal{X}}|$ to 1M and $|\Omega_{\mathbf{Y}}|$ to 100K, and vary dimensionality I from 1K to 100M. Fig 1A shows the running time (for one iteration) of all methods with $J = 10$. Note that all of our proposed methods achieve constant running time as

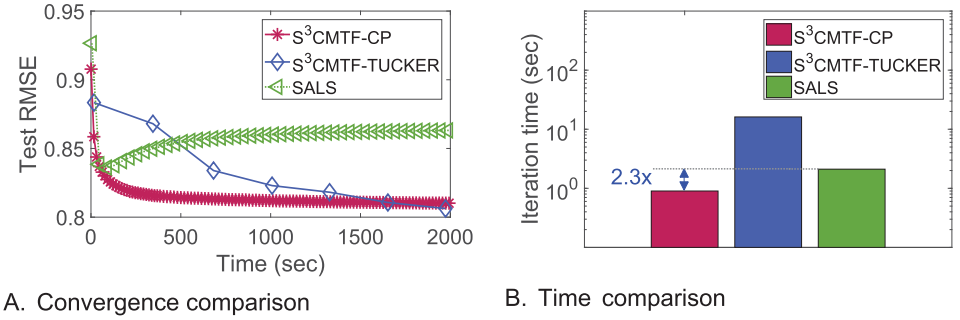

Fig 5. Comparison with SALS-single for movieLens dataset. We compare two non-coupled version of S^3CMTF, S^3CMTF-CP-opt and S^3CMTF-TUCKER-opt with the parallel CP decomposition method, SALS-single. For (a), we set 1 mark per 20 iterations for clarity. (a) S^3CMTF converges faster to a lower error than SALS does. (b) S^3CMTF-CP-opt is 2.3× faster than SALS-single.

https://doi.org/10.1371/journal.pone.0217316.g005

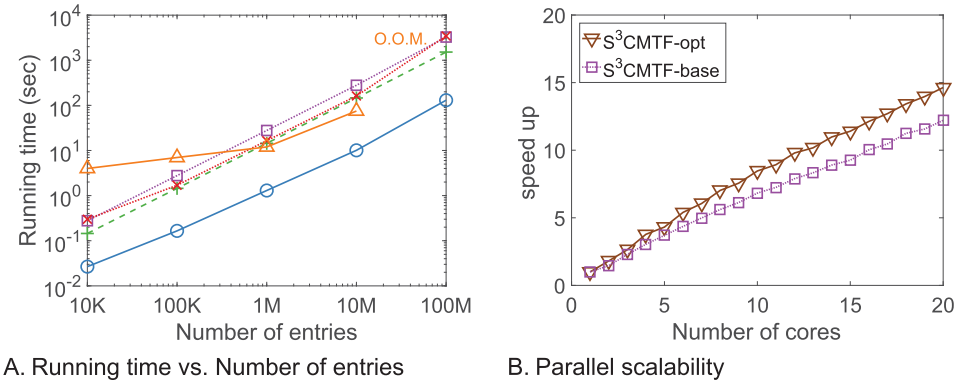

Fig 6. Comparison of scalability. (a) S^3CMTF shows linear scalability as the number of entries increases. (b) S^3CMTF-base and S^3CMTF-opt show linear *speed up* as the number of cores grows. O.O.M.: out of memory error.

https://doi.org/10.1371/journal.pone.0217316.g006

dimensionality increases because they exploit the sparsity of data by updating factors related to only observed data entries. However, CMTF-Tucker-ALS and CMTF-OPT show exponentially increasing running time, and CMTF-OPT shows O.O.M. when $I = 10M$. Next, we investigate the data scalability over the number of entries as shown in Fig 6A. We fix I to 10K and raise $|\Omega_{\mathcal{X}}|$ from 10K to 100M. CMTF-Tucker-ALS shows O.O.M. when $|\Omega_{\mathcal{X}}| = 100M$, and CMTF-OPT shows near-linear scalability. Focusing on the results of S^3CMTF, all three versions of our approach show linear relation between running time and $|\Omega_{\mathcal{X}}|$.

Parallel scalability. We conduct experiments to examine parallel scalability of S^3CMTF on shared memory systems. For measurement, we define *speed up* as *(iteration time on 1 core)/(iteration time)*. Fig 6B shows the linear *speed up* of S^3CMTF-base and S^3CMTF-opt. The slope of the parallel scalability curve is not one (perfectly parallelizable) since the growing number of cores leads to the concurrent read accesses to memory, which leads to conflicts. S^3CMTF-opt shows higher *speed up* than S^3CMTF-base because it reduces reading accesses for core tensor by utilizing intermediate data.

Discovery

In this section, we use S^3CMTF for mining real-world data, Yelp, to answer the question Q3 in the beginning of the previous section. First, we demonstrate that S^3CMTF has better discernment for business entities compared to the naive decomposition method by jointly capturing spatial and categorical prior knowledge. Second, we show how S^3CMTF is possibly applied to the real recommender systems. It is an open challenge to jointly capture the spatio-temporal context along with user preference data [35]. We exemplify a personal recommendation for a specific user. For discovery, we use the total Yelp data tensor along with coupled matrices as explained in Table 4. For better interpretability, we found a non-negative factorization by applying projected gradient method [36]. An orthogonality condition is not imposed to keep non-negativity, and each column of factors is normalized.

Discovery

First, we compare discernment by S^3CMTF and the Tucker decomposition. We use the business factor $\mathbf{U}^{(2)}$. Fig 7A shows the gap statistic values of clustering business entities with k-means clustering algorithm. Gap statistic is a theoretical tool to measure separability between k-means clusters [37]. A higher gap statistic means higher separability between clusters. S^3CMTF shows higher gap statistic values compared to the Tucker decomposition which

Fig 7. (a) Gap statistics on $\mathbf{U}^{(2)}$ of S^3CMTF and the Tucker decomposition for Yelp dataset. S^3CMTF outperforms the naive Tucker decomposition for its clustering ability. (b) Visualization of the personal recommendation scenario.

https://doi.org/10.1371/journal.pone.0217316.g007

means S^3CMTF outperforms the naive Tucker decomposition for entity clustering with respect to the gap statistic.

As the difference between S^3CMTF and the Tucker decomposition is in the existence of coupled matrices, the high performance of S^3CMTF is attributed to the unified factorization using spatial and categorical data as prior knowledge. Table 5 shows the found clusters of business entities. Note that each cluster represents a certain combination of spatial and categorical characteristics of business entities.

User-specific recommendation

Commercial recommendations are one of the most important applications of factorization models [4, 9]. Here we illustrate how factor matrices are used for personalized recommendations with a real example. Fig 7B shows the process for recommendation. Below, we illustrate the process in detail.

Table 5. Clustering results on business factor $\mathbf{U}^{(2)}$ found by S^3CMTF. We found dominant spatial and categorical characteristics from each cluster. Businesses in a same cluster tend to be in adjacent cities and are included in similar categories.

Cluster	Location / Category	Top-10 Businesses
C1	Las Vegas, US/ Travel & Entertainment	Nocturnal Tours, Eureka Casino, Happi Inn, Planet Hollywood Poker Room, Circus Midway Arcade, etc.
C2	Arizona, US/ Real estate & Home services	ENMAR Hardwood Flooring, Sprinkler Dude LLC, Eklund Refrigeration, NR Quality Handyman, The Daniel Montez Real Estate Group, etc.
C11	Ontario, Canada/ Restaurants & Deserts	Jyuban Ramen House, Tim Hortons, Captain John Donlands Fish and Chips, Cora's Breakfast & Lunch, Pho Pad Thai, etc.
C17	Ohio, US/ Food & Drinks	ALDI, Pulp Juice and Smoothie Bar, One Barrel Brewing, Wok N Roll Food Truck, Gas Pump Coffee Company, etc.

https://doi.org/10.1371/journal.pone.0217316.t005

- An example user Tyler has a factor vector **u**, namely user profile, which has been calculated by previous review histories.
- We then calculate the personalized profile matrix $\mathcal{R} = \mathcal{G} \times_1 \mathbf{u} (\in \mathbb{R}^{J_2 \times J_3})$. $\mathcal{R}$ measures the amount of interaction of user profile with business and time factors.
- Norm values of rows in $\mathcal{R}$ indicate the influence of latent business concepts on Tyler. Dominant and weak concepts are found based on the calculated norm values. In the example, B4 is the dominant, and B7 is the weak latent concept.
- We inspect the corresponding columns of business factor matrix $\mathbf{U}^{(2)}$ and find relevant business entities with high values for the found concepts (B4 and B7).

We found both strong and weak entities by the above process. The strong and weak entities provide recommendation information by themselves in the sense that the probability of the user to like strong and weak entities are high and low, respectively, and they also give extended user preference information. For example, strong entities for Tyler are related to 'spa & health' and located in neighborhood cities of Arizona, US. Weak entities are related to 'grill & restaurants' and located in Toronto, Canada. The captured user preference information potentially makes commercial recommender systems interpretable with additional user-specific information such as address, current location among others.

Conclusion

We propose S^3CMTF, a fast, accurate, and scalable CMTF method. S^3CMTF provides up to 930× faster running times and the best accuracy by sparse CMTF with carefully derived update rules, lock-free parallel SGD, and reusing intermediate computation results. S^3CMTF shows linear scalability for the number of data entries and parallel cores. Moreover, we show the usefulness of S^3CMTF for cluster analysis and recommendation by applying S^3CMTF to real-world Yelp data. For future improvements, applying recent achievements in the literature to improve CP gradient algorithm [38, 39] to our method is possible. Also, future works include extending the method to a distributed setting.

Author Contributions

Conceptualization: Dongjin Choi, U Kang.

Data curation: Dongjin Choi.

Formal analysis: Dongjin Choi.

Funding acquisition: U Kang.

Investigation: Dongjin Choi, U Kang.

Methodology: Dongjin Choi, U Kang.

Project administration: U Kang.

Resources: U Kang.

Software: Dongjin Choi.

Supervision: U Kang.

Validation: Dongjin Choi, Jun-Gi Jang, U Kang.

Visualization: Dongjin Choi.

Writing – original draft: Dongjin Choi, Jun-Gi Jang.

Writing – review & editing: Dongjin Choi, U Kang.

References

1. Park N, Jeon B, Lee J, Kang U. BIGtensor: Mining Billion-Scale Tensor Made Easy. In: Proceedings of the International Conference on Information and Knowledge Management. ACM; 2016.
2. Park N, Oh S, Kang U. Fast and Scalable Distributed Boolean Tensor Factorization. In: Data Engineering (ICDE), 2017 IEEE 33rd International Conference on. IEEE; 2017. p. 1071–1082.
3. Oh S, Park N, Sael L, Kang U. Scalable Tucker Factorization for Sparse Tensors—Algorithms and Discoveries. In: Data Engineering (ICDE), 2018 IEEE 34th International Conference on. IEEE; 2018. p. 1120–1131.
4. Koren Y, Bell R, Volinsky C. Matrix factorization techniques for recommender systems. Computer. 2009; 42(8). https://doi.org/10.1109/MC.2009.263
5. Kolda TG, Bader BW. Tensor decompositions and applications. SIAM review. 2009; 51(3):455–500. https://doi.org/10.1137/07070111X
6. Ding C, Li T, Peng W. On the equivalence between non-negative matrix factorization and probabilistic latent semantic indexing. Computational Statistics & Data Analysis. 2008; 52(8):3913–3927. https://doi.org/10.1016/j.csda.2008.01.011
7. Peng W, Li T. On the equivalence between nonnegative tensor factorization and tensorial probabilistic latent semantic analysis. Applied Intelligence. 2011; 35(2):285–295. https://doi.org/10.1007/s10489-010-0220-9
8. Xu W, Liu X, Gong Y. Document clustering based on non-negative matrix factorization. In: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval. ACM; 2003. p. 267–273.
9. Karatzoglou A, Amatriain X, Baltrunas L, Oliver N. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In: Proceedings of the fourth ACM conference on Recommender systems. ACM; 2010. p. 79–86.
10. Rendle S, Schmidt-Thieme L. Pairwise interaction tensor factorization for personalized tag recommendation. In: Proceedings of the third ACM international conference on Web search and data mining. ACM; 2010. p. 81–90.
11. Sael L, Jeon I, Kang U. Scalable tensor mining. Big Data Research. 2015; 2(2):82–86. https://doi.org/10.1016/j.bdr.2015.01.004
12. Acar E, Kolda TG, Dunlavy DM. All-at-once optimization for coupled matrix and tensor factorizations. arXiv preprint arXiv:11053422. 2011.
13. Acar E, Rasmussen MA, Savorani F, Næs T, Bro R. Understanding data fusion within the framework of coupled matrix and tensor factorizations. Chemometrics and Intelligent Laboratory Systems. 2013; 129:53–63. https://doi.org/10.1016/j.chemolab.2013.06.006
14. Narita A, Hayashi K, Tomioka R, Kashima H. Tensor factorization using auxiliary information. Data Mining and Knowledge Discovery. 2012; 25(2):298–324. https://doi.org/10.1007/s10618-012-0280-z
15. Ozcaglar C. Algorithmic data fusion methods for tuberculosis. Rensselaer Polytechnic Institute; 2012.
16. Tucker LR. Some mathematical notes on three-mode factor analysis. Psychometrika. 1966; 31(3):279–311. https://doi.org/10.1007/BF02289464 PMID: 5221127
17. Oh J, Shin K, Papalexakis EE, Faloutsos C, Yu H. S-HOT: Scalable High-Order Tucker Decomposition. In: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining. ACM; 2017. p. 761–770.
18. Hitchcock FL. The expression of a tensor or a polyadic as a sum of products. Studies in Applied Mathematics. 1927; 6(1-4):164–189.
19. Sorber L, Van Barel M, De Lathauwer L. Structured data fusion. IEEE Journal of Selected Topics in Signal Processing. 2015; 9(4):586–600. https://doi.org/10.1109/JSTSP.2015.2400415
20. Kolda TG, Sun J. Scalable tensor decompositions for multi-aspect data mining. In: Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on. IEEE; 2008. p. 363–372.
21. De Lathauwer L, De Moor B, Vandewalle J. On the best rank-1 and rank-(r 1, r 2,. . ., rn) approximation of higher-order tensors. SIAM journal on Matrix Analysis and Applications. 2000; 21(4):1324–1342. https://doi.org/10.1137/S0895479898346995

22. Ermiş B, Acar E, Cemgil AT. Link prediction in heterogeneous data via generalized coupled tensor factorization. Data Mining and Knowledge Discovery. 2015; 29(1):203–236. https://doi.org/10.1007/s10618-013-0341-y

23. Yılmaz KY, Cemgil AT, Simsekli U. Generalised coupled tensor factorisation. In: Advances in neural information processing systems; 2011. p. 2151–2159.

24. Khan SA, Leppäaho E, Kaski S. Bayesian multi-tensor factorization. Machine Learning. 2016; 105(2):233–253. https://doi.org/10.1007/s10994-016-5563-y

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

26. Jeon I, Papalexakis EE, Kang U, Faloutsos C. Haten2: Billion-scale tensor decompositions. In: Data Engineering (ICDE), 2015 IEEE 31st International Conference on. IEEE; 2015. p. 1047–1058.

27. Papalexakis EE, Faloutsos C, Mitchell TM, Talukdar PP, Sidiropoulos ND, Murphy B. Turbo-smt: Accelerating coupled sparse matrix-tensor factorizations by 200x. In: Proceedings of the 2014 SIAM International Conference on Data Mining. SIAM; 2014. p. 118–126.

28. Beutel A, Talukdar PP, Kumar A, Faloutsos C, Papalexakis EE, Xing EP. Flexifact: Scalable flexible factorization of coupled tensors on hadoop. In: Proceedings of the 2014 SIAM International Conference on Data Mining. SIAM; 2014. p. 109–117.

29. Jeon I, Papalexakis EE, Faloutsos C, Sael L, Kang U. Mining billion-scale tensors: algorithms and discoveries. The VLDB Journal. 2016; 25(4):519–544. https://doi.org/10.1007/s00778-016-0427-4

30. Shin K, Sael L, Kang U. Fully scalable methods for distributed tensor factorization. IEEE Transactions on Knowledge and Data Engineering. 2017; 29(1):100–113. https://doi.org/10.1109/TKDE.2016.2610420

31. Bradley JK, Kyrola A, Bickson D, Guestrin C. Parallel coordinate descent for l1-regularized loss minimization. arXiv preprint arXiv:11055379. 2011.

32. Recht B, Re C, Wright S, Niu F. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In: Advances in neural information processing systems; 2011. p. 693–701.

33. Bottou L. Stochastic gradient descent tricks. In: Neural networks: Tricks of the trade. Springer; 2012. p. 421–436.

34. Bader BW, Kolda TG. Efficient MATLAB computations with sparse and factored tensors. SIAM Journal on Scientific Computing. 2007; 30(1):205–231. https://doi.org/10.1137/060676489

35. Gao H, Tang J, Hu X, Liu H. Exploring temporal effects for location recommendation on location-based social networks. In: Proceedings of the 7th ACM conference on Recommender systems. ACM; 2013. p. 93–100.

36. Lin CJ. Projected gradient methods for nonnegative matrix factorization. Neural computation. 2007; 19 (10):2756–2779. https://doi.org/10.1162/neco.2007.19.10.2756 PMID: 17716011

37. Tibshirani R, Walther G, Hastie T. Estimating the number of clusters in a data set via the gap statistic. Journal of the Royal Statistical Society: Series B (Statistical Methodology). 2001; 63(2):411–423. https://doi.org/10.1111/1467-9868.00293

38. Vannieuwenhoven N, Meerbergen K, Vandebril R. Computing the gradient in optimization algorithms for the CP decomposition in constant memory through tensor blocking. SIAM Journal on Scientific Computing. 2015; 37(3):C415–C438. https://doi.org/10.1137/14097968X

39. Phan AH, Tichavskỳ P, Cichocki A. Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations. IEEE Transactions on Signal Processing. 2013; 61(19):4834–4846. https://doi.org/10.1109/TSP.2013.2269903