SGD_Tucker: A Novel Stochastic Optimization Strategy for Parallel Sparse Tucker Decomposition

Hao Li, *Student Member, IEEE*, Zixuan Li, Kenli Li, *Senior Member, IEEE*, Jan S. Rellermeyer, *Member, IEEE*, Lydia Chen, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

Abstract—Sparse Tucker Decomposition (STD) algorithms learn a core tensor and a group of factor matrices to obtain an optimal low-rank representation feature for the High-Order, High-Dimension, and Sparse Tensor (HOHDST). However, existing STD algorithms face the problem of intermediate variables explosion which results from the fact that the formation of those variables, i.e., matrices Khatri-Rao product, Kronecker product, and matrix-matrix multiplication, follows the whole elements in sparse tensor. The above problems prevent deep fusion of efficient computation and big data platforms. To overcome the bottleneck, a novel stochastic optimization strategy (SGD_Tucker) is proposed for STD which can automatically divide the high-dimension intermediate variables into small batches of intermediate matrices. Specifically, SGD_Tucker only follows the randomly selected small samples rather than the whole elements, while maintaining the overall accuracy and convergence rate. In practice, SGD_Tucker features the two distinct advancements over the state of the art. First, SGD_Tucker can prune the communication overhead for the core tensor in distributed settings. Second, the low data-dependence of SGD_Tucker enables fine-grained parallelization, which makes SGD_Tucker obtaining lower computational overheads with the same accuracy. Experimental results show that SGD_Tucker runs at least $2X$ faster than the state of the art.

Index Terms—High-order, high-dimension and sparse tensor, low-rank representation learning, machine learning algorithm, sparse tucker decomposition, stochastic optimization, parallel strategy

1 Introduction

Tensors are a widely used data representation style for interaction data in the Machine Learning (ML) application community [1], e.g, in Recommendation Systems [2], Quality of Service (QoS) [3], Network Flow [4], Cyber-Physical-Social (CPS) [5], or Social Networks [6]. In addition to applications in which the data is naturally represented in the form of tensors, another common used case is the fusion in multi-view or multi-modality problems [7]. Here, during the learning process, each modality corresponds to a feature and the feature alignment involves fusion. Tensors are a common form of feature fusion for multi-modal learning [7], [8], [9], [10]. Unfortunately, tensors can be difficult to process in practice. For instance, an N-order tensor comprises of the interaction relationship between N kinds of attributes and if each attribute has millions of items, this results in a substantially large size of data [11]. As a remedy, *dimensionality reduction* can be used to represent the original state using much fewer parameters [12].

Specifically in the ML community, Tensor Factorization (TF), as a classic dimensionality reduction technique, plays a key role for low-rank representation. Xu *et al.*, [13] proposed a Spatio-temporal multi-task learning model via TF and in this work, tensor data is of 5-order, i.e., weather, traffic volume, crime rate, disease incidents, and time. Meanwhile, this model made predictions through the time-order for the multi-task in weather, traffic volume, crime rate, and disease incidents orders and the relationship construction between those orders is via TF. In the community of Natural Language Processing (NLP), Liu *et al.*, [14] organized a mass of texts into a tensor and each slice is modeled as a sparse symmetric matrix. Furthermore, the tensor representation is a widely-used form for Convolutional Neural Networks (CNNs), e.g., in the popular TensorFlow [15] framework, and Kossaifi *et al.*, [16] took tensorial parametrization of a CNNs and pruned the parametrization by Tensor Network Decomposition (TND). Meanwhile, Ju, *et al.*, [17] pruned and then accelerated the Restricted Boltzmann Machine (RBM) coupling with TF. In the Computer Vision (CV) community, Wang *et al.*, [18] modeled various factors, i.e., pose and illumination, as an unified tensor and make disentangled representation by adversarial autoencoder via TF. Zhang *et al.*, [19] constructed multi subspaces of

- *Hao Li is with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China, the National Supercomputing Center, Changsha, Hunan 410082, China, and also with the TU Delft, 2628 CD Delft, Netherlands. E-mail: lihao123@hnu.edu.cn.*
- *Zixuan Li and Kenli Li are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China, and also with the National Supercomputing Center, Changsha, Hunan 410082, China. E-mail: {zixuanli, lkl}@hnu.edu.cn.*
- *Jan S. Rellermeyer and Lydia Chen are with the TU Delft, 2628 CD Delft, Netherlands. E-mail: {j.s.rellermeyer, Y.Chen-10}@tudelft.nl.*
- *Keqin Li is with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China, the National Supercomputing Center, Changsha, Hunan 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA. E-mail: lik@newpaltz.edu.*

Manuscript received 1 July 2020; revised 23 Oct. 2020; accepted 30 Nov. 2020. Date of publication 25 Dec. 2020; date of current version 11 Feb. 2021. (Corresponding author: Kenli Li.)
Recommended for acceptance by P. Balaji, J. Zhai, and M. Si.
Digital Object Identifier no. 10.1109/TPDS.2020.3047460

1045-9219 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: TU Delft Library. Downloaded on February 21,2021 at 18:47:34 UTC from IEEE Xplore. Restrictions apply.

multi-view data and then abstract factor matrices via TF for the unified latent of each view.

High-Order, High-Dimension, and Sparse Tensor (HOHDST) is a common situation in the big-data processing and ML application community [20], [21]. Dimensionality reduction can also be used to find the low-rank space of HOHDST in ML applications [22], which can help to make prediction from existing data. Therefore, it is non-trivial to learn the pattern from the existed information in a HOHDST and then make the corresponding prediction. Sparse Tucker Decomposition (STD) is one of the most popular TF models for HOHDST, which can find the N-coordinate systems and those systems are tangled by a core tensor between each other [23]. Liu *et al.*, [24] proposed to accomplish the visual tensor completion via STD. The decomposition process of STD involves the entanglement of N-factor matrices and core tensor and the algorithms follow one of the following two directions: 1) search for optimal orthogonal coordinate system of factor matrices, e.g., High-order Orthogonal Iteration (HOOI) [25]; 2) designing optimization solving algorithms [26]. HOOI is a common solution for STD [27] and able to find the N orthogonal coordinate systems which are similar to Singular Value Decomposition (SVD), but requires frequent intermediate variables of Khatri-Rao and Kronecker products [28].

An interesting topic is that stochastic optimization [29], e.g., Count Sketch and Singleshot [30], [31], etc. can alleviate this bottleneck to a certain extent, depending on the size of dataset. However, those methods depend on the count sketch matrix, which cannot be easily implemented in a distributed environment and is notoriously difficult to parallelize. The Stochastic Gradient Descent (SGD) method can approximate the gradient from randomly selected subset and it forms the basis of the state of art methods, e.g., variance SGD [32], average SGD [33], Stochastic Recursive Gradient [34], and momentum SGD [35]. SGD is adopted to approximate the eventual optimal solver with lower space and computational complexities; meanwhile, the low data-dependence makes the SGD method amenable to parallelization [36]. The idea of construction for the computational graph of the mainstream platforms, e.g., Tensorflow [15] and Pytorch [37], is based on the SGD [38] and practitioners have already demonstrated its powerful capability on large-scale optimization problems. The computational process of SGD only needs a batch of training samples rather than the full set which gives the ML algorithm the low-dependence between each data block and low communication overhead [39].

There are three challenges to process HOHDST in a fast and accurate way: 1) how to define a suitable optimization function to find the optimal factor matrices and core tensor? 2) how to find an appropriate optimization strategy in a low-overhead way and then reduce the entanglement of the factor matrices with core tensor which may produce massive intermediate matrices? 3) how to parallelize STD algorithms and make distributed computation with low communication cost? In order to solve these problems, we present the main contributions of this work which are listed as follows:

1) A novel optimization objective for STD is presented. This proposed objective function not only has a low number of parameters via coupling the Kruskal product (Section 4.1) but also is approximated as a convex function;
2) A novel stochastic optimization strategy is proposed for STD, SGD_Tucker, which can automatically divide the high-dimension intermediate variables into small batches of intermediate matrices that only follows the index of the randomly selected small samples; meanwhile, the overall accuracy and convergence are kept (Section 4.3);
3) The low data-dependence of SGD_Tucker creates opportunities for fine-grained parallelization, which makes SGD_Tucker obtaining lower computational overhead with the same accuracy. Meanwhile, SGD_Tucker does not rely on the specific compressive structure of a sparse tensor (Section 4.4).

To our best knowledge, SGD_Tucker is the first work that can divide the high-dimension intermediate matrices of STD into small batches of intermediate variables, a critical step for fine-grained parallelization with low communication overhead. In this work, the related work is presented in Section 2. The notations and preliminaries are introduced in Section 3. The SGD_Tucker model as well as parallel and communication overhead on distributed environment for STD are showed in Section 4. Experimental results are illustrated in Section 5.

2 Related Studies

For HOHDST, there are many studies to accelerate STD on the state of the art parallel and distributed platforms, i.e., OpenMP, MPI, CUDA, Hadoop, Spark, and OpenCL. Ge *et al.*, [40] proposed distributed CANDECOMP/PARAFAC Decomposition (CPD) which is a special STD for HOHDST. Shaden *et al.*, [41] used a Compressed Sparse Tensors (CSF) structure which can optimize the access efficiency for HOHDST. Tensor-Time-Matrix-chain (TTMc) [42] is a key part for Tucker Decomposition (TD) and TTMc is a data intensive task. Ma *et al.*, [42] optimized the TTMc operation on GPU which can take advantage of intensive and partitioned computational resource of GPU, i.e., a warp threads (32) are automatically synchronized and this mechanism is apt to matrices block-block multiplication. Non-negative Tucker Decomposition (NTD) can extract the non-negative latent factor of a HOHDST, which is widely used in ML community. However, NTD need to construct the numerator and denominator and both of them involve TTMc. Chakaravarthy *et al.*, [43] designed a mechanism which can divide the TTMc into multiple independent blocks and then those tasks are allocated to distributed nodes.

HOOI is a common used TF algorithm which comprises of a series of TTMc matrices and SVD for the TTMc. [44] focused on dividing the computational task of TTMc into a list of independent parts and a distributed HOOI for HOHDST is presented in [45]. However, the intermediate cache memory for TTMc of HOOI increased explosively. Shaden *et al.*, [41] presented a parallel HOOI algorithm and this work used a Compressed Sparse Tensors structure which can improve the access efficiency and save the memory for HOHDST. Oh and Park [46], [47] presented a parallelization strategy of ALS and CD for STD on OpenMP. A heterogeneous OpenCL parallel version of ALS for STD is proposed on [48]. However, the above works are still focused on the naive algorithm parallelization which cannot solve the problem of fine-grained parallelization.

Authorized licensed use limited to: TU Delft Library. Downloaded on February 21,2021 at 18:47:34 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Table of Symbols

Symbol	Definition
I_n	The size of row in the nth factor matrix;
J_n	The size of column in the nth factor matrix;
$\mathcal{X}$	Input N order tensor $\in \mathbb{R}_+^{I_1 \times I_2 \times \cdots \times I_N}$;
$x_{i_1,i_2,\ldots,i_n}$	$i_1, i_2, \ldots, i_n$th element of tensor $\mathcal{X}$;
$\mathcal{G}$	Core N order tensor $\in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$;
$\mathbf{X}$	Input matrix $\in \mathbb{R}_+^{I_1 \times I_2}$;
$\mathbf{X}^{(n)}$	nth unfolding matrix for tensor $\mathcal{X}$;
$\mathrm{Vec}_n(\mathcal{X})$	nth vectorization of a tensor $\mathcal{X}$;
Ω	Index $(i_1, \ldots, i_n, \ldots, i_N)$ of a tensor $\mathcal{X}$;
$\Omega_M^{(n)}$	Index (i_n, j) of nth unfolding matrix $\mathbf{X}^{(n)}$;
$(\Omega_M^{(n)})_i$	Column index set in ith row of $\Omega_M^{(n)}$;
$(\Omega_M^{(n)})^j$	Row index set in jth column of $\Omega_M^{(n)}$;
$\Omega_V^{(n)}$	Index i of nth unfolding vector $\mathrm{Vec}_n(\mathcal{X})$;
$\{N\}$	The ordered set $\{1, 2, \ldots, N-1, N\}$;
$\mathbf{A}^{(n)}$	nth feature matrix $\in \mathbb{R}^{I_n \times J_n}$;
$a_{i_n,:}^{(n)}$	i_nth row vector $\in \mathbb{R}^{K_n}$ of $\mathbf{A}^{(n)}$;
$a_{:,j}^{(n)}$	jth column vector $\in \mathbb{R}^{K_n}$ of $\mathbf{A}^{(n)}$;
$a_{i_n,k_n}^{(n)}$	k_nth element of feature vector $a_{i_n,:}^{(n)}$;
$\cdot/(-,/)$	Element-wise multiplication/ division;
$\circ$	Outer production of vectors;
$\odot$	Khatri-Rao (columnwise Kronecker) product;
$\times$	Matrix product;
$\times_{(n)}$	n-Mode Tensor-Matrix product;
$\otimes$	Kronecker product.

3 Notation and Preliminaries

The main notations include the tensors, matrices and vectors, along with their basic elements and operations (in Table 1). Fig. 1 illustrates the details of TD including the tanglement of the core tensor $\mathcal{G}$ with the N factor matrices $\mathbf{A}^{(n)}$, $n \in \{N\}$. The data styles for TF include sparse and dense tensors and STD is devoted to HOHDST. Here, basic definitions for STD and models are rendered in Section 3.1. Finally, the STD process for the HOHDST is illustrated in Section 3.2.

3.1 Basic Definitions

Definition 1 (Tensor Unfolding (Matricization)). *nth tensor unfolding (Matricization) refers to that a low order matrix $\mathbf{X}^{(n)} \in \mathbb{R}^{I_n \times I_1 \cdots I_{n-1} \cdot I_{n+1} \cdots I_N}$ stores all information of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots I_n \cdots \times I_N}$ and the matrix element $x_{i_n,j}^{(n)}$ of $\mathbf{X}^{(n)}$ at the position $j = 1 + \sum_{k=1,n\neq k}^{N}[(i_k - 1)\prod_{m=1,m\neq n}^{k-1} I_m]$ contains the tensor element $x_{i_1,i_2,\ldots,i_n}$ of a tensor $\mathcal{X}$.*

Definition 2 (Vectorization of a tensor $\mathcal{X}$). *nth tensor vectorization refers to that a vector $x^{(n)}$ ($Vec_n(\mathcal{X})$ and $Vec(\mathbf{X}^{(n)})$) stores all elements in the nth matricization $\mathbf{X}^{(n)}$ of a tensor $\mathcal{X}$ and $x_k^{(n)} = \mathbf{X}_{i,j}^{(n)}$, where $k = (j-1)I_n + i$.*

Definition 3 (Tensor Approximation). *A N-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ can be approximated by $\widehat{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, as well as a N-order residual or noisy tensor $\mathcal{E} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$. The low-rank approximation problem is defined as $\mathcal{X} = \widehat{\mathcal{X}} + \mathcal{E}$, where $\widehat{\mathcal{X}}$ is denoted by a low-rank tensor.*

Definition 4 (n-Mode Tensor-Matrix product). *n-Mode Tensor-Matrix product is an operation which can reflect coordinate projection of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ with projection matrix $\mathbf{U} \in \mathbb{R}^{I_n \times J_n}$ into a tensor $(\mathcal{X} \times_{(n)} \mathbf{U}) \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J_n \times \cdots I_N}$ where $(\mathcal{X} \times_{(n)} \mathbf{U})_{i_1 \times \cdots \times i_{n-1} \times j_n \times i_{n+1} \times \cdots \times i_N} = \sum_{i_n=1}^{I_n} x_{i_1 \times \cdots \times i_n \times \cdots \times i_N} u_{j_n,i_n}$.*

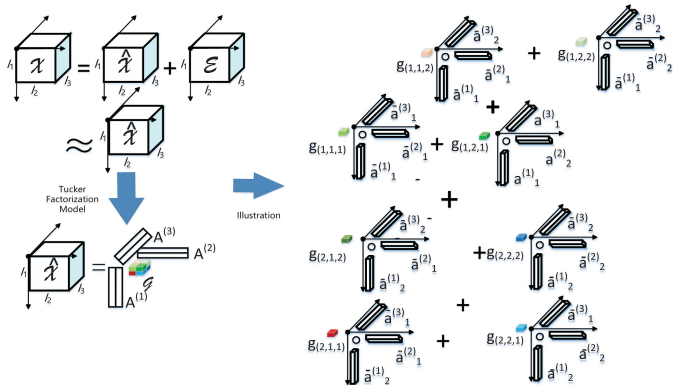


Fig. 1. Illustration of TD.

Definition 5 (R Kruskal Product). *For an N-order tensor $\widehat{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, the R Kruskal Product of $\widehat{\mathcal{X}}$ is given by R Kruskal product as: $\widehat{\mathcal{X}} = \sum_{r=1}^{R} a_{:,r}^{(1)} \circ \cdots \circ a_{:,r}^{(n)} \circ \cdots \circ a_{:,r}^{(N)}$.*

Definition 6 (Sparse Tucker Decomposition). *For a N-order sparse tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, the STD of the optimal approximated $\widehat{\mathcal{X}}$ is given by $\widehat{\mathcal{X}} = \mathcal{G} \times_{(1)} \mathbf{A}^{(1)} \times_{(2)} \cdots \times_{(n)} \mathbf{A}^{(n)} \times_{(n+1)} \cdots \times_{(N)} \mathbf{A}^{(N)}$, where $\mathcal{G}$ is the core tensor and $\mathbf{A}^{(n)}$, $n \in \{N\}$ are the low-rank factor matrices. The rank of TF for a tensor is $[J_1, \ldots, J_n, \ldots, J_N]$. The determination process for the core tensor $\mathcal{G}$ and factor matrices $\mathbf{A}^{(n)}$, $n \in \{N\}$ follows the sparsity model of the sparse tensor $\mathcal{X}$.*

In the convex optimization community, the literature [49] gives the definition of Lipschitz-continuity with constant L and strong convexity with constant μ.

Definition 7 (L-Lipschitz continuity). *A continuously differentiable function $f(\mathbf{x})$ is called L-smooth on $\mathbb{R}^r$ if the gradient $\nabla f(\mathbf{x})$ is L-Lipschitz continuous for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^r$, that is $\| \nabla f(\mathbf{x}) - \nabla f(\mathbf{y}) \|_2 \leq L \| \mathbf{x} - \mathbf{y} \|_2$, where $\| \bullet \|_2$ is L_2-norm $\|\mathbf{x}\|_2 = (\sum_{k=1}^{r} x_k^2)^{1/2}$ for a vector $\mathbf{x}$.*

Definition 8 (μ-Convex). *A continuously differentiable function $f(\mathbf{x})$ is called strongly-convex on $\mathbb{R}^r$ if there exists a constant $\mu > 0$ for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^r$, that is $f(\mathbf{x}) \geq f(\mathbf{y}) + \nabla f(\mathbf{y}) (\mathbf{x} - \mathbf{y})^T + \frac{1}{2}\mu\|\mathbf{x} - \mathbf{y}\|_2^2$.*

Due to limited spaces, we provide the description of basic optimization as the supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2020.3047460.

3.2 Problems for Large-Scale Optimization Problems

Many ML tasks are transformed into the solvent of optimization problem [32], [33], [34], [35] and the basis optimization problem is organized as

$$\begin{aligned} \arg\min_{w \in \mathbb{R}^R} f(w) &= \underbrace{L\left(w \middle| y_i, x_i, w\right)}_{Loss\ Function} + \underbrace{\lambda_w R(w)}_{Regularization\ Item} \\ &= \sum_{i=1}^{N} L_i\left(w \middle| y_i, x_i, w\right) + \lambda_w R_i(w), \end{aligned} \tag{1}$$

Authorized licensed use limited to: TU Delft Library. Downloaded on February 21,2021 at 18:47:34 UTC from IEEE Xplore. Restrictions apply.

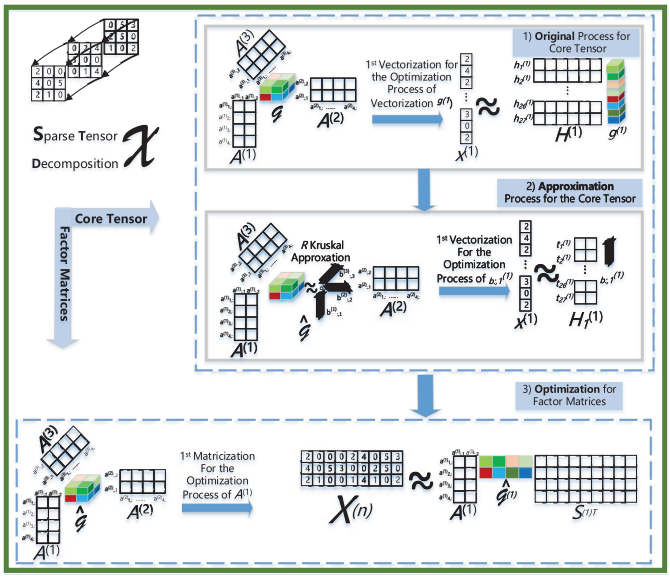


Fig. 2. Illustration of Optimization Process for SGD_Tucker: 1) Original problem for core tensor. 2) Kruskal product for approximating core tensor. 3) Optimization process for factor matrices.

where $y_i \in \mathbb{R}^1$, $x_i \in \mathbb{R}^R$, $i \in \{N\}$, $w \in \mathbb{R}^R$ and the original optimization model needs gradient which should select all the samples $\{x_i | i \in \{N\}\}$ from the dataset Ω and the GD is presented as

$$\begin{aligned} w &\leftarrow w - \gamma \frac{\partial f_{\Omega}(w)}{\partial w} \\ &= w - \gamma \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \Big(L_i(w) + \lambda_w R_i(w) \Big)}{\partial w}. \end{aligned} \tag{2}$$

The second-order solution, i.e., ALS and CD, etc, are deduced from the construction of GD from the whole dataset. In large-scale optimization scenarios, SGD is a common strategy [32], [33], [34], [35] and promises to obtain the optimal accuracy via a certain number of training epoches [32], [33], [34], [35]. An M entries set Ψ is randomly selected from the set Ω, and the SGD is presented as

$$\begin{aligned} w &\leftarrow w - \gamma \frac{\partial f_{\Psi}(w)}{\partial w} \\ &\approx w - \gamma \frac{1}{M} \sum_{i \in \Psi} \frac{\partial \Big(L_i(w) + \lambda_w R_i(w) \Big)}{\partial w}. \end{aligned} \tag{3}$$

Equ. (3) is an average SGD [33], and the averaged SG can be applied to build the basic tool of the modern stochastic optimization strategies, e.g., Stochastic Recursive Gradient [34], variance SGD [32], or momentum SGD [35], which can retain robustness and fast convergence. The optimization function can be packaged in the form of $SGD(M, \lambda, \gamma, w, \frac{\partial f_{\Psi}(w)}{\partial w})$.

4 SGD_TUCKER

Overview. In this section, SGD_Tucker is proposed to decompose the optimization objectives which involves frequent operations of intermediate matrices into a problem which only needs the manipulations of small batches of intermediate matrices. Fig. 2 illustrates the framework for our work. As Fig. 2 shows, SGD_Tucker comprises of an approximation process of the core tensor and an optimization for factor matrices. Table 2 records the intermediate variables which follow the problem deduction process. Section 4.1 presents the deduction process for the core tensor (Lines 1-16, Algorithm 1), and Section 4.2 presents the proceeding process for the factor matrices of SGD_Tucker (Lines 17-26, Algorithm 1). Section 4.3 shows the stochastic optimization process for the proposed novel optimization objectives, which illustrates the details of the process that how can the SGD_Tucker divide the high-dimension intermediate matrices $\{\mathbf{H}^{(n)}, \mathbf{S}^{(n)}, \mathbf{E}^{(n)} | n \in \{N\}\}$ into a small batches of intermediate matrices. Section 4.4 shows the parallel and distributed strategies for SGD_Tucker. Finally, Section 4.5 illustrates the analysis of space and time complexities. Because the optimization process for approximating the core tensor and factor

TABLE 2
Table of Intermediate Varibles for SGD_Tucker

Symbol	Description	Emerging Equations
$\widehat{\mathcal{G}} \in \mathbb{R}_{+}^{J_1 \times J_2 \times \cdots \times J_N}$	$\mathbb{R}_{core}$ Kruskal product for $\mathcal{G}$	Equ. (4);
$\mathbf{B}^{(n)} \in \mathbb{R}^{J_n \times R_{core}}$	nth Kruskal matrix for $\widehat{\mathcal{G}}$	Equ. (4); ;
$\widehat{g}^{(n)} \in \mathbb{R}^{\prod_{n=1}^{N} J_n}$	nth vectorization of tensor $\widehat{\mathcal{G}}$	Equ. (5);
$\mathbf{H}^{(n)} \in \mathbb{R}^{\prod_{n=1}^{N} I_n \times \prod_{n=1}^{N} J_n}$	Coefficient for $\widehat{g}^{(n)}$	Equ. (5);
$x^{(n)} \in \mathbb{R}^{\prod_{n=1}^{N} I_n}$	nth vectorization of Tensor $\mathcal{X}$	Equ. (5);
$\mathbf{Q}^{(n)} \in \mathbb{R}^{\prod_{r=1, r \neq n}^{N} J_r \times R_{core}}$	Coefficient of $\mathbf{B}^{(n)}$ for constructing $\widehat{g}^{(n)}$	Equ. (6);
$\mathbf{U}^{(n)} \in \mathbb{R}^{J_n \times J_n}$	Unity matrix	Equ. (7);
$\mathbf{O}_r^{(n)} \in \mathbb{R}^{\prod_{r=1}^{N} J_r \times J_n}$	Coefficient of $b_{:,r}^{(n)}$ for constructing $\widehat{g}^{(n)}$	Equ. (8);
$x_{r_{core}}^{(n)} \in \mathbb{R}^{\prod_{r=1}^{N} I_r}$	Intermediate vector of $x^{(n)}$	Equ. (9); ;
$\mathbf{H}_{r_{core}}^{(n)} \in \mathbb{R}^{\prod_{n=1}^{N} I_n \times J_n}$	Coefficient of $b_{:,r}^{(n)}$ for constructing $\widehat{g}^{(n)}$	Equ. (9);
$\mathbf{S}^{(n)} \in \mathbb{R}^{\prod_{k=1, k \neq n}^{N} J_n \times \prod_{k=1, k \neq n}^{N} I_k}$	Coefficient of $\mathbf{A}^{(n)} \widehat{\mathbf{G}}^{(n)}$ for constructing $\widehat{\boldsymbol{X}}^{(n)}$	Equ. (10);
$\mathbf{E}^{(n)} \in \mathbb{R}^{J_n \times \prod_{k=1, k \neq n}^{N} I_k}$	Coefficient of $\mathbf{A}^{(n)}$ for constructing $\widehat{\boldsymbol{X}}^{(n)}$	Equ. (11);
$x_{\Psi_V^{(n)}}^{(n)} \in \mathbb{R}^{\lvert \Psi_V^{(n)} \rvert}$	Partial vector $x^{(n)}$ from set Ψ	Equ. (14); ;
$\mathbf{H}_{\Psi_V^{(n)},:}^{(n)} \in \mathbb{R}^{\lvert \Psi_V^{(n)} \rvert \times \prod_{n=1}^{N} J_n}$	Partial matrix of $\mathbf{H}^{(n)}$	Equ. (14);
$\boldsymbol{X}_{:,(\Psi_M^{(n)})_{i_n}}^{(n)} \in \mathbb{R}^{I_n \times \lvert (\Psi_M^{(n)})_{i_n} \rvert}$	Partial matrix of $\boldsymbol{X}^{(n)}$	Equ. (17);
$\mathbf{E}_{:,(\Psi_M^{(n)})_{i_n}}^{(n)} \in \mathbb{R}^{J_n \times \lvert (\Psi_M^{(n)})_{i_n} \rvert}$	Partial matrix of $\mathbf{E}^{(n)}$	Equ. (17).

Authorized licensed use limited to: TU Delft Library. Downloaded on February 21,2021 at 18:47:34 UTC from IEEE Xplore. Restrictions apply.

matrices are non-convex. We fix the core tensor and optimize a factor matrix and fix factor matrices to optimize Kruskal product for core tensor. This alternative minimization strategy is a convex solution [46], [47], [48], [50].

4.1 Optimization Process for the Core Tensor

Due to the non-trivial coupling of the core tensor $\mathcal{G}$, the efficient and effective way to infer it is through an approximation. A tensor $\mathcal{G}$ can be approximated by $R_{core} \leq J_n, n \in \{N\}$ Kruskal product of low-rank matrices $\{\mathbf{B}^{(n)} \in \mathbb{R}^{J_n \times R_{core}} | n \in \{N\}\}$ to form $\widehat{\mathcal{G}}$

$$\widehat{\mathcal{G}} = \sum_{r_{core}=1}^{R_{core}} b_{:,r_{core}}^{(1)} \circ \cdots \circ b_{:,r_{core}}^{(n)} \circ \cdots \circ b_{:,r_{core}}^{(N)}. \tag{4}$$

As the direct approximation for the core tensor $\mathcal{G}$ may result in instability, we propose to apply the coupling process to approximate STD and tensor approximation. Specifically, we use Kruskal product of low-rank matrices $\{\mathbf{B}^{(n)} \in \mathbb{R}^{J_n \times R_{core}} | n \in \{N\}\}$ as follows:

$$\begin{aligned} &\arg\min_{\widehat{\mathcal{G}}} f\left(\widehat{g}^{(n)} \middle| x^{(n)}, \{\mathbf{A}^{(n)}\}\right) \\ &\quad = \left\| x^{(n)} - \mathbf{H}^{(n)} \widehat{g}^{(n)} \right\|_2^2 + \lambda_{\widehat{g}^{(n)}} \left\| \widehat{g}^{(n)} \right\|_2^2, \end{aligned} \tag{5}$$

where $\widehat{g}^{(n)} = Vec(\mathbf{B}^{(n)} \mathbf{Q}^{(n)^T})$ and $\mathbf{Q}^{(n)} = \mathbf{B}^{(N)} \odot \cdots \odot \mathbf{B}^{(n+1)} \odot \mathbf{B}^{(n-1)} \odot \cdots \odot \mathbf{B}^{(1)}$. The tanglement problem resulted from R_{core} Kruskal product of low-rank matrices $\{\mathbf{B}^{(n)} \in \mathbb{R}^{J_n \times R_{core}} | n \in \{N\}\}$ leads to a non-convex optimization problem for the optimization objective (5). The alternative optimization strategy [50] is adopted to update the parameters $\{\mathbf{B}^{(n)} \in \mathbb{R}^{J_n \times R_{core}} | n \in \{N\}\}$ and then the non-convex optimization objective (5) is turned into the objective as

$$\begin{aligned} &\arg\min_{\mathbf{B}^{(n)}, n \in \{N\}} f\left(\mathbf{B}^{(n)} \middle| x^{(n)}, \{\mathbf{A}^{(n)}\}, \{\mathbf{B}^{(n)}\}\right) \\ &\quad = \left\| x^{(n)} - \mathbf{H}^{(n)} Vec(\mathbf{B}^{(n)} \mathbf{Q}^{(n)^T}) \right\|_2^2 + \lambda_{\mathbf{B}} \left\| \mathbf{B} \right\|_2^2, \end{aligned} \tag{6}$$

where $Vec(\mathbf{B}^{(n)} \mathbf{Q}^{(n)^T}) = Vec(\sum_{r=1}^{R_{core}} b_{:,r}^{(n)} q_{:,r}^{(n)^T})$.

The optimization objective (6) is not explicit for the variable $\mathbf{B}^{(n)}, n \in \{N\}$ and it is hard to find the gradient of the variables $\mathbf{B}^{(n)}, n \in \{N\}$ under the current formulation. Thus, we borrow the intermediate variables with a unity matrix as $\mathbf{O}_r^{(n)} \in \mathbb{R}^{\prod_{k=1}^{N} J_k \times J_n}, r \in \{R_{core}\}$ and the unity matrix $\mathbf{U}^{(n)} \in \mathbb{R}^{J_n \times J_n}$, $n \in \{N\}$ which can present the variables $\mathbf{B}^{(n)}, n \in \{N\}$ in a gradient-explicit formation. The matrix $\mathbf{O}_r^{(n)}$ is defined as

$$\begin{aligned} \mathbf{O}_r^{(n)} = \Big[& q_{1,r} \mathbf{U}^{(n)}, \ldots, q_{m,r} \mathbf{U}^{(n)}, \ldots, \\ & q_{\prod_{k=1, k \neq n}^{N} J_k, r} \mathbf{U}^{(n)} \Big]^T. \end{aligned} \tag{7}$$

The key transformation of the gradient-explicit formation for the variables $\mathbf{B}^{(n)}, n \in \{N\}$ is as $Vec(\sum_{r=1}^{R_{core}} b_{:,r}^{(n)} q_{:,r}^{(n)^T}) = \sum_{r=1}^{R_{core}} \mathbf{O}_r^{(n)} b_{:,r}^{(n)}$. Then the optimization objective (6) is reformulated into:

$$\begin{aligned} &\arg\min_{\mathbf{B}^{(n)}, n \in \{N\}} f\left(\mathbf{B}^{(n)} \middle| x^{(n)}, \{\mathbf{A}^{(n)}\}, \{\mathbf{B}^{(n)}\}\right) \\ &\quad = \left\| x^{(n)} - \mathbf{H}^{(n)} \sum_{r=1}^{R_{core}} \mathbf{O}_r^{(n)} b_{:,r}^{(n)} \right\|_2^2 + \lambda_{\mathbf{B}} \left\| \mathbf{B} \right\|_2^2. \end{aligned} \tag{8}$$

The cyclic block optimization strategy [51] is adopted to update the variables $\{b_{:,r_{core}}^{(n)} | r_{core} \in \{R_{core}\}\}$ within a low-rank matrix $\mathbf{B}^{(n)}, n \in \{N\}$ and eventually the optimization objective is reformulated into

$$\begin{aligned} &\arg\min_{b_{:,r_{core}}^{(n)}, n \in \{N\}} f\left(b_{:,r_{core}}^{(n)} \middle| x^{(n)}, \{\mathbf{A}^{(n)}\}, \{\mathbf{B}^{(n)}\}\right) \\ &\quad = \left\| x_{r_{core}}^{(n)} - \mathbf{H}_{r_{core}}^{(n)} b_{:,r_{core}}^{(n)} \right\|_2^2 + \lambda_{\mathbf{B}} \left\| b_{:,r_{core}}^{(n)} \right\|_2^2, \end{aligned} \tag{9}$$

where $x_{r_{core}}^{(n)} = x^{(n)} - \mathbf{H}^{(n)} \sum_{r=1, r \neq r_{core}}^{R_{core}} \mathbf{O}_r^{(n)} b_{:,r}^{(n)}$ and $\mathbf{H}_{r_{core}}^{(n)} = \mathbf{H}^{(n)} \mathbf{O}_{r_{core}}^{(n)} \in \mathbb{R}^{\prod_{n=1}^{N} I_n \times J_n}$.

Theorem 1. *From the function form of (9), the optimization objective for $b_{:,r_{core}}^{(n)}$ is a u-convex and L-smooth function.*

Proof. The proof is divided into the following two parts: 1) The non-convex optimization problem is transformed into fixing factor matrices $\mathbf{A}^{(n)}, n \in \{N\}$ and updating core tensor part, and this transformation can keep promise that convex optimization for each part and appropriate accuracy [46], [47], [48], [50]. 2) The distance function $\left\| x_{r_{core}}^{(n)} - \mathbf{H}_{r_{core}}^{(n)} b_{:,r_{core}}^{(n)} \right\|_2^2$ is an euclidean distance with L_2 norm regularization $\lambda_{\mathbf{B}} \left\| b_{:,r_{core}}^{(n)} \right\|_2^2$ [52]. Thus, the optimization objective (9) is a u-convex and L-smooth function obviously [49], [50], [53]. The proof details are omitted which can be found on [32], [54]. □

4.2 Optimization Process for Factor Matrices

After the optimization step is conducted for $\{\mathbf{B}^{(n)} \in \mathbb{R}^{J_n \times R_{core}} | n \in \{N\}\}$, the Kruskal product for approximated core tensor $\hat{g}^{(n)}$ is constructed by Equ. (4). Thus, we should consider the optimization problem for factor matrices $\mathbf{A}^{(n)}, \{n | n \in \{N\}\}$ as

$$\begin{aligned} &\arg\min_{\mathbf{A}^{(n)}, n \in \{N\}} f\left(\mathbf{A}^{(n)} \middle| \boldsymbol{X}^{(n)}, \{\mathbf{A}^{(n)}\}, \widehat{\mathbf{G}}^{(n)}\right) \\ &\quad = \left\| \boldsymbol{X}^{(n)} - \widehat{\boldsymbol{X}}^{(n)} \right\|_2^2 + \lambda_{\mathbf{A}} \left\| \mathbf{A}^{(n)} \right\|_2^2, \end{aligned} \tag{10}$$

where $\widehat{\boldsymbol{X}}^{(n)} = \mathbf{A}^{(n)} \widehat{\mathbf{G}}^{(n)} \mathbf{S}^{(n)^T}$ and $\mathbf{S}^{(n)} = \mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \otimes \cdots \otimes \mathbf{A}^{(1)}$.

The optimization process for the whole factor matrix set $\{\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}\}$ is non-convex. The alternative optimization strategy [50] is adopted to transform the non-convex optimization problem into N convex optimization problems under a fine-tuned initialization as

Authorized licensed use limited to: TU Delft Library. Downloaded on February 21,2021 at 18:47:34 UTC from IEEE Xplore. Restrictions apply.

$$\begin{cases} \arg\min_{\mathbf{A}^{(1)}} f\left(\mathbf{A}^{(1)}\right) = \left\|\boldsymbol{X}^{(1)} - \mathbf{A}^{(1)}\mathbf{E}^{(1)}\right\|_2^2 + \lambda_{\mathbf{A}}\left\|\mathbf{A}^{(1)}\right\|_2^2; \\ \vdots \\ \arg\min_{\mathbf{A}^{(n)}} f\left(\mathbf{A}^{(n)}\right) = \left\|\boldsymbol{X}^{(n)} - \mathbf{A}^{(n)}\mathbf{E}^{(n)}\right\|_2^2 + \lambda_{\mathbf{A}}\left\|\mathbf{A}^{(n)}\right\|_2^2; \\ \vdots \\ \arg\min_{\mathbf{A}^{(N)}} f\left(\mathbf{A}^{(N)}\right) = \left\|\boldsymbol{X}^{(N)} - \mathbf{A}^{(N)}\mathbf{E}^{(N)}\right\|_2^2 + \lambda_{\mathbf{A}}\left\|\mathbf{A}^{(N)}\right\|_2^2, \end{cases} \tag{11}$$

where $\mathbf{E}^{(n)} = \widehat{\mathbf{G}}^{(n)}\mathbf{S}^{(n)^T} \in \mathbb{R}^{J_n \times \prod_{k=1,k\neq n}^{N} I_k}$. The optimization objectives of the nth variables are presented as an independent form as

$$\begin{cases} \arg\min_{a_{1,:}^{(n)}} f(a_{1,:}^{(n)}) = \left\|\mathbf{X}_{1,:}^{(n)} - a_{1,:}^{(n)}\mathbf{E}^{(n)}\right\|_2^2 + \lambda_{\mathbf{A}}\left\|a_{1,:}^{(n)}\right\|_2^2; \\ \vdots \\ \arg\min_{a_{i_n,:}^{(n)}} f(a_{i_n,:}^{(n)}) = \left\|\mathbf{X}_{i_n,:}^{(n)} - a_{i_n,:}^{(n)}\mathbf{E}^{(n)}\right\|_2^2 + \lambda_{\mathbf{A}}\left\|a_{i_n,:}^{(n)}\right\|_2^2; \\ \vdots \\ \arg\min_{a_{I_n,:}^{(n)}} f(a_{I_n,:}^{(n)}) = \left\|\mathbf{X}_{I_n,:}^{(n)} - a_{I_n,:}^{(n)}\mathbf{E}^{(n)}\right\|_2^2 + \lambda_{\mathbf{A}}\left\|a_{I_n,:}^{(n)}\right\|_2^2. \end{cases} \tag{12}$$

Theorem 2. *From the function form of (12), the optimization objective for $\mathbf{A}^{(n)}$ is a u-convex and L-smooth function.*

Proof. By adopting the alternative strategy [46], [47], [48], [50], we fix $\widehat{\mathcal{G}}$ and $\mathbf{A}^{(k)}, k \neq n, k \in \{N\}$. Then, we update $\mathbf{A}^{(n)}$. The distance function $\|\mathbf{X}_{i_n,:}^{(n)} - a_{i_n,:}^{(n)}\mathbf{E}^{(n)}\|_2^2$ is an euclidean distance with L_2 norm regularization $\lambda_{\mathbf{A}}\|a_{i_n,:}^{(n)}\|_2^2$, $i_n \in I_n$ [52]. Thus, the optimization objective (9) is a u-convex and L-smooth function obviously [49], [50], [53]. Due to the limited space, the proof details are omitted which can be found on [32], [54]. □

4.3 Stochastic Optimization

The previous Sections 4.1 and 4.2 presented the transformation of the optimization problem. In this section, the solvent is introduced. The average SGD method is a basic part of state of the art stochastic optimization models. Thus, in this section, we present the average SGD for our optimization objectives. The optimization objectives for the core tensor are presented in Equs. (8) and (9). The optimization objectives for the factor matrix are presented in Equ. (12) which are in the form of a basic optimization model. In the industrial and big-data communities, the HOHDST is very common. Thus, SGD is proposed to replace the original optimization strategy.

The solution for $b_{:,r_{core}}^{(n)}$ is presented as

$$\begin{aligned} &\arg\min_{b_{:,r_{core}}^{(n)}, n\in\{N\}} f\left(b_{:,r_{core}}^{(n)} \middle| x^{(n)}, \{\mathbf{A}^{(n)}\}, \{\mathbf{B}^{(n)}\}\right) \\ &= \sum_{i\in\Omega_V^{(n)}} L_i\left(b_{:,r_{core}}^{(n)} \middle| x^{(n)}\right) + \lambda_{\mathbf{B}}\left\|b_{:,r_{core}}^{(n)}\right\|_2^2. \end{aligned} \tag{13}$$

If a set Ψ including M randomly entries is selected, the approximated SGD solution for $b_{:,r_{core}}^{(n)}$ is presented as

$$\begin{aligned} &\arg\min_{b_{:,r_{core}}^{(n)}, n\in\{N\}} f_{\Psi_V^{(n)}}\left(b_{:,r_{core}}^{(n)} \middle| x_{\Psi_V^{(n)}}^{(n)}, \{\mathbf{A}^{(n)}\}, \{\mathbf{B}^{(n)}\}\right) \\ &= \sum_{i\in\Psi_V^{(n)}} L_i\left(b_{:,r_{core}}^{(n)} \middle| x^{(n)}\right) + \lambda_{\mathbf{B}}\left\|b_{:,r_{core}}^{(n)}\right\|_2^2 \\ &= \left\|x_{\Psi_V^{(n)}}^{(n)} - \mathbf{H}_{\Psi_V^{(n)},:}^{(n)} \sum_{r=1}^{R_{core}} \mathbf{O}_r^{(n)} b_{:,r}^{(n)}\right\|_2^2 + \lambda_{\mathbf{B}}\left\|b_{:,r_{core}}^{(n)}\right\|_2^2 \\ &= \left\|(x_{r_{core}}^{(n)})_{\Psi_V^{(n)}} - \mathbf{H}_{\Psi_V^{(n)},:}^{(n)} \mathbf{O}_{r_{core}}^{(n)} b_{:,r_{core}}^{(n)}\right\|_2^2 + \lambda_{\mathbf{B}}\left\|b_{:,r_{core}}^{(n)}\right\|_2^2. \end{aligned} \tag{14}$$

The SGD for the approximated function $f_{\Psi_V^{(n)}}(b_{:,r_{core}}^{(n)}|x_{\Psi_V^{(n)}}^{(n)}, \{\mathbf{A}^{(n)}\}, \{\mathbf{B}^{(n)}\})$ is deduced as

$$\begin{aligned} &\frac{\partial f_{\Psi_V^{(n)}}\left(b_{:,r_{core}}^{(n)} \middle| x_{\Psi_V^{(n)}}^{(n)}, \{\mathbf{A}^{(n)}\}, \{\mathbf{B}^{(n)}\}\right)}{\partial b_{:,r_{core}}^{(n)}} \\ &= \frac{1}{M}\Big(- \mathbf{O}_{r_{core}}^{(n)T}\mathbf{H}_{\Psi_V^{(n)},:}^{(n)T}(x_{r_{core}}^{(n)})_{\Psi_V^{(n)}} \\ &\quad + \mathbf{O}_{r_{core}}^{(n)T}\mathbf{H}_{\Psi_V^{(n)},:}^{(n)T}\mathbf{H}_{\Psi_V^{(n)},:}^{(n)}\mathbf{O}_{r_{core}}^{(n)} b_{:,r_{core}}^{(n)}\Big) + \lambda_{\mathbf{B}} b_{:,r_{core}}^{(n)}. \end{aligned} \tag{15}$$

The solution for factor matrices $a_{i_n,:}^{(n)}$, $i_n \in \{I_N\}$, $n \in \{N\}$ is presented as

$$\begin{aligned} &\arg\min_{a_{i_n,:}^{(n)}, n\in\{N\}} f\left(a_{i_n,:}^{(n)} \middle| \boldsymbol{X}_{i_n,:}^{(n)}, \{\mathbf{A}^{(n)}\}, \widehat{\mathbf{G}}^{(n)}\right) \\ &\qquad = \sum_{j\in(\Omega_M^{(n)})_{i_n}} L_j\left(a_{i_n,:}^{(n)} \middle| \boldsymbol{X}_{i_n,j}^{(n)}\right) + \lambda_{\mathbf{A}}\left\|a_{i_n,:}^{(n)}\right\|_2^2. \end{aligned} \tag{16}$$

If a set Ψ including M randomly chosen entries is selected, the SGD solution for $a_{i_n,:}^{(n)}$ can be expressed as

$$\begin{aligned} &\arg\min_{a_{i_n,:}^{(n)}, n\in\{N\}} f_{\Psi_M^{(n)}}\left(a_{i_n,:}^{(n)} \middle| \boldsymbol{X}_{i_n,(\Psi_M^{(n)})_{i_n}}^{(n)}, \{\mathbf{A}^{(n)}\}, \widehat{\mathbf{G}}^{(n)}\right) \\ &= \sum_{j\in(\Psi_M^{(n)})_{i_n}} L_j\left(a_{i_n,:}^{(n)} \middle| \boldsymbol{X}_{i_n,j}^{(n)}\right) + \lambda_{\mathbf{A}}\left\|a_{i_n,:}^{(n)}\right\|_2^2 \\ &= \left\|\mathbf{X}_{i_n,(\Psi_M^{(n)})_{i_n}}^{(n)} - a_{i_n,:}^{(n)}\mathbf{E}_{:,(\Psi_M^{(n)})_{i_n}}^{(n)}\right\|_2^2 + \lambda_{\mathbf{A}}\left\|a_{i_n,:}^{(n)}\right\|_2^2. \end{aligned} \tag{17}$$

The stochastic gradient for the approximated function $f_{\Psi_V^{(n)}}(a_{i_n,:}^{(n)}|\boldsymbol{X}_{i_n,(\Psi_M^{(n)})_{i_n}}^{(n)}, \{\mathbf{A}^{(n)}\}, \mathbf{G}^{(n)})$ is deduced as

$$\begin{aligned} &\frac{\partial f_{\Psi_V^{(n)}}\left(a_{i_n,:}^{(n)} \middle| \boldsymbol{X}_{i_n,(\Psi_M^{(n)})_{i_n}}^{(n)}, \{\mathbf{A}^{(n)}\}, \widehat{\mathbf{G}}^{(n)}\right)}{\partial a_{i_n,:}^{(n)}} \\ &= \frac{1}{M}\Big(- \mathbf{X}_{i_n,(\Psi_M^{(n)})_{i_n}}^{(n)}\mathbf{E}_{:,(\Psi_M^{(n)})_{i_n}}^{(n)T} \\ &\quad + a_{i_n,:}^{(n)}\mathbf{E}_{:,(\Psi_M^{(n)})_{i_n}}^{(n)}\mathbf{E}_{:,(\Psi_M^{(n)})_{i_n}}^{(n)T}\Big) + \lambda_{\mathbf{A}} a_{i_n,:}^{(n)}. \end{aligned} \tag{18}$$

Authorized licensed use limited to: TU Delft Library. Downloaded on February 21,2021 at 18:47:34 UTC from IEEE Xplore. Restrictions apply.

The computational details are presented in Algorithm 1, which shows that SGD_Tucker is able to divide the high-dimension intermediate matrices $\{\mathbf{H}^{(n)}_{\Omega^{(n)}_V,:}, \mathbf{S}^{(n)}_{(\Omega^{(n)}_M)_{i_n},:}, \mathbf{E}^{(n)}_{:,(\Omega^{(n)}_M)_{i_n}} | i_n \in \{I_n\}, n \in \{N\}\}$ into small batches of intermediate matrices $\{\mathbf{H}^{(n)}_{\Psi^{(n)}_V,:}, \mathbf{S}^{(n)}_{(\Psi^{(n)}_M)_{i_n},:}, \mathbf{E}^{(n)}_{:,(\Psi^{(n)}_M)_{i_n}} | i_n \in \{I_n\}, n \in \{N\}\}$. We summarized all steps of SGD_Tucker in Algorithm 1.

Algorithm 1. Stochastic Optimization Strategies on a Training Epoch for SGD_Tucker

Input: Sparse tensor $\mathcal{X}$, randomly selected set Ψ with M entries, Learning step $\gamma_{\mathbf{A}}$, learning step $\gamma_{\mathbf{B}}$;
Initializing $\mathbf{B}^{(n)}, n \in \{N\}$, core tensor $\widehat{\mathcal{G}}$, $\mathbf{A}^{(n)}, n \in \{N\}$, $\mathbf{H}^{(n)}_{\Psi^{(n)}_V,:} \in \mathbb{R}^{M \times \prod_{n=1}^{N} J_n}$, $r \in \{R_{core}\}$, $n \in \{N\}$, $\mathbf{O}^{(n)}_r \in \mathbb{R}^{\prod_{k=1}^{N} J_k \times J_n}$, $r \in \{R_{core}\}, n \in \{N\}$, $\widehat{x}^{(n)}_{\Psi^{(n)}_V} \in \mathbb{R}^M$.
Output: $\mathbf{B}^{(n)}, n \in \{N\}$, $\widehat{\mathcal{G}}$, $\mathbf{A}^{(n)}, n \in \{N\}$.

1: **for** n from 1 to N **do**
2: Obtain $\mathbf{H}^{(n)}_{\Psi^{(n)}_V,:}$;
3: **for** r_{core} from 1 to R_{core} **do**
4: Obtain $\mathbf{O}^{(n)}_{r_{core}}$ by Equ. (7);
5: Obtain $\mathbf{W}^{(n)}_{r_{core}} \leftarrow \mathbf{H}^{(n)}_{\Psi^{(n)}_V,:} \mathbf{O}^{(n)}_{r_{core}}$;
6: **end for**
7: **for** r_{core} from 1 to R_{core} **do**
8: $\widehat{x}^{(n)}_{\Psi^{(n)}_V} \leftarrow x^{(n)}_{\Psi^{(n)}_V}$;
9: **for** r from 1 to R_{core} $(r \neq r_{core})$ **do**
10: $\widehat{x}^{(n)}_{\Psi^{(n)}_V} \leftarrow \widehat{x}^{(n)}_{\Psi^{(n)}_V} - \mathbf{W}^{(n)}_r b^{(n)}_{:,r}$;
11: **end for**
12: $\mathbf{C}_{r_{core}} \leftarrow \mathbf{W}^{(n)^T}_{r_{core}} \mathbf{W}^{(n)}_{r_{core}}$;
13: $\mathbf{V}^{(n)}_{r_{core}} \leftarrow -\mathbf{W}^{(n)^T}_{r_{core}} \widehat{x}^{(n)}_{\Psi^{(n)}_V} + \mathbf{C}_{r_{core}} b^{(n)}_{:,r_{core}}$;
14: $b^{(n)}_{:,r_{core}} \leftarrow SGD(M, \lambda_{\mathbf{B}}, \gamma_{\mathbf{B}}, b^{(n)}_{:,r_{core}}, \mathbf{V}^{(n)}_{r_{core}})$;
15: **end for**
16: **end for**
17: Obtain $\widehat{\mathcal{G}}$ by Equ. (4);
18: **for** n from 1 to N **do**
19: **for** i_n from 1 to I_n **do**
20: Obtain $\mathbf{S}^{(n)}_{(\Psi^{(n)}_M)_{i_n},:}(cache_S)$;
21: $\mathbf{E}^{(n)}_{:,(\Psi^{(n)}_M)_{i_n}} \leftarrow \widehat{\mathbf{G}}^{(n)} \mathbf{S}^{(n)^T}_{(\Psi^{(n)}_M)_{i_n},:}(cache_E)$;
22: $\mathbf{C}^{(n)} \leftarrow \mathbf{E}^{(n)}_{:,(\Psi^{(n)}_M)_{i_n}} \mathbf{E}^{(n)T}_{:,(\Psi^{(n)}_M)_{i_n}}$;
23: $\mathbf{F}^{(n)} \leftarrow \underbrace{-\mathbf{X}^{(n)}_{i_n,(\Psi^{(n)}_M)_{i_n}} \mathbf{E}^{(n)T}_{:,(\Psi^{(n)}_M)_{i_n}}}_{cache_{Fact1}} + \overbrace{\underbrace{a^{(n)}_{i_n,:}\mathbf{C}^{(n)}}_{cache_{Fact2}}}^{cache_{Factp}, cache_{Factvec}}$;
24: $a^{(n)}_{i_n,:} \leftarrow SGD(|(\Psi^{(n)}_M)_{i_n}|, \lambda_{\mathbf{A}}, \gamma_{\mathbf{A}}, a^{(n)}_{i_n,:}, \mathbf{F}^{(n)})$.
25: **end for**
26: **end for**
27: **Return**: $\mathbf{B}^{(n)}, n \in \{N\}$, $\widehat{\mathcal{G}}$, $\mathbf{A}^{(n)}, n \in \{N\}$.

Theorem 3. *From the function forms of (9) and (12), the optimization objectives for the core tensor and factor matrices are both u-convex and L-smooth functions. The stochastic update rules of (15) and (18) can ensure the convergency of alternative optimization objectives (9) and (12), respectively.*

Proof. The two function forms of (9) and (12) can be conclude as $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x})$, where $f(\mathbf{x})$ is a strongly-convex with constant μ, and each $f_i(\mathbf{x})$ is smooth and Lipschitz-continuous with constant L. At tth iteration, for chosen f_{i_t} randomly, and a learning rate sequence $\gamma_t > 0$, the expectation $\mathbb{E}[\nabla f_i(\mathbf{x}_t)|\mathbf{x}_t]$ of $\nabla f_{i_t}(\mathbf{x}_t)$ is equivalent to $\nabla f(\mathbf{x}_t)$ [49], [53], [54], [55]. □

4.4 Parallel and Distributed Strategy

We first explain the naive parallel strategy (Section 4.4.1), which relies on the coordinate continuity and the specific style of matricization unfolding of the sparse tensor to keep the entire continuous accessing in the process of updating $\mathbf{A}^{(n)}, n \in \{N\}$. Then, we present the improved parallel strategy (Section 4.4.2), which can save the N compressive styles of the sparse tensor $\mathcal{X}$ to just a compressive format. At last, the analysis of the communication cost is reported on Section 4.4.3.

4.4.1 Naive Parallel Strategy

To cater to the need of processing big data, the algorithm design shall leverage the increasing number of cores while minimizing the communication overehead among parallel threads. If there are L threads, the randomly selected set Ψ is divided into L subsets $\{\Psi_1, \ldots, \Psi_l, \ldots, \Psi_L | l \in \{L\}\}$ and the parallel computation analysis follows each step in Algorithm 1 as:

(i) Updating the core tensor:

Line 2: The computational tasks of the L Intermediate matrices $\{\mathbf{H}^{(n)}_{\Psi^{(n)}_{l_V},:} \in \mathbb{R}^{|\Psi_l| \times \prod_{n=1}^{N} J_n} | l \in \{L\}, n \in \{N\}\}$ be allocated to L threads;

Line 4: Intermediate matrix $\mathbf{O}^{(n)}_r \in \mathbb{R}^{\prod_{k=1}^{N} J_k \times J_n}$, $r \in \{R_{core}\}$, $n \in \{N\}$; thus, the independent computational tasks of the $\prod_{k=1,k\neq n}^{N} J_k$ diagonal sub-matrices can be allocated to the L threads;

Line 5: The computational tasks of the L intermediate matrices $\{\mathbf{H}^{(n)}_{\Psi^{(n)}_{l_V},:} \mathbf{O}^{(n)}_r \in \mathbb{R}^{|\Psi_l| \times J_n} | l \in \{L\}, n \in \{N\}\}$ can be allocated to L threads;

Line 8: The L assignment tasks $\{\widehat{x}^{(n)}_{\Psi^{(n)}_{l_V}} | l \in \{L\}\}$ can be allocated to the L threads;

Line 10: The computational tasks of the L intermediate matrices $\{(\mathbf{W}^{(n)}_r)_{\Psi^{(n)}_{l_V},:} b^{(n)}_{:,r} \in \mathbb{R}^{|\Psi_l|} | r \in \{R_{core}\}, n \in \{N\}\}$ can be allocated to L threads;

Line 12: The computational tasks of the L intermediate matrices $\{\mathbf{C}^l_{r_{core}} = (\mathbf{W}^{(n)}_r)^T_{\Psi^{(n)}_{l_V},:} (\mathbf{W}^{(n)}_r)_{\Psi^{(n)}_{l_V},:} \in \mathbb{R}^{J_n \times J_n} | l \in \{L\}, r \in \{R_{core}\}, n \in \{N\}\}$ can be allocated to L threads and the l thread does the intra-thread summation $\mathbf{C}^l_{r_{core}} = (\mathbf{W}^{(n)}_r)^T_{\Psi^{(n)}_{l_V},:} (\mathbf{W}^{(n)}_r)_{\Psi^{(n)}_{l_V},:}$. Then, the main thread sums $\mathbf{C}_{r_{core}} = \sum_{l=1}^{L} \mathbf{C}^l_{r_{core}}$;

Line 13: The computational tasks of the L intermediate matrices $\{(\mathbf{W}^{(n)}_r)^T_{\Psi^{(n)}_{l_V},:} \widehat{x}^{(n)}_{\Psi^{(n)}_{l_V}} \in \mathbb{R}^{J_n}, \mathbf{C}_r b^{(n)}_{:,r} \in \mathbb{R}^{J_n} | r \in \{R_{core}\}, n \in \{N\}\}$ can be allocated to L threads.

Line 14: This step can be processed by the main thread.

(ii) Updating factor matrices: I_n loops including the *Lines* $20 - 24$, $n \in \{N\}$ are independent. Thus, the I_n, $n \in \{N\}$ loops can be allocated to L threads.

The parallel training process for $\mathbf{B}^{(n)}, n \in \{N\}$ does not need the load balance. The computational complexity of $\mathbf{S}^{(n)}_{(\Psi^{(n)}_M)_{i_n},:}$ for each thread is proportional to $|(\Psi^{(n)}_M)_{i_n}|$, and the I_n, $n \in \{N\}$

Authorized licensed use limited to: TU Delft Library. Downloaded on February 21,2021 at 18:47:34 UTC from IEEE Xplore. Restrictions apply.

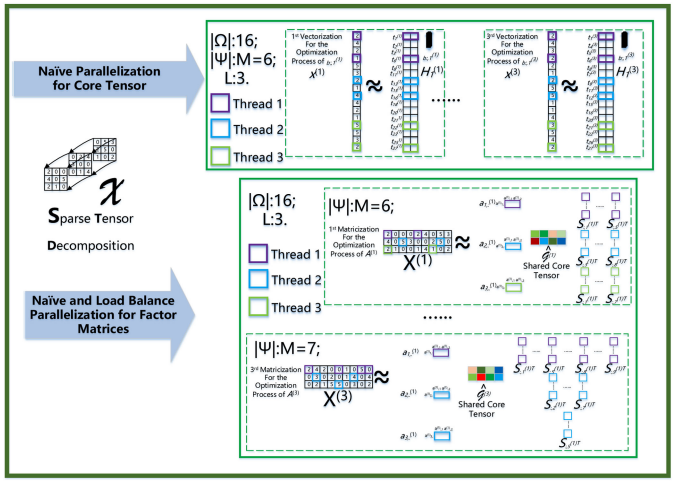


Fig. 3. Naive parallel strategy for SGD_Tucker.

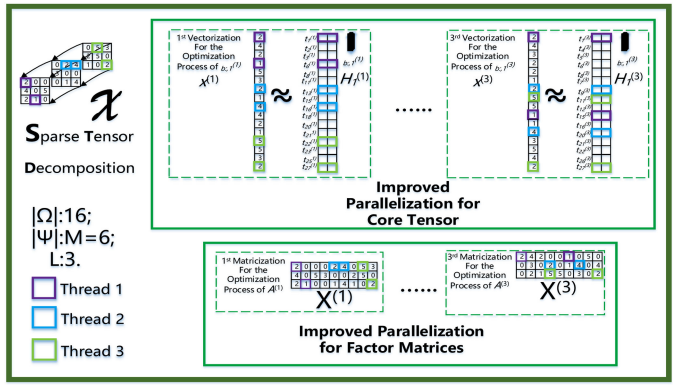


Fig. 4. Improved parallel strategy for SGD_Tucker.

independent tasks are allocated to the L threads. Then, there is load imbalance problem for updating the factor matrices $\mathbf{A}^{(n)}$, $n \in \{N\}$. Load balance can fix the problem of non-even distribution for non-zero elements.

A toy example is shown in Fig. 3 which comprises of:

(i) Each thread l first selects the even number of non-zero elements: The 3 threads $\{1, 2, 3\}$ select $\{\{\, \boldsymbol{x}_1^{(1)}, \boldsymbol{x}_6^{(1)} \,\}, \{\, \boldsymbol{x}_{13}^{(1)}, \boldsymbol{x}_{16}^{(1)} \,\}, \{\, \boldsymbol{x}_{22}^{(1)}, \boldsymbol{x}_{27}^{(1)} \,\}\}$, respectively. *Step 1*: The 3 threads $\{1, 2, 3\}$ construct $\{\boldsymbol{H}_{1,:}^{(1)}, \boldsymbol{H}_{13,:}^{(1)}, \boldsymbol{H}_{22,:}^{(1)}\}$, respectively. Then, the 3 threads $\{1, 2, 3\}$ compute $\{\mathbf{W}_1^{(1)} = \boldsymbol{H}_{1,:}^{(1)}\mathbf{O}^{(1)}, \mathbf{W}_{13}^{(1)} = \boldsymbol{H}_{13,:}^{(1)}\mathbf{O}^{(1)}, \mathbf{W}_{22}^{(1)} = \boldsymbol{H}_{22,:}^{(1)}\mathbf{O}^{(1)}\}$, respectively. *Step 2*: The 3 threads $\{1, 2, 3\}$ construct $\{\boldsymbol{H}_{6,:}^{(1)}, \boldsymbol{H}_{16,:}^{(1)}, \boldsymbol{H}_{27,:}^{(1)}\}$, respectively. Then, the 3 threads $\{1, 2, 3\}$ compute $\{\mathbf{W}_6^{(1)} = \boldsymbol{H}_{6,:}^{(1)}\mathbf{O}^{(1)}, \mathbf{W}_{16}^{(1)} = \boldsymbol{H}_{16,:}^{(1)}\mathbf{O}^{(1)}, \mathbf{W}_{27}^{(1)} = \boldsymbol{H}_{27,:}^{(1)}\mathbf{O}^{(1)}\}$, respectively. Each thread does the summation within the thread and the 3 threads do the entire summation by the code $\#\ pragma\ omp\ parallel\ for\ reduction\ (+ : sum)$ for multi-thread summation. We observe that the computational process of $\mathbf{W}_k^{(1)^T}\mathbf{W}_k^{(1)}b_{1,:}^{(1)}$, $k \in \{1, 6, 13, 16, 22, 27\}$ is divided into the vectors reduction operation $p = \mathbf{W}_k^{(1)}b_{1,:}^{(1)}$ and $vec = \mathbf{W}_k^{(1)^T}p$. *Step 3*: The $b_{:,1}^{(1)}$ is updated. The process of updating $b_{:,1}^{(3)}$ is similar the process of updating $b_{:,1}^{(1)}$. The description is omitted. We observe that each thread selects 2 elements. Thus, the load for the 3 threads is balanced.

(ii) Each thread selects the independent rows and L threads realize naive parallelization for $\mathbf{A}^{(n)}, n \in \{N\}$ by the nth matricization $\mathbf{X}^{(n)}$, $n \in \{N\}$: As show in Fig. 3, the 3 threads $\{1, 2, 3\}$ update $\{a_{1,:}^{(1)}, a_{2,:}^{(1)}, a_{3,:}^{(1)}\}$ explicitly. Thus, the description is omitted. The 2 threads $\{1, 2\}$ update $\{a_{1,:}^{(1)}, \{a_{2,:}^{(1)}, a_{3,:}^{(1)}\}\}$, respectively and independently by $\{\{\boldsymbol{X}_{1,1}^{(3)}, \boldsymbol{S}_{:,1}^{(3)}\}, \{\boldsymbol{X}_{1,5}^{(3)}, \boldsymbol{S}_{:,5}^{(3)}\}, \{\boldsymbol{X}_{1,8}^{(3)}, \boldsymbol{S}_{:,8}^{(3)}, \{\boldsymbol{X}_{1,9}^{(3)}, \boldsymbol{S}_{:,9}^{(3)}\}\}, \{\{\boldsymbol{X}_{2,2}^{(3)}, \boldsymbol{S}_{:,2}^{(3)}\}, \{\boldsymbol{X}_{2,7}^{(3)}, \boldsymbol{S}_{:,7}^{(3)}\}\}, \{\{\boldsymbol{X}_{3,5}^{(3)}, \boldsymbol{S}_{:,5}^{(3)}\}\}\}$, respectively, with the shared matrix $\widehat{\mathbf{G}}^{(3)}$. Thread 1 selects 4 elements for updating $a_{1,:}^{(3)}$. Thread 2 selects 2 elements and 1 element for updating $a_{2,:}^{(3)}$ and $a_{3,:}^{(3)}$, respectively, which can dynamically balance the load. In this condition, the load for the 3 threads is slightly more balanced.

As shown in Fig. 3, the naive parallel strategy for $\{\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}\}$ relies on the matricization format $\{X^{(1)}, \ldots, X^{(N)}\}$ of a sparse tensor $\mathcal{X}$, respectively, which is used to avoid the read-after-write and write-after-read conflicts. Meanwhile, the transformation process for the compressive format to another one is time consuming. Thus, the space and computational overheads are not scalable.

4.4.2 *Improved Parallel Strategy*

The improved parallel strategy is developed to use only one compressive format for the entire updating process and hence save the memory requirements. At the same time, it can avoid the read-after-write or write-after-read conflicts. In Fig. 4, a toy example of improved parallel strategy is illustrated. As show in Fig. 4, the 3 threads $\{1, 2, 3\}$ select 6 non-zeros elements and the Coordinate Format (COO) is $\{\{$ (1,1,1,2.0), (3,2,1,1.0) $\}$, $\{$ (1,2,2,2.0), (1,3,2,4.0) $\}$, $\{$ (1,2,3,5.0), (3,3,3,2.0) $\}\}$, respectively. By the structure of COO, the training process of $B^{(1)}$ and $B^{(3)}$ does not need a specific shuffling order of a sparse tensor. Thus, the description of updating $b_{:,1}^{(1)}$ and $b_{:,1}^{(3)}$ is omitted.

As for $X^{(3)}$ to update $\mathbf{A}^{(3)}$, we neglect this condition because the selected elements of 3 threads lie in independent rows and it is trivial to parallelize. In the style of $X^{(1)}$ for updating $\mathbf{A}^{(1)}$, updating $a_{1,:}^{(1)}$ relies on $\{(1, 1, 1, 2.0), \mathbf{S}_{:,1}^{(1)}\}$ (selected by thread 1), $\{\{(1, 2, 2, 2.0), \mathbf{S}_{:,5}^{(1)}\}, \{(1, 3, 2, 4.0), \mathbf{S}_{:,6}^{(1)}\}\}$ (selected by thread 2), and $\{(1, 2, 3, 5.0), \mathbf{S}_{:,8}^{(1)}\}$ (selected by thread 3) with the shared matrix $\widehat{\mathbf{G}}^{(1)}$. It has following three steps.

(i) (Lines 20-21 in Algorithm 1) The 3 threads $\{1, 2, 3\}$ compute $\{\mathbf{E}_{:,1}^{(1)} = \widehat{\mathbf{G}}^{(1)}\mathbf{S}_{:,1}^{(1)^T}, \{\mathbf{E}_{:,5}^{(1)} = \widehat{\mathbf{G}}^{(1)}\mathbf{S}_{:,5}^{(1)^T}, \mathbf{E}_{:,6}^{(1)} = \widehat{\mathbf{G}}^{(1)}\mathbf{S}_{:,6}^{(1)^T}\}, \mathbf{E}_{:,8}^{(1)} = \widehat{\mathbf{G}}^{(1)}\mathbf{S}_{:,8}^{(1)^T}\}$ independently, by the *private cache matrix* $\{cache_S, cache_E\}$ of each thread;

(ii) (Lines 22-23 in Algorithm 1) The computational process of $a_{i_n,:}^{(n)}\mathbf{E}_{:,k}^{(1)}\mathbf{E}_{:,k}^{(1)^T}$, $k \in \{1, 5, 6, 8\}$ is divided into the vectors reduction operation $cache_{Factp} = a_{i_n,:}^{(n)}\mathbf{E}_{:,k}^{(1)}$ and $cache_{Factvec} = cache_{Factp}\mathbf{E}_{:,k}^{(1)^T}$. The 3 threads $\{1, 2, 3\}$ compute $\{a_{i_n,:}^{(n)}\mathbf{E}_{:,1}^{(1)}\mathbf{E}_{:,1}^{(1)^T}, \{a_{i_n,:}^{(n)}\mathbf{E}_{:,5}^{(1)}\mathbf{E}_{:,5}^{(1)^T}, a_{i_n,:}^{(n)}\mathbf{E}_{:,6}^{(1)}\mathbf{E}_{:,6}^{(1)^T}\}, a_{i_n,:}^{(n)}\mathbf{E}_{:,8}^{(1)}\mathbf{E}_{:,8}^{(1)^T}\}$, respectively and independently, by the *private cache* $cache_{Factp}$ and $cache_{Factvec}$ of each thread. Then, the 3 threads $\{1, 2, 3\}$ can use the synchronization mechanisms, i.e., *atomic*, *cirtical* or *mutex*, of OpenMP to accomplish $\prod_{k=1,5,6,8} a_{i_n,:}^{(n)}\mathbf{E}_{:,k}^{(1)}\mathbf{E}_{:,k}^{(1)^T}$. Then, the results are returned to *global shared cache* $cache_{Fact2}$;

Authorized licensed use limited to: TU Delft Library. Downloaded on February 21,2021 at 18:47:34 UTC from IEEE Xplore. Restrictions apply.

(iii) (Line 23 in Algorithm 1). The 3 threads $\{1,2,3\}$ compute $\{x_{1,1,1}\mathbf{E}^{(1)}_{:,1}, \{x_{1,2,2}\mathbf{E}^{(1)}_{:,5}, x_{1,3,2}\mathbf{E}^{(1)}_{:,6}\}, x_{1,2,3}\mathbf{E}^{(1)}_{:,8}\}$, respectively and independently. Then, the 3 threads $\{1,2,3\}$ can use the synchronization mechanisms, i.e., *atomic*, *cirtical* or *mutex*, to accomplish $\mathbf{F}^{(1)}_1$. Then, the results are returned to the *global shared cache* $cache_{Fact1}$. Eventually, the I_1 tasks $SGD(6, \lambda_{\mathbf{A}}, \gamma_{\mathbf{A}}, a^{(1)}_{1,:}, \mathbf{F}^{(1)}_1)$ be allocated to the the 3 threads $\{1,2,3\}$ in a parallel and *load balance* way. Due to the same condition of updating $a^{(1)}_{3,:}$ and limited space, the description of updating $a^{(1)}_{3,:}$ is omitted.

By the *global shared caches* and *private caches*, *SGD_Tucker* can handle the parallelization on OpenMP by just a compressive format and the space overhead is much less than the compressive data structure of a sparse tensor $\mathcal{X}$; meanwhile, this strategy does not increase extra computational overhead.

4.4.3 Communication in Distributed Platform

In distributed platform, the communication overhead for a core tensor is $O(\prod_{n=1}^{N} J_n)$, which is non-scalable in HOHDST scenarios. SGD_Tucker can prune the number of the parameters for constructing an updated core tensor from $O(\prod_{n=1}^{N} J_n)$ to $O(\sum_{n=1}^{N} J_n R_{core})$ where $R_{core} \ll J_n, n \in \{N\}$ while maintaining the same overall accuracy and lower computational complexity. Hence, nodes only need to communicate the Kruckal product matrices $\{\mathbf{B}^{(n)}_r \in \mathbb{R}^{J_n \times R_{core}} | r \in \{R_{core}\}, n \in \{N\}\}$ rather than the entire core tensor $\mathcal{G} \in \mathbb{R}_+^{J_1 \times J_2 \times \cdots \times J_N}$. Hence, SGD_Tucker features that 1) a stochastic optimization strategy is adopted to core tensor and factor matrices which can keep the low computation overhead and storage space while not compromising the accuracy; 2) each minor node can select sub-samples from allocated subtensor and then compute the partial gradient and the major node gathers the partial gradient from all minor nodes and then obtains and returns the full gradient; 3) the communication overhead for information exchange of the core tensor $\mathcal{G}$ is $O(\sum_{n=1}^{N} J_n R_{core})$.

4.5 Complexity Analysis

The space and complexity analyses follow the steps which are presented in Algorithm 1 as

Theorem 4. *The space overhead for updating* $\mathbf{B}^{(n)}$, $n \in \{N\}$ *is* $O\big((MN + R_{core}NJ_n)\prod_{k=1}^{N} J_k + R_{core}MJ_n + MN + J_n\big)$.

Theorem 5. *The computational complexity for updating* $\mathbf{B}^{(n)}$, $n \in \{N\}$ *is* $O((MN + R_{core}NJ_n)\prod_{k=1}^{N} J_k + R_{core}MJ_n + R_{core}(R_{core} - 1)MN + R_{core}NJ_n)$.

Proof. In the process updating $\mathbf{B}^{(n)}$, $n \in \{N\}$, the space overhead and computational complexity of intermediate matrices $\{\mathbf{H}^{(n)}_{\Psi^{(n)}_V,:}, \mathbf{O}^{(n)}_{r_{core}}, \mathbf{W}^{(n)}_{r_{core}}, \widehat{x}^{(n)}_{\Psi^{(n)}_V}, \mathbf{V}^{(n)}_{r_{core}} | n \in \{N\}, r_{core} \in \{R_{core}\}\}$ are $\{M\prod_{k=1}^{N} J_k, J_n\prod_{k=1}^{N} J_k, MJ_n, M, J_n | n \in \{N\}, r_{core} \in \{R_{core}\}\}$ and $\{M\prod_{k=1}^{N} J_k, \prod_{k=1}^{N} J_k, MJ_n\prod_{k=1}^{N} J_k, (R_{core} - 1)MJ_n, MJ_n + MJ_n^2 | n \in \{N\}, r_{core} \in \{R_{core}\}\}$, respectively. □

Theorem 6. *The space overhead for updating* $\mathbf{A}^{(n)}$, $n \in \{N\}$ *is* $O((max(|(\Psi^{(n)}_M)_{i_n}|) + 1)\prod_{k=1}^{N} J_k + max(|(\Psi^{(n)}_M)_{i_n}|)J_n + J_n)$.

Theorem 7. *The computational complexity for updating* $\mathbf{A}^{(n)}$, $n \in \{N\}$ *is* $O((N + MN + M)\prod_{n=1}^{N} J_n + \sum_{n=1}^{N} I_n J_n)$.

Proof. In the process updating $\mathbf{A}^{(n)}$, $n \in \{N\}$, the space overhead and computational complexity of intermediate matrices $\{\widehat{\mathbf{G}}^{(n)}, \mathbf{S}^{(n)}_{(\Psi^{(n)}_M)_{i_n},:}, \mathbf{E}^{(n)}_{:,(\Psi^{(n)}_M)_{i_n}}, \mathbf{F}^{(n)} | i_n \in \{I_n\}, n \in \{N\}\}$ are $\{\prod_{k=1}^{N} J_k, max(|(\Psi^{(n)}_M)_{i_n}|)\prod_{k=1}^{N} J_k, max(|(\Psi^{(n)}_M)_{i_n}|)J_n, \ J_n | i_n \in \{I_n\}, n \in \{N\}\}$ and $\{\prod_{k=1}^{N} J_k, |(\Psi^{(n)}_M)_{i_n}|\prod_{k=1}^{N} J_k, |(\Psi^{(n)}_M)_{i_n}| J_n, J_n | i_n \in \{I_n\}, n \in \{N\}\}$, respectively. □

5 Evaluation

The performance demonstration of SGD_Tucker comprises of two parts: 1) SGD can split the high-dimension intermediate variables into small batches of intermediate matrices and the scalability and accuracy are presented; 2) SGD_Tucker can be parallelized in a straightfoward manner and the speedup results are illustrated. The experimental settings are presented in Section 5.1. Sections 5.2 and 5.3 show the scalability and the influence of parameters, respectively. At last, Section 5.4 presents the speedup performance of SGD_Tucker and comparison with the state of the art algorithms for STD, i.e., P−Tucker [46], CD [47], and HOOI [41]. Due to the limited space, the full experimental details for HOOI and 2 small datasets, i.e., Movielen-100K and Movielen-1M, are presented in the supplementary material, available online.

5.1 Experimental Settings

The CPU server is equipped with 8 Intel(R) Xeon(R) E5-2620 v4 CPUs and each core has 2 hyperthreads, running on 2.10 GHz, for the state of the art algorithms for STD, e.g., P−Tucker [46], CD [47] and HOOI [41]. The experiments are conducted 3 public datasets : Netflix,[1] Movielens,[2] and Yahoo-music.[3] The datasets which be used in our experiments can be downloaded in this link. https://drive.google.com/drive/folders/1tcnAsUSC9jFty7AfWetb5R7nevN2WUxH?usp=sharing

For Movielens, data cleaning is conducted and 0 values are changed from zero to 0.5 and we make compression for Yahoo-music dataset in which the maximum value 100 be compressed to 5.0. The specific situation of datasets is shown in Table 3. In this way, the ratings of the two data sets are in the same interval, which facilitates the selection of parameters. Gaussian distribution $N(0.5, 0.1^2)$ is adopted for initialization; meanwhile, the regularization parameters $\{\lambda_{\mathbf{A}}, \lambda_{\mathbf{B}}\}$ are set as 0.01 and the learning rate $\{\gamma_{\mathbf{A}}, \gamma_{\mathbf{B}}\}$ are set as $\{0.002, 0.001\}$, respectively. For simplification, we set $M = 1$. The accuracy is measured by $RMSE$ and MAE as

$$RMSE = \sqrt{\Bigg(\sum_{(i_1,\ldots,i_N)\in\Gamma} (x_{i_1,\ldots,i_N} - \widehat{x}_{i_1,\ldots,i_N})^2\Bigg) \Big/ |\Gamma|}; \qquad MAE = \Bigg(\sum_{(i_1,\ldots,i_N)\in\Gamma} \Big| x_{i_1,\ldots,i_N} - \widehat{x}_{i_1,\ldots,i_N} \Big|\Bigg) \Big/ |\Gamma|, \tag{19}$$

respectively, where Γ denotes the test sets. All the experiments are conducted on double-precision float numeric.

1. https://www.netflixprize.com/
2. https://grouplens.org/datasets/movielens/
3. https://webscope.sandbox.yahoo.com/catalog.php?datatype

Authorized licensed use limited to: TU Delft Library. Downloaded on February 21,2021 at 18:47:34 UTC from IEEE Xplore. Restrictions apply.

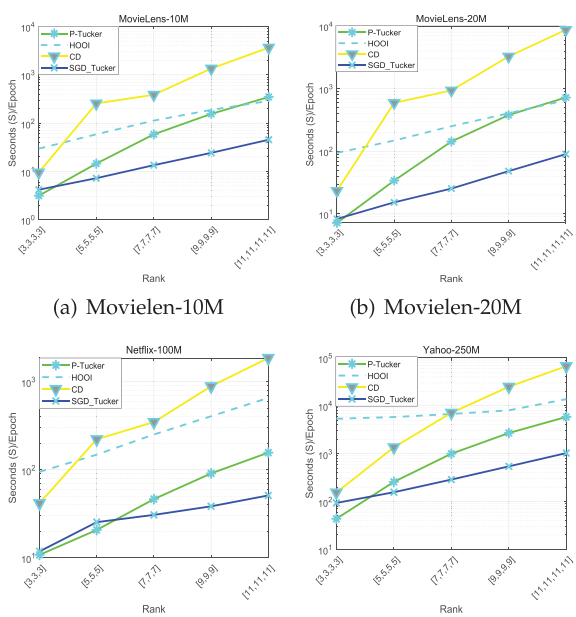
TABLE 3
Datasets

Parameters \ Datasets	Movielens-100K	Movielens-1M	Movielens-10M	Movielens-20M	Netflix-100M	Yahoo-250M
N_1	943	6, 040	71, 567	138, 493	480, 189	1, 000, 990
N_2	1, 682	3, 706	10, 677	26, 744	17, 770	624, 961
N_3	2	4	15	21	2, 182	133
N_4	24	24	24	24	-	24
$\|\Omega\|$	90, 000	990, 252	9, 900, 655	19, 799, 448	99, 072, 112	227, 520, 273
$\|\Gamma\|$	10, 000	9, 956	99, 398	200, 815	1, 408, 395	25, 280, 002
Order	4	4	4	4	3	4
Max Value	5.0	5.0	5.0	5.0	5.0	5.0
Min Value	0.5	0.5	0.5	0.5	1.0	1.0

5.2 Scalability of the Approach

When ML algorithms process large-scale data, two factors to influence the computational overhead are: 1) time spent for data access on memory, i.e., reading and writing data; 2) time spent on the computational components for executing the ML algorithms. Fig. 5 presents the computational overhead per training epoch. The codes of P−Tucker and CD have fixed settings for the rank value, i.e., $J_1 = \cdots = J_n = \cdots = J_N$. The rank value is set as an increasing order, $\{3, 5, 7, 9, 11\}$. As shown in Fig. 5, SGD_Tucker has the lowest computational time overhead. HOOI needs to construct the intermediate matrices $\mathbf{Y}_{(n)} \in \mathbb{R}^{I_n \times \prod_{k \neq n}^{N} J_k}$, $n \in \{N\}$ and SVD for $\mathbf{Y}_{(n)}$ (In supplementary material), available online. P−Tucker should construct Hessian matrix and the computational complexity of each Hessian matrix inversion is $O(J_n^3)$, $n \in \{N\}$. P−Tucker saves the memory overhead a lot at the expense of adding the memory accessing overhead. Eventually, the accessing and computational overheads for inverse operations of Hessian matrices make the overall computational time of P−Tucker surpass SGD_Tucker. The computational structure of CD has linear scalability. However, the CD makes the update process of each feature element in a feature vector be discrete. Thus, the addressing process is time-consuming. Meanwhile, as Fig. 5 shows, the computational overhead satisfies the constraints of the time complexity analyses, i.e., P−Tucker, HOOI, CD (Supplementary material, available online) and SGD_Tucker (Section 4.5).

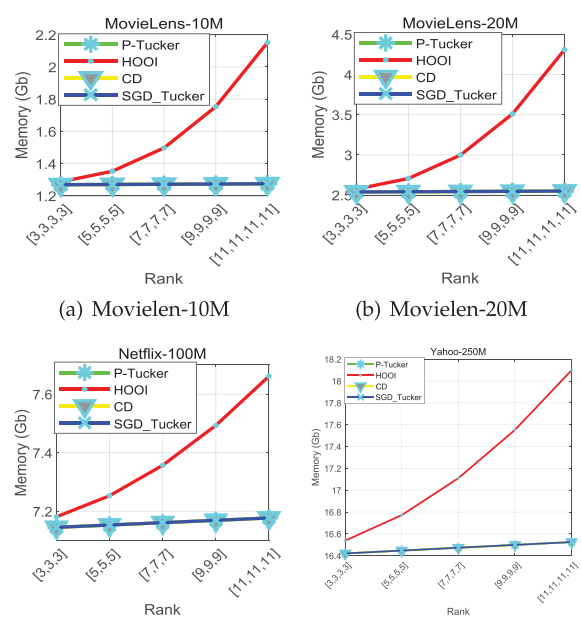
The main space overheads for STD comes from the storage need of the intermediate matrices. The SVD operation for the intermediate matrices $\mathbf{Y}_{(n)} \in \mathbb{R}^{I_n \times \prod_{k \neq n}^{N} J_k}$, $n \in \{N\}$ makes the HOOI scale exponentially. P−Tucker constructs the eventual Hessian matrices directly which can avoid the huge construction of intermediate matrices. However, the overhead for addressing is huge and the long time-consuming for CD lies in the same situation. CD is a linear approach from the update equation (Supplementary material, available online). However, in reality, the linearity comes from the approximation of the second-order Hessian matrices and, thus, the accessing overhead of discrete elements of CD overwhelms the accessing overhead of continuous ones (P−Tucker and SGD_Tucker). As shown in Fig. 6, SGD_Tucker has linear scalability for space overhead and HOOI is exponentially scalable. P−Tucker and CD have



(a) Movielen-10M (b) Movielen-20M

(c) Netflix-100M (d) Yahoo-250M

Fig. 5. Rank scalability for time overhead on full threads. The computational scalability for P−Tucker, HOOI, CD, and SGD_Tucker on the 4 datasets with successively increased number of total elements, i.e.,Movielen-10M, Movielen-20M, Netflix-100M, and Yahoo-250M.

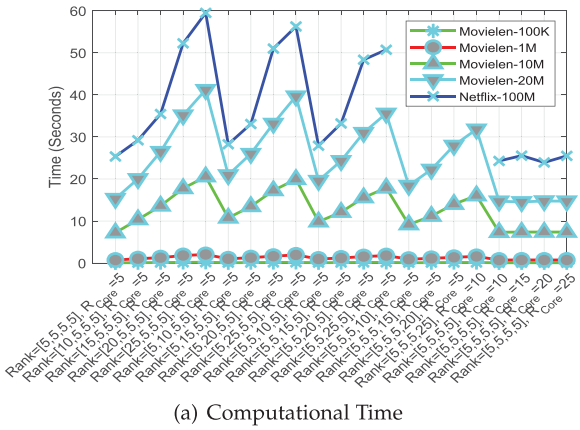


(a) Movielen-10M (b) Movielen-20M

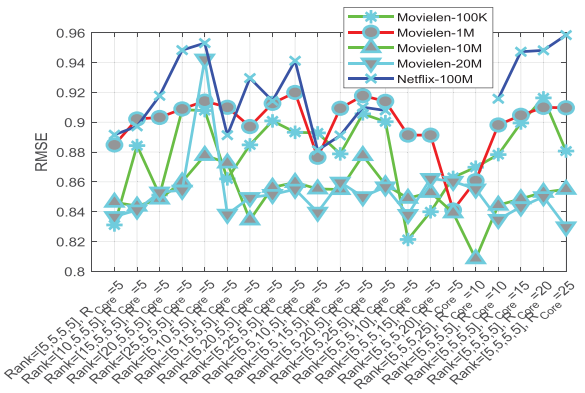
(c) Netflix-100M (d) Yahoo-250M

Fig. 6. Rank scalability for memory overhead on a thread running. (In this work, GB refers to GigaBytes and MB refers to Megabytes). The space scalability for P−Tucker, CD, HOOI, and SGD_Tucker on the 4 datasets with successively increased total elements, i.e., Movielen-10M, Movielen-20M, Netflix-100M, and Yahoo-250M.

Authorized licensed use limited to: TU Delft Library. Downloaded on February 21,2021 at 18:47:34 UTC from IEEE Xplore. Restrictions apply.



(a) Computational Time



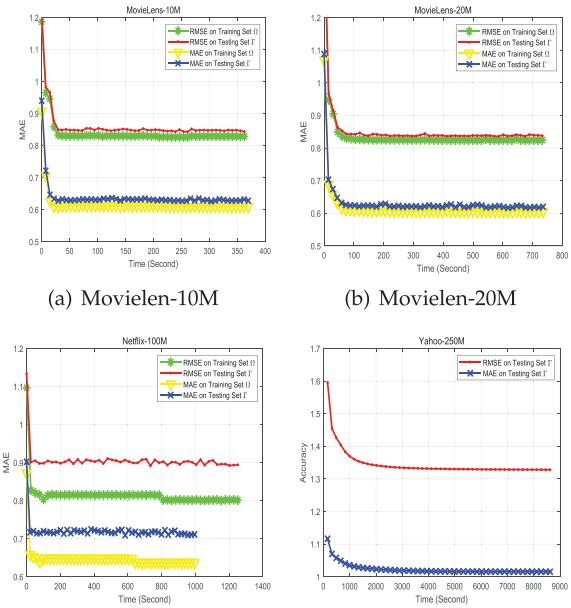
(b) RMSE

Fig. 7. Computational overhead, RMSE, and MAE VS. various rank values for SGD_Tucker on 8 Cores. Due to limited space, each dataset is presented in 4-order. As Netflix-100M data set only has 3-order, its 4th index shall be neglected.

the near-linear space overhead. However, as Fig. 5 shows, the computational overheads of P−Tucker and CD is significantly higher than SGD_Tucker. Meanwhile, as Fig. 6 shows, the space overhead satisfies the constraints of the space complexity analyses, i.e., HOOI, P−Tucker, and CD (Supplementary material, available online) and SGD_Tucker (Section 4.5).

5.3 The Influence of Parameters

In this section, the influence of the rank on the behavior of the algorithm is presented. We summarize the influence on the computational time and the RMSE are presented in Fig. 7. The Netflix-100M dataset has only 3-order. Owing to the limited space, the performances on Netflix-100M dataset are combined with other 5 datasets (Netflix-100M dataset has only 3-order and the performances on the 4th order of Netflix-100M dataset should be negligible). The update for the Kruskal matrices $\mathbf{B}^{(n)}, n \in N$ on the steps of beginning and last epochs can also obtain a comparable results. Thus, the presentation for the Kruskal matrices $\mathbf{B}^{(n)}$, $n \in N$ is omitted. The influence for computational time is divided into 5 sets, i.e., $\{J_1 \in \{5, 10, 15, 20, 25\}, J_k = 5, k \neq 1, R_{Core} = 5\}$, $\{J_2 \in \{5, 10, 15, 20, 25\}, J_k = 5, k \neq 2, R_{Core} = 5\}$, $\{J_3 \in \{5, 10, 15, 20, 25\}, J_k = 5, k \neq 3, R_{Core} = 5\}$, $\{J_4 \in \{5, 10, 15, 20, 25\}, J_k = 5, k \neq 4, R_{Core} = 5\}$, $\{J_n = 5, n \in \{N\}, R_{Core} \in \{10, 15, 20, 25\}\}$. As the Fig. 7a shows, the computation time increases with J_n, $n \in \{N\}$ increasing linearly and the R_{Core} only has slight influence for computational time.



(a) Movielen-10M (b) Movielen-20M

(c) Netflix-100M (d) Yahoo-250M

Fig. 8. RMSE and MAE versus time for SGD_Tucker on training set Ω and testing set Γ.

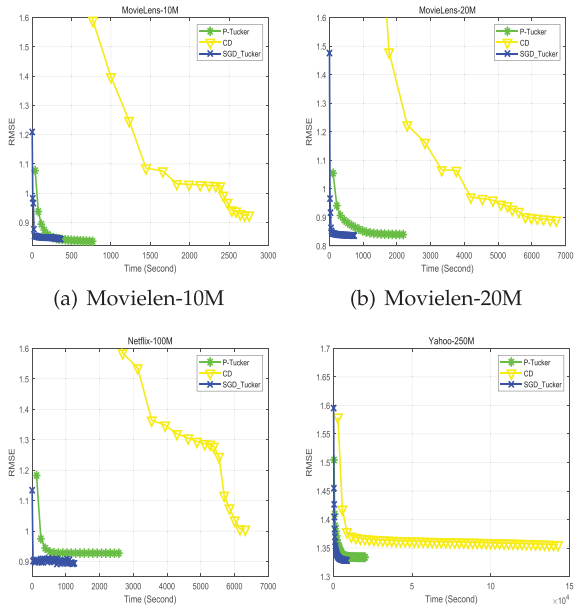
The codes of P−Tucker and CD have fixed settings for the rank value ($J_1 = \cdots = J_n = \cdots = J_N$). For a fair comparison, the rank values that we select should have slight changes with the RMSE performances of other rank values. The computational overhead of P−Tucker and CD is sensitive to the rank value and RMSE is comparable non-sensitive. Hence, we choose a slightly small value $J_n = 5 | n \in \{N\}$. As Fig. 7b shows, when rank is set as $J_n = 5, n \in \{N\}$, the RMSE is equivalent to other situations on average.

5.4 Speedup and Comparisons

We test the RMSE performances on the 6 datasets and the RMSE is used to estimate the missing entries. The speedup is evaluated as $Time_1/Time_T$ where $Time_T$ is the running time with T threads. When $T = \{1, \ldots, 8\}$, as the Fig. 10 shows, the speedup performance of SGD_Tucker has a near-linear speedup. The speedup performance is presented in Fig. 10 which shows the whole speedup performance of the SGD_Tucker. The speedup performance is a bit more slower when the number of threads is larger than 8. The reason is that the thread scheduling and synchronization occupies a large part of time. When we use 16 threads, there is still a $11X$ speedup and the efficiency is $11/16 = 68\%$.

The rank value for 3-order tensor is set to [5,5,5] and the rank value for 4-order tensor is set to [5,5,5,5]. To demonstrate the convergence of SGD_Tucker, the convergence performances of SGD_Tucker for the Movielen and Netflix are presented in Fig. 8. As shown in Fig. 8, SGD_Tucker can get more stable RMSE and MAE metrices which mean that SGD_Tucker has more robustness in large-scale datasets than in small datasets. As Fig. 9 shows, SGD_Tucker can not only run faster than the state of the art approaches, i.e., P−Tucker and CD, but also can obtain higher RMSE value. SGE_Tucker runs $2X$ and $20X$ faster than P−Tucker and CD, respectively, to obtain the optimal RMSE value.

Authorized licensed use limited to: TU Delft Library. Downloaded on February 21,2021 at 18:47:34 UTC from IEEE Xplore. Restrictions apply.



(a) Movielen-10M (b) Movielen-20M

(c) Netflix-100M (d) Yahoo-250M

Fig. 9. RMSE comparison of SGD_Tucker, P−Tucker, and CD on the 4 datasets.

6 Conclusion and Future Works

STD is widely used in low-rank representation learning for sparse big data analysis. Due to the entanglement problem of core tensor and factor matrices, the computational process for STD has the intermediate variables explosion problem due to following all elements in HOHDST. To solve this problem, we first derive novel optimization objection function for STD and then propose SGD_Tucker to solve it by dividing the high-dimension intermediate variables into small batches of intermediate matrices. Meanwhile, the low data-dependence of SGD_Tucker makes it amenable to fine-grained parallelization. The experimental results show that SGD_Tucker has linear computational time and space overheads and SGD_Tucker runs at least $2X$ faster than the state of the art STD solutions. In the future works, we plan to explore how to accelerate the SGD_Tucker by the state of the art stochastic models, e.g., variance SGD [32], Stochastic Recursive Gradient [34], or momentum SGD [35].

SGD_Tucker is a linearly scalable method for STD on big-data platforms. For the future work, we will embed SGD_Tucker into popular distributed data processing platforms such as Hadoop, Spark, or Flink. We will also aim to extend SGD_Tucker to GPU platforms, i.e., OpenCL and CUDA.

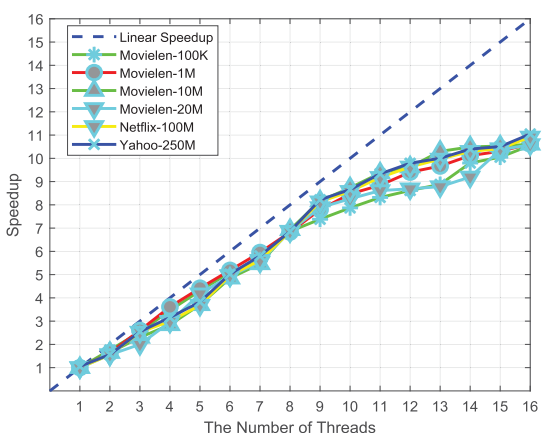


Fig. 10. Speedup performance on the 6 datasets.

Acknowledgments

This work has been partly funded by the Swiss National Science Foundation NRP75 project (Grant No. 407540_167266), the China Scholarship Council (CSC) (Grant No. CSC20190 6130109). This work has also been partly funded by the Program of National Natural Science Foundation of China (Grant No. 61751204), the National Outstanding Youth Science Program of National Natural Science Foundation of China (Grant No. 61625202), and the International (Regional) Cooperation and Exchange Program of National Natural Science Foundation of China (Grant No. 61860206011). We would like to express our gratitude to Smith, Shaden and De-nian Yang who help us to finish the overall experiments.

References

[1] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic, "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions," *Found. Trends® Mach. Learn.*, vol. 9, no. 4/5, pp. 249–429, 2016.

[2] V. N. Ioannidis, A. S. Zamzam, G. B. Giannakis, and N. D. Sidiropoulos, "Coupled graph and tensor factorization for recommender systems and community detection," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2019.2941716.

[3] X. Luo, H. Wu, H. Yuan, and M. Zhou, "Temporal pattern-aware QoS prediction via biased non-negative latent factorization of tensors," *IEEE Trans. Cybern.*, vol. 50, no. 5, pp. 1798–1809, May 2020.

[4] K. Xie, X. Li, X. Wang, G. Xie, J. Wen, and D. Zhang, "Active sparse mobile crowd sensing based on matrix completion," in *Proc. Int. Conf. Manage. Data*, 2019, pp. 195–210.

[5] X. Wang, L. T. Yang, X. Xie, J. Jin, and M. J. Deen, "A cloud-edge computing framework for cyber-physical-social services," *IEEE Commun. Mag.*, vol. 55, no. 11, pp. 80–85, Nov. 2017.

[6] P. Wang, L. T. Yang, G. Qian, J. Li, and Z. Yan, "HO-OTSVD: A novel tensor decomposition and its incremental decomposition for cyber-physical-social networks (CPSN)," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 2, pp. 713–725, Second Quarter 2020.

[7] T. Wang, X. Xu, Y. Yang, A. Hanjalic, H. T. Shen, and J. Song, "Matching images and text with multi-modal tensor fusion and re-ranking," in *Proc. 27th ACM Int. Conf. Multimedia*, 2019, pp. 12–20.

[8] M. Hou, J. Tang, J. Zhang, W. Kong, and Q. Zhao, "Deep multimodal multilinear fusion with high-order polynomial pooling," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 12 113–12 122.

[9] P. P. Liang, Z. Liu, Y.-H. H. Tsai, Q. Zhao, R. Salakhutdinov, and L.-P. Morency, "Learning representations from imperfect time series data via tensor rank regularization," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 1569–1576.

[10] Y. Liu, L. He, B. Cao, S. Y. Philip, A. B. Ragin, and A. D. Leow, "Multi-view multi-graph embedding for brain network clustering analysis," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 117–124.

[11] A. Cichocki *et al.*, "Tensor networks for dimensionality reduction and large-scale optimizations: Part 2 applications and future perspectives," *Found. Trends Mach. Learn.*, vol. 9, no. 6, pp. 431–673, 2017.

[12] L. Van Der Maaten, E. Postma, and J. Van den Herik, "Dimensionality reduction: a comparative," *J. Mach. Learn. Res.*, vol. 10, no. 66/71, 2009, Art. no. 13.

[13] J. Xu, J. Zhou, P.-N. Tan, X. Liu, and L. Luo, "Spatio-temporal multi-task learning via tensor decomposition," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2019.2956713.

[14] X. Liu, X. You, X. Zhang, J. Wu, and P. Lv, "Tensor graph convolutional networks for text classification," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 8409–8416.

[15] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 265–283.

Authorized licensed use limited to: TU Delft Library. Downloaded on February 21,2021 at 18:47:34 UTC from IEEE Xplore. Restrictions apply.

[16] J. Kossaifi, A. Bulat, G. Tzimiropoulos, and M. Pantic, "T-net: Parametrizing fully convolutional nets with a single high-order tensor," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 7822–7831.

[17] F. Ju, Y. Sun, J. Gao, M. Antolovich, J. Dong, and B. Yin, "Tensorizing restricted Boltzmann machine," *ACM Trans. Knowl. Discov. Data*, vol. 13, no. 3, pp. 1–16, 2019.

[18] M. Wang, Z. Shu, S. Cheng, Y. Panagakis, D. Samaras, and S. Zafeiriou, "An adversarial neuro-tensorial approach for learning disentangled representations," *Int. J. Comput. Vis.*, vol. 127, no. 6/7, pp. 743–762, 2019.

[19] C. Zhang, H. Fu, J. Wang, W. Li, X. Cao, and Q. Hu, "Tensorized multi-view subspace representation learning," *Int. J. Comput. Vis.*, vol. 128, pp. 2344–2361, 2020.

[20] Y. Luo, D. Tao, K. Ramamohanarao, C. Xu, and Y. Wen, "Tensor canonical correlation analysis for multi-view dimension reduction," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 11, pp. 3111–3124, Nov. 2015.

[21] X. Liu *et al.*, "Multiple kernel k-means with incomplete kernels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 5, pp. 1191–1204, May 2020.

[22] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos, "Tag recommendations based on tensor dimensionality reduction," in *Proc. ACM Conf. Recommender Syst.*, 2008, pp. 43–50.

[23] I. Balazevic, C. Allen, and T. Hospedales, "TuckER: Tensor factorization for knowledge graph completion," in *Proc. Conf. Empir. Methods Natural Lang. Process. and the 9th Int. Joint Conf. Natural Lang. Process.*, 2019, pp. 5188–5197.

[24] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 208–220, Jan. 2013.

[25] S. Zubair and W. Wang, "Tensor dictionary learning with sparse tucker decomposition," in *Proc. Int. Conf. Digit. Signal Process.*, 2013, pp. 1–6.

[26] S. Oh, N. Park, S. Lee, and U. Kang, "Scalable tucker factorization for sparse tensors-algorithms and discoveries," in *Proc. IEEE Int. Conf. Data Eng.*, 2018, pp. 1120–1131.

[27] V. T. Chakaravarthy *et al.*, "On optimizing distributed tucker decomposition for sparse tensors," in *Proc. Int. Conf. Supercomputing*, 2018, pp. 374–384.

[28] B. N. Sheehan and Y. Saad, "Higher order orthogonal iteration of tensors (HOOI) and its relation to PCA and GLRAM," in *Proc. SIAM Int. Conf. Data Mining*, 2007, pp. 355–365.

[29] A. Kulunchakov and J. Mairal, "A generic acceleration framework for stochastic composite optimization," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 12 556–12 567.

[30] A. Traoré, M. Berar, and A. Rakotomamonjy, "Singleshot: A scalable tucker tensor decomposition," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 6301–6312.

[31] O. A. Malik and S. Becker, "Low-rank tucker decomposition of large tensors using TensorSketch," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 10 096–10 106.

[32] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 315–323.

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

[34] L. M. Nguyen, J. Liu, K. Scheinberg, and M. Takáč, "SARAH: A novel method for machine learning problems using stochastic recursive gradient," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2613–2621.

[35] H. Yu, R. Jin, and S. Yang, "On the linear speedup analysis of communication efficient momentum SGD for distributed non-convex optimization," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 7184–7193.

[36] S. Shalev-Shwartz *et al.*, "Online learning and online convex optimization," *Found. Trends Mach. Learn.*, vol. 4, no. 2, pp. 107–194, 2012.

[37] N. Ketkar, "Introduction to pytorch," in *Deep Learning With Python*. Berlin, Germany: Springer, 2017, pp. 195–208.

[38] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Comput. Surv.*, vol. 53, no. 2, Mar. 2020, Art. no. 30.

[39] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, "Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD," in *Proc. 21st Int. Conf. Artif. Intell. Statist.*, 2018, pp. 803–812.

[40] H. Ge, K. Zhang, M. Alfifi, X. Hu, and J. Caverlee, "DisTenC: A distributed algorithm for scalable tensor completion on spark," in *Proc. IEEE 34th Int. Conf. Data Eng.*, 2018, pp. 137–148.

[41] S. Smith and G. Karypis, "Accelerating the tucker decomposition with compressed sparse tensors," in *Proc. Eur. Conf. Parallel Process.*, 2017, pp. 653–668.

[42] Y. Ma, J. Li, X. Wu, C. Yan, J. Sun, and R. Vuduc, "Optimizing sparse tensor times matrix on GPUs," *J. Parallel Distrib. Comput.*, vol. 129, pp. 99–109, 2019.

[43] V. T. Chakaravarthy, S. S. Pandian, S. Raje, and Y. Sabharwal, "On optimizing distributed non-negative tucker decomposition," in *Proc. ACM Int. Conf. Supercomputing*, 2019, pp. 238–249.

[44] O. Kaya and B. Uçar, "High performance parallel algorithms for the tucker decomposition of sparse tensors," in *Proc. 45th Int. Conf. Parallel Process.*, 2016, pp. 103–112.

[45] V. T. Chakaravarthy *et al.*, "On optimizing distributed tucker decomposition for sparse tensors," in *Proc. Int. Conf. Supercomputing*, 2018, pp. 374–384.

[46] S. Oh, N. Park, S. Lee, and U. Kang, "Scalable tucker factorization for sparse tensors-algorithms and discoveries," in *Proc. IEEE Int. Conf. Data Eng.*, 2018, pp. 1120–1131.

[47] M. Park, J.-G. Jang, and S. Lee, "VeST: Very sparse tucker factorization of large-scale tensors," 2019, *arXiv: 1904.02603*.

[48] S. Oh, N. Park, J.-G. Jang, L. Sael, and U. Kang, "High-performance tucker factorization on heterogeneous platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 10, pp. 2237–2248, Oct. 2019.

[49] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, vol. 87. Berlin, Germany: Springer Science & Business Media, 2013.

[50] Y. Chi, Y. M. Lu, and Y. Chen, "Nonconvex optimization meets low-rank matrix factorization: An overview," *IEEE Trans. Signal Process.*, vol. 67, no. 20, pp. 5239–5269, Oct. 2019.

[51] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM J. Imag. Sci.*, vol. 6, no. 3, pp. 1758–1789, 2013.

[52] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

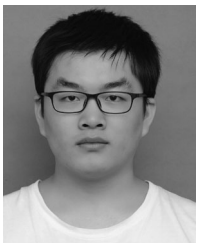
[53] H. Li, C. Fang, and Z. Lin, "Accelerated first-order optimization algorithms for machine learning," *Proc. IEEE*, vol. 108, no. 11, pp. 2067–2082, Nov. 2020.

[54] S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. Smola, "Stochastic variance reduction for nonconvex optimization," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 314–323.

[55] H. Li, K. Li, J. An, and K. Li, "MSGD: A novel matrix factorization approach for large-scale collaborative filtering recommender systems on GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 7, pp. 1530–1544, Jul. 2018.

Hao Li (Student Member, IEEE) is currently working toward the PhD degree at Hunan University, China. He is currently a visiting PhD student with TU Delft, Netherlands, 2019-2021. His research interests are mainly in large-scale sparse matrix and tensor factorization, recommender systems, machine learning, and parallel and distributed computing. He has published several journal and conference papers in the *IEEE Transactions on Parallel and Distributed Systems*, *Information Sciences*, *IEEE Transactions on Industrial Informatics*, CIKM, and ISPA. He also serves reviewer of the top-tier conferences and journals, e.g., HPCC, IJCAI, WWW, the *Neurocomputing*, the *IEEE Access*, *Journal of Parallel and Distributed Computing*, the *Information Sciences*, *IEEE Transactions on Network and Service Management*, *IEEE Internet of Things Journal*, *ACM Transactions on Knowledge Discovery from Data* and *IEEE Transactions on Dependable and Secure Computing*.

Authorized licensed use limited to: TU Delft Library. Downloaded on February 21,2021 at 18:47:34 UTC from IEEE Xplore. Restrictions apply.



Zixuan Li received the graduated degree from the College of Mathematics and Econometrics, Hunan University, China, in 2017. He is currently working toward the master's degree at Hunan University, China. His research interests are mainly in large-scale sparse matrix and tensor factorization, recommender systems, data mining, graph neural network, bayesian inference and probabilistic graph model.

Kenli Li (Senior Member, IEEE) received the PhD degree in computer science from the Huazhong University of Science and Technology, China, in 2003. He was a visiting scholar with the University of Illinois at Urbana-Champaign, Champaign, Illinois from 2004 to 2005. He is currently a full professor of computer science and technology at Hunan University, China, and deputy director of National Supercomputing Center in Changsha. His major research areas include parallel computing, high-performance computing, grid and cloud computing. He has published more than 130 research papers in international conferences and journals such as the *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Signal Processing*, *Journal of Parallel and Distributed Computing*, ICPP, CCGrid. He is an outstanding member of CCF. He is serves on the editorial board of the *IEEE Transactions on Computers*.

Jan S. Rellermeyer (Member, IEEE) received the MSc and PhD degrees in distributed systems from ETH Zürich, Switzerland and spent several years in industry. He is currently an assistant professor with the Department of Computer Science, TU Delft, Netherlands. He was an Apache Committer and is an active Eclipse committer and projet lead of the IoT Concierge project. His research revolves around the system architecture and the dependability aspects of large-scale distributed systems for demanding applications like big data processing and distributed machine learning. He authored several highly cited peer-reviewed publications, has been an active reviewer for a variety of top-tier conferences and journals, and, together with his co-authors Gustavo Alonso and Timothy Roscoe, received the 10 Years Test-of-Time Award at the ACM/IFIP/USENIX Middleware Conference ins 2017 for his work on R-OSGi: Distributed Applications through Software Modularization.

Lydia Chen (Senior Member, IEEE) received the BA degree from National Taiwan University, Taiwan, and the PhD degree from Pennsylvania State University, State College, Pennsylvania, in 2002 and 2006, respectively. She is currently an associate professor with the Department of Computer Science, Delft University of Technology, Netherlands. Her research interests center around dependability management, resource allocation and privacy enhancement for large scale data processing systems and services. More specifically, her work focuses on developing stochastic and machine learning models and applying these techniques to application domains, such as datacenters and AI systems. She has published more than 80 papers in journals, e.g., the *IEEE Transactions on Distributed Systems*, *IEEE Transactions on Service Computing*, and conference proceedings, e.g., INFOCOM, Sigmetrics, DSN, and Eurosys. She was a co-recipient of the best paper awards at CCgrid'15 and eEnergy'15. She received TU Delft Professor fellowship, in 2018. She was program co-chair for Middleware Industry Track 2017 and IEEE ICAC 2019 and track vice-chair for ICDCS 2018. She has served on the editorial boards of the *IEEE Transactions on Network and Service Management*, *IEEE Transactions on Service Computing*, *IEEE Transactions on Dependable and Secure Computing* and *IEEE Transactions on Parallel and Distributed Systems*.

Keqin Li (Fellow, IEEE) is currently a SUNY distinguished professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems. He has published more than 480 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Sustainable Computing*.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: TU Delft Library. Downloaded on February 21,2021 at 18:47:34 UTC from IEEE Xplore. Restrictions apply.