Neurocomputing 399 (2020) 18–36

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

An online and generalized non-negativity constrained model for large-scale sparse tensor estimation on multi-GPU

Linlin Zhuo[a], Kenli Li[a,*], Hao Li[a], Jiwu Peng[a], Keqin Li[a,b,**]

[a] College of Computer Science and Electronic Engineering, Hunan University, Changsha, 410082, China
[b] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Article history:
Received 29 July 2019
Revised 18 November 2019
Accepted 16 February 2020
Available online 22 February 2020

Communicated by Dr Xin Luo

Keywords:
Canonical polyadic decomposition
Generalized model
GPU and multi-GPU
High-dimension and sparse data
Single-thread-based model
Sparse non-negative tensor factorization
Online learning
Stream-like data merging

ABSTRACT

Non-negative Tensor Factorization (NTF) models are effective and efficient in extracting useful knowledge from various types of probabilistic distribution with multi-way information. Current NTF models are mostly designed for problems in computer vision which involve the whole *Matricized Tensor Times Khatri – Rao Product* (MTTKRP). Meanwhile, a Sparse NTF (SNTF) proposed to solve the problem of sparse Tensor Factorization (TF) can result in large-scale intermediate data. A Single-thread-based SNTF (SSNTF) model is proposed to solve the problem of non-linear computing and memory overhead caused by large-scale intermediate data. However, the SSNTF is not a generalized model. Furthermore, the above methods cannot describe the stream-like data from industrial applications in mainstream processors, e.g, Graphics Processing Unit (GPU) and multi-GPU in an online way. To address these two issues, a Generalized SSNTF (GSSNTF) is proposed, which extends the works of SSNTF to the Euclidean distance, KullbackLeibler (KL)-divergence, and ItakuraSaito (IS)-divergence. The GSSNTF only involves the feature elements instead of the entire factor matrices during its update process, which can avoid the formation of large-scale intermediate matrices with convergence and accuracy promises. Furthermore, GSSNTF can merge the new data into the state-of-the-art built tree dataset for sparse tensor, and then online learning has the promise of the correct data format. At last, a model of Compute Unified Device Architecture (CUDA) parallelizing GSSNTF (CUGSSNTF) is proposed on GPU and Multi-GPU (MCUGSSNTF). Thus, CUGSSNTF has linear computing complexity and space requirement, and linear communication overhead on multi-GPU. CUGSSNTF and MCUGSSNTF are implemented on 8 P100 GPUs in this work, and the experimental results from real-world industrial data sets indicate the linear scalability and 40X speedup performances of CUGSSNTF than the state-of-the-art parallelized approachs.

1. Introduction

With the rapid development of the Internet and the variety of information collection approach, i.e., spatio-temporal, optic, thermology, vibration, and voice, multi-way data has gradually become a major representation form for information, and arises in many applications, e.g., dimension reduction, clustering, recommender systems, social network, and modeling on wireless communication networks, smart city and Internet of Things (IoT), etc, [1]. These type of data is represented by tensor whose number of dimensions is greater than three. Tensor preserves the interactive information for each dimension. The more the amount of information a system has, the higher the dimension a tensor has; however, in addition to having high dimensions, the data is highly sparse, and the reason for data missing is that the systems are time-varying and it's hard to obtain the full information in real-time [2]. We observe that the scenes that a system misses information are common in reality, e.g., information networks, recommender systems, and Quality of Service (QoS), etc. In medical image processing communities, to obtain a more clear Magnetic Resonance Imaging (MRI) of a human organ, the doctor should increase the radiation dose; however, over-radiation will damage the health. A solution is that the researcher can recover a clear MRI image from the obscured and sparse one, which is obtained by the low radiation dose MRI

* Corresponding author.
** Corresponding author at: College of Computer Science and Electronic Engineering, Hunan University, Changsha, 410082, China.
E-mail addresses: zhuoninnin@hnu.edu.cn (L. Zhuo), lkl@hnu.edu.cn (K. Li), lihao123@hnu.edu.cn (H. Li), pjw@hnu.edu.cn (J. Peng), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.neucom.2020.02.068
0925-2312/

[3]. An estimation way for the missing data is via TF, which takes advantage of the low-rank structure of the data [4]. Due to the high efficiency and low computing complexity, Canonical Polyadic Decomposition (CPD) becomes the most popular factorization method [5]. Because of the low-rank and non-negativity essence of many data, NTF has become one of the most popular approaches in CPD communities [6,7]. Meanwhile, something should be observed that the MTTKPP is deduced from gradient descent (GD) which involves in the manipulation of unfolding tensor with Khatri-Rao product of $N-1$ matrices [8–12]. Due to explosive increased space and computational overhead, MTTKPP is a computational bottleneck of CPD [8–12].

Due to the powerful processing ability for fine-grained and stream-like tasks, GPU has become a popular processor in industrial application, and in cloud platforms, i.e., Flink and Spark, etc, GPU gradually plays a role as a core processor [13]. However, SNTF suffers the following computing problems on the GPU: (1) The sparsity will add extra computing burdens. The update process for each feature vector of Dense NTF (DNTF), which is designed to computer vision and the image and video present a dense style [14,15], shares the same Hessian matrix, which only needs a Khatri-Rao product of $N-1$ matrices. However, the update process for each feature vector of SNTF don't share the same Hessian matrix, which only need N Khatri-Rao product of $N-1$ matrices; (2) Due to memory limitation, GPU cannot process frequent large-scale matrices manipulation and store non-linear increased intermediate matrices without tuning the update approach of DNTF; (3) The intermediate Hessian matrices need square communication costs. SSNTF [16] is proposed to solve these problems which are caused by large-scale intermediate matrices based on Euclidean distance. However, it is not applied to the generalized distance, e.g., KL-divergence and IS-divergence, etc. Thus, it is also not a generalized model. What's more, the above methods cannot describe the stream-like data from industrial applications in mainstream processors, e.g, GPU and multi-GPU in an online way.

The GSSNTF is proposed to perfect the theoretic flaw on large-scale industrial data. The matrix manipulations will be transformed into the needed multiplication and summation operations of feature vector elements. At the same time, our method benefits from its more generalized model and its online processing. The main contributions of our study are as follows:

1. **Algorithm analysis**. The GSSNTF model is derived theoretically, which obeys various common probabilistic styles, e.g., $\mathcal{G}$aussian, $\mathcal{P}$oisson, and $\mathcal{E}$xponential distribution styles, and fine-grained parallelization inherence. And GSSNTF represents a more generalized update rule for factor matrices. (see Sections 4.1 and 4.2).
2. **CUDA Parallelization** and **Multi-GPU**. CUGSSNTF harnesses local memory more instead of increasing the global memory overhead of the GPU, and CUGSSNTF has linear scalability of computation and space overhead (see Section 4.3.1). MCUGSSNTF adopts a multi-GPU model with mode scalability for tensor data. GSSNTF and CUGSSNTF have linear time and space complexities. (see Section 4.3.2).
3. **Online Learning**. The stream-like computing style of the single-thread-based model gives the GSSNTF an online learning ability. Online learning and high efficient CUDA parallelization are the two byproducts of the single-thread-based model. (see Section 4.4).

The rest of this paper is organized as follows. Section 2 presents related works. Section 3 lays out the problem formulation and GPU computing. Section 4 introduces the GSSNTF model, CUGSSNTF, MCUGSSNTF and online learning. The experimental results and discussion on future works are presented on Sections 5 and 6, respectively.

2. Related work

Matrix can represent two-way and single modal data [17], i.e., image, text, and video, etc. However, due to abundant information collection equipment, in the industrial era, the source of modern data comes from heterogeneous and multi-modal information. Thus, a real dataset from industrial applications may comprise of audio, video, and image, etc, and matrix cannot be competent to the representation ability. However, tensor can represent those data types [18,19]. Matrix Factorization (MF) and TF can extract the latent information for matrix and tensor respectively, and then, the latent information can approximate the original matrix and tensor. Meanwhile, non-negative MF (NMF) and NTF can keep the non-negative and inherent latent information for matrix and tensor respectively. NTF has been drawn wide attention, and NTF is a generalized factor analysis method for multi-way data, and it is an extension of NMF, which can only represent 2-way matrix data [20,21]. NMF is an useful tool for low-rank representation, which can be used as K-means based image clustering [22], subspace based clustering [23] [24], QoS [25,26], manifold space based clustering [27], and unmixing for hyperspectral image [28]. Clustering is an unsupervised method in machine learning communities, which can find the several inherent similarity features; however, when the data increase rapidly, direct clustering will face the curse of dimensionality [29] and applying cluster method such as K-means on the projected low-rank space can obtain the same clustering accuracy [22]. NMF can mine the non-negativity and low-rank features of big data, which is a preparation step for K-means methods [22]. Subspace clustering can search the clusters in different dimension [23], and NMF for low-rank representation plays a great role in subspace segmentation [24]. When high-dimension data and feature have intrinsic manifold structure, NMF can cooperate with the manifold information with graph regularization [30]. Furthermore, NMF can unmix the blended information of the satellite obtained a hyperspectral image, which can take advantage of low-rank structure [31]. However, the NMF only consider 2-dimension information and cannot fuse the extra information, e.g., Spatio-temporal dynamics information, etc [32]. NTF can make up the disadvantage of NMF by introducing low-rank feature matrices and tensor algebra manipulation [33,34].

It can be observed that research on NTF is mainly reflected in small and dense data sets in the field of computer vision, i.e., [14,15]. Thus, the formation of the update rules can follow the matrices operations regardless of memory limitation. However, in real-world industrial applications, in addition to having high dimensions, the data is highly sparse, and direct matrix manipulations will result in redundant time and space overhead. For example, the prior works on SNTF of Euclidean distance [15], the construction of the intermediate Gram matrices during their process, which cannot be shared by each row feature vector of a factor matrix in SNTF, will result in increased time and space overhead. Thus, space and computational time overheads will increase sharply. Meanwhile, there are three research topics on accelerating tensor factorization: (1) Mathematical optimization for basic optimization methods [33]; (2) Compressive data structure [8–10]; (3) Parallel and distributed algorithm designing for basic optimization methods [11,12].

The formation of the update rules for the KL- and IS-divergence involves the formation of the whole approximation tensor, and the element is redundant, the position of which is the corresponding zero elements of the target sparse tensor. Furthermore, the generalized NTF model can extract the low-rank inherence of a dataset;

however, when the generalized NTF faces the sparse datasets, the following 3 real application problems should be solved: (1) How to distinguish the unevenly distributed non-zero and zero values; (2) How to reduce the computing and memory overheads for the distinguishing mechanism; (3) How to keep the parallelism. Kim et al. [35] and Choi et al. [36] introduced a weight tensor strategy. The strategy used the weight tensor to distinguish the zero and non-zero values in a sparse tensor; however, the memory overhead of the weight tensor may overwhelm the amount of non-zero values and the unevenly distributed non-zero values limit the usage of the weight tensor.

The modern computer has the linear access model. Thus, a tensor with 3 or more ways needs vectorization on computer memory hardware. Furthermore, sparse tensor from industrial applications has an uneven distribution of non-zero value. Thus, high efficient access and vectorization should be considered for sparse tensor data on computer hardware. Chen et al. [37]. proposed a probabilistic distribution based combinational approach, which can combine several data structures for compressive sparse matrix by probabilistic statistical style. This combination model cannot solve the problem of uneven distribution for non-zero elements of a sparse tensor and sparse tensor transformation. Smith et al. [8]. proposed a Compressed Sparse Fiber (CSF) structure for compressive preservation of a sparse tensor, and Li et al. [38]. designed a efficient sparse tensor transformation method.

Alternative Least Square (ALS) and Cyclic Coordinate Descend (CCD) are two common optimization algorithms for unconstraint minimization optimization. ALS needs to consider the relationship between elements within a feature vector. Due to the $O(R^3)$ cubic computational complexity for Hessian matrix inverse operations, ALS is unscalable with the rank R. CCD neglects the relations between elements within a feature vector. Then, CCD has $O(R)$ complexity. However, CCD has a slow convergence rate [11]. Stochastic Gradient Descent (SGD) is a simple optimization method. SGD can transform the whole needed parameters of the GD into the part and randomized choice [39–41]. However, the over-writing problems of SGD on MF and TF make it hard to parallelize [39–41]. Hogwild! is a parallelization method for SGD [40], in which multi-threads select the non-overwriting points with convergence proof, and Hogwild! converges fast. However, the Hogwild! cannot make a promise of accuracy. Shin et al. [11] proposed a scalable algorithm for tensor factorization, which combines with the high convergence rate of ALS and fine-grained parallelization of CCD, which is to approximate the original Hessian matrix by a diagonal blocked matrix and the inverse operations save more cost. However, this method needs the inverse operations of the block matrix which cannot make a promise of linear scalability of the communication and computational overheads with the rank R. Chakaravarthy et al. [10] proposed distributed a distributed Tucker Decomposition method; however, the generalized styles are not involved. Since the entire factor matrix is involved in each iteration, these methods are not scalable for memory requirements. Smith et al. proposed medium-grained [8,9] and fine-grained parallelization methods [42], which need to consider the entire factor and Hessian matrices and the computational overhead of hyper-graph partitioning is expensive. Some works are proposed to accelerate 3-way DNTF for image processing [43]. However, for multi-way SNTF, there are few studies to effectively reduce the computation and memory overhead. GSSNTF is a fine-grained algorithm for generalized NTF. CUGSSNTF takes advantage of the powerful CUDA parallelization; meanwhile, the compressive method for sparse tensor data be used.

Table 1
Full name and acronym.

Full name	Acronym
Compressed Sparse Fiber	CSF
Stochastic Gradient Descent	SGD
Alternative Least Square	ALS
Cyclic Coordinate Descend	CCD
KullbackLeibler divergence	KL-divergence
Itakura-Saito divergence	IS-divergence
Matrix Factorization	MF
Non-negative Matrix Factorization	NMF
Tensor Factorization	TF
Non-negative Tensor Factorization	NTF
Dense Non-negative Tensor Factorization	DNTF
Sparse Non-negative Tensor Factorization	SNTF
Single-thread-based SNTF	SSNTF
Generalized SSNTF	GSSNTF
Compute Unified Device Architecture	CUDA
Graphics Processing Unit	GPU
CUDA parallelizing GSSNTF	CUGSSNTF
Multi-GPU CUGSSNTF	MCUGSSNTF

3. Problem formulation

Some related notations and definitions of NTF are presented in Sections 3.1 and 3.2, respectively. Then, the preliminary of NTF is illustrated in Section 3.3. Computing theory on GPU is discussed in Section 3.4.

3.1. Related notations

To make the notations more clear, the full names and acronyms are presented in Table 1 and the main notations are presented in Table 2.

3.2. Definition

Definition 1 (Tensor Approximation)**.** Fig. 1 shows that a N-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1\times,\ldots,\times I_N}$ can be approximated by $\widehat{\boldsymbol{\mathcal{X}}} \in \mathbb{R}^{I_1\times,\ldots,\times I_N}$, as well as a N-order residual tensor $\boldsymbol{\mathcal{E}} \in \mathbb{R}^{I_1\times,\ldots,\times I_N}$. The low-rank approximation problem is defined as

$$\mathcal{X} = \widehat{\mathcal{X}} + \mathcal{E}, \tag{1}$$

where $\widehat{\boldsymbol{\mathcal{X}}}$ is denoted by a low-rank tensor.

Definition 2 (Tensor Factorization)**.** The low-rank tensor $\widehat{\boldsymbol{\mathcal{X}}}$ can be obtained via CP decomposition and the decomposition is represented as

$$\underset{\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}}{\arg\min}\, d = \mathcal{D}(\boldsymbol{\mathcal{X}} \| \widehat{\boldsymbol{\mathcal{X}}}), \tag{2}$$

where $\widehat{\boldsymbol{\mathcal{X}}} = \sum_{r=1}^{R} \bar{a}_r^{(1)} \circ \cdots \circ \bar{a}_r^{(N)}$, and the nth mode matrix unfolding format is denoted by $\widehat{\mathbf{X}}^{(n)} = \mathbf{A}^{(n)}(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)})^T \in \mathbb{R}_+^{I_n \times I_1 \cdots I_{n-1} \cdot I_{n+1} \cdots I_N}$, and $\widehat{x}_{i_n,j}^{(n)} = \sum_{r=1}^{R} a_{i_n,r}^{(n)} \cdot a_{i_1,r}^{(1)} \cdots a_{i_{n-1},r}^{(n-1)} \cdot a_{i_{n+1},r}^{(n+1)} \cdots a_{i_N,r}^{(N)}$. $\mathbf{X}_{(n)}$ contains the tensor element $x_{i_n,j}$ at the position in the unfolding matrix (The detail for tensor unfolding is presented in Fig. 2) of $\boldsymbol{\mathcal{X}}$ and the row index i_n and column index j of the $\boldsymbol{\mathcal{X}}$ is given by $j = 1 + \sum_{k=1,n\neq k}^{N}[(i_k - 1)\prod_{m=1,m\neq n}^{k-1} I_m]$. More details about TF are shown in Fig. 3.

Definition 3 (Sparse Tensor Approximation)**.** Given N entity sets $\{I_1, \ldots, I_N\}$, $\boldsymbol{\mathcal{X}}$ is a tensor whose element $x_{i_1,\ldots,i_N}$ describes certain relationship between N entities $\{i_1 \leq I_1, \ldots, i_N \leq I_N\}$. Let Ω and Ξ be the known and unknown elements sets, where $|\Omega| \ll |\Xi|$. The problem of sparse tensor approximation is to obtain $\widehat{\boldsymbol{\mathcal{X}}}$, and $\widehat{\boldsymbol{\mathcal{X}}}$ can approximate the unknown elements in $\boldsymbol{\mathcal{X}}$.

Table 2
Table of symbols.

Symbol	Definition
$\mathcal{X}$	input tensor ($\in \mathbb{R}_{+}^{I_1 \times \cdots \times I_N}$, I_n, $n \in \{1, \ldots, N\}$)
$\widehat{\mathcal{X}}$	the approximation tensor ($\in \mathbb{R}_{+}^{I_1 \times \cdots \times I_N}$, I_n, $n \in \{1, \ldots, N\}$)
$\mathbf{X}^{(n)}$	the nth mode tensor unfolding matrix format $\mathbb{R}_{+}^{I_n \times I_1 \cdots I_{n-1} I_{n+1} \cdots I_N}$)
$\widehat{\mathbf{X}}^{(n)}$	the nth mode tensor unfolding matrix format $\mathbb{R}_{+}^{I_n \times I_1 \cdots I_{n-1} I_{n+1} \cdots I_N}$) for approximation tensor
$\mathbf{A}$	factor matrix ($\in \mathbb{R}^{I_n \times R}$)
$\mathbf{A}^{(k)}$	kth factor matrix
a	vector
T	matrix transform operation
$\odot$	Khatri-Rao product where $\mathbf{C} = \mathbf{A} \odot \mathbf{B} \in \mathbb{R}^{I_1 I_2 \times J}$, $c_{(i_1-1)I_2+i_2,j} = a_{i_1,j} b_{i_2,j}$
$''\circ''$	vectors outer product
$''\,.\,''$	element-wise (Hadamard) product
$\circ/(-,/)$	element-wise multiplication/element-wise division
$\overline{a}_{:,i_n}$	the i_nth column of a matrix $\mathbf{A}$
$a_{i_1,:}$	the i_1th row of a matrix $\mathbf{A}$
a_i	the ith entry of a vector a
Ω	the set of indices of observable entries in a sparse tensor $\mathcal{X}$.
$\Omega_{i_n}^{(n)}$	the entry indexes subset of nth mode.

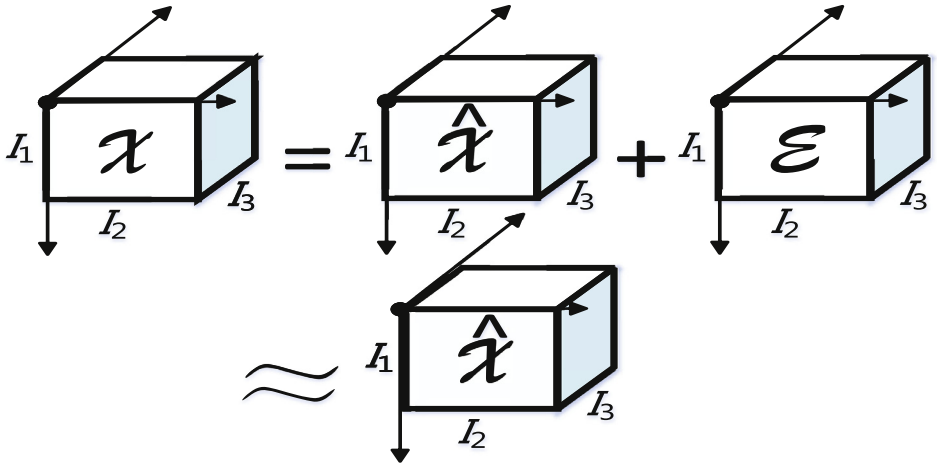


Fig. 1. Tensor approximation with 3-way tensor

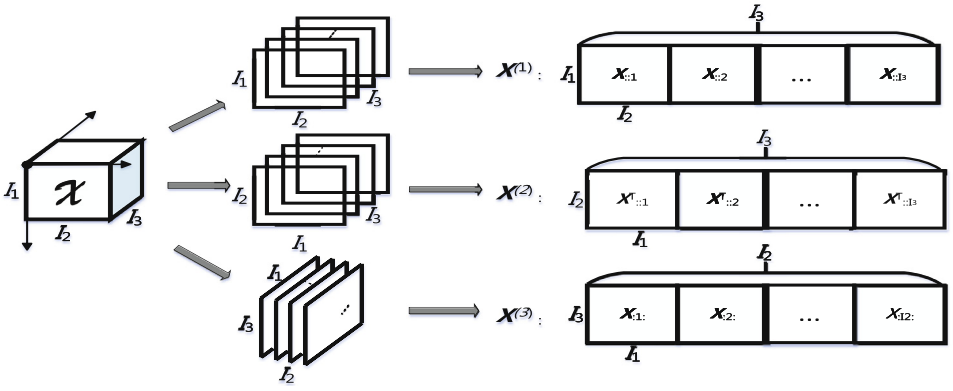


Fig. 2. Tensor unfolding.

Definition 4 (Sparse Non-negative Tensor Factorization)**.** If the low rank tensor has non-negative constraint, the tensor $\widehat{\mathcal{X}}$ is obtained by

$$\arg\min_{\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}} d = \mathcal{D}\left(\mathcal{P}_\Omega(\mathcal{X}) \| \mathcal{P}_\Omega(\widehat{\mathcal{X}})\right),$$
$$s.t. \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \geq 0,$$

where $\mathcal{P}_\Omega$ is the projection operator on the index set Ω. The non-negative $\widehat{\mathcal{X}}$ can be formalized via $\widehat{\mathcal{X}} = \sum_{r=1}^{R} \overline{a}_r^{(1)} \circ \cdots \circ \overline{a}_r^{(N)}$ where $R \ll I_1, \ldots, I_N$ with N non-negative factor matrices limitation, $\mathbf{A}^{(1)} = [\overline{a}_1^{(1)}, \ldots, \overline{a}_r^{(1)}, \ldots, \overline{a}_R^{(1)}] \in \mathbb{R}_{+}^{I_1 \times R}, \ldots, \mathbf{A}^{(N)} = [\overline{a}_1^{(N)}, \ldots, \overline{a}_r^{(N)}, \ldots, \overline{a}_R^{(N)}] \in \mathbb{R}_{+}^{I_N \times R}$.

3.3. Preliminary

A probabilistic interpretation of NTF is to take $x_{i_1,\ldots,i_N}$ as an observation from a distribution view. In the following, we briefly review the likelihood maximization problems of the three

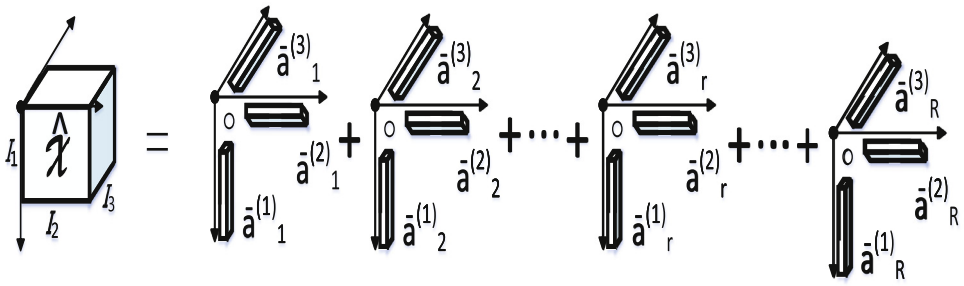


Fig. 3. Low rank tensor factorization with 3 modes tensor.

most popular probabilistic distribution, i.e., $\mathcal{G}aussian$, $\mathcal{P}oisson$, and $\mathcal{E}xponential$. When we take $x_{i_1,\ldots,i_N} \sim \mathcal{G}aussian(\widehat{x}_{i_1,\ldots,i_N}, \sigma^2)$, $x_{i_1,\ldots,i_N} \sim \mathcal{P}oisson(\widehat{x}_{i_1,\ldots,i_N})$, and $x_{i_1,\ldots,i_N} \sim \mathcal{E}xponential(\widehat{x}_{i_1,\ldots,i_N})$ maximizing the likelihood of observing $\mathcal{X}$ and $\{\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}\}$ [44–46], and the three likelihood maximum problems become the followings

$$\begin{cases} \underset{\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}}{\arg\min}\, \mathcal{D}_{Eu}(\mathcal{X}\|\widehat{\mathcal{X}}) = \|\mathcal{X}-\widehat{\mathcal{X}}\|^2; \\ \underset{\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}}{\arg\min}\, \mathcal{D}_{KL}(\mathcal{X}\|\widehat{\mathcal{X}}) = \sum\left(\widehat{\mathcal{X}} - \mathcal{X}.\log(\widehat{\mathcal{X}})\right); \\ \underset{\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}}{\arg\min}\, \mathcal{D}_{IS}(\mathcal{X}\|\widehat{\mathcal{X}}) = \sum\left(\frac{\mathcal{X}}{\widehat{\mathcal{X}}} + \log(\widehat{\mathcal{X}})\right), \end{cases} \tag{3}$$

respectively, with non-negativity constraints $\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)} \geq 0$. The $\mathcal{D}_{Eu}$, $\mathcal{D}_{KL}$, and $\mathcal{D}_{IS}$ are the Euclidean distance, KL-divergence, and IS-divergence, respectively. The low-rank tensor $\widehat{\mathcal{X}}$ comprises of the Khatri–Rao product of N low-rank factor matrices. Thus, the optimization problems in (3) are not convex for the N low-rank factor matrices. If a factor matrix is treated as a list of variables, and the others are fixed, the original optimization problems in (3) can be transformed into a convex optimization.

The update rules are given as follows

$$\begin{cases} a^{(n)}_{i_n,r} \leftarrow a^{(n)}_{i_n,r} \dfrac{\left(\mathbf{X}^{(n)}\mathbf{S}^{(n)}\right)_{i_n,r}}{\left(\mathbf{A}^{(n)}\mathbf{S}^{(n),T}\mathbf{S}^{(n)}\right)_{i_n,r}}; \\ a^{(n)}_{i_n,r} \leftarrow a^{(n)}_{i_n,r} \dfrac{\left(\dfrac{\mathbf{X}^{(n)}}{\mathbf{A}^{(n)}\mathbf{S}^{(n),T}}\mathbf{S}^{(n)}\right)_{i_n,r}}{\left(\mathbf{E}^{(n)}\mathbf{S}^{(n)}\right)_{i_n,r}}; \\ a^{(n)}_{i_n,r} \leftarrow a^{(n)}_{i_n,r} \dfrac{\left(\dfrac{\mathbf{X}^{(n)}}{\mathbf{A}^{(n)}\mathbf{S}^{(n),T}}\mathbf{S}^{(n)}\right)_{i_n,r}}{\left(\mathbf{E}^{(n)}\mathbf{S}^{(n)}\right)_{i_n,r}}, \end{cases} \tag{4}$$

respectively, where $\mathbf{E}^{(n)} \in \mathbb{R}_+^{I_n \times I_1 \cdots I_{n-1} \cdot I_{n+1} \cdots I_N}$, $E^{(n)}_{i_n,j} = 1$, and $\widehat{\mathbf{X}}^{(n)} = \mathbf{A}^{(n)}(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)})^T$, and $\mathbf{S}^{(n)} = \mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)}$. $\{\mathbf{E}^{(n)}, \mathbf{X}^{(n)}, \widehat{\mathbf{X}}^{(n)}\}$ are the nth mode matricized tensor and $\mathbf{S}^{(n)}$ is the nth Khatri–Rao product. $\{\mathbf{E}^{(n)}\mathbf{S}^{(n)}, \mathbf{X}^{(n)}\mathbf{S}^{(n)}, \widehat{\mathbf{X}}^{(n)}\mathbf{S}^{(n)}\}$ are the MTTKRP.

Fig. 4 illustrates the computational process of MTTKPP. A small example is shown to make more illustrations. We take

$$\mathbf{X}^{(1)} = \begin{bmatrix} 3 & 2 & 4 & 3 & 4 & 5 \\ 3 & 5 & 3 & 2 & 2 & 1 \\ 2 & 1 & 4 & 2 & 1 & 2 \\ 3 & 2 & 3 & 4 & 5 & 3 \end{bmatrix}, \mathbf{A}^{(2)} = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix},$$

$$\mathbf{A}^{(3)} = \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}.$$

The Khatri–Rao product matrix and MTTKRP are

$$\mathbf{A}^{(3)} \odot \mathbf{A}^{(2)} = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \\ 2 & 2 \\ 4 & 4 \\ 6 & 6 \end{bmatrix}, \mathbf{X}^{(1)}(\mathbf{A}^{(3)} \odot \mathbf{A}^{(2)}) = \begin{bmatrix} 71 & 71 \\ 40 & 40 \\ 36 & 36 \\ 80 & 80 \end{bmatrix},$$

respectively.

The difference between MF and TF is presented in Fig. 5. As shown in Fig. 5, MF can be approximated by two low-rank matrices manipulation, and 3-way tensor can be approximated by three low-rank matrices manipulations. Meanwhile, TF involves the Khatri - Rao product of $N-1$ factor matrices and is equivalent to MF when N=2. We observe that the compression effect of TF about $\frac{(\sum_{r=1}^{R} I_r)R}{\prod_{r=1}^{R} I_r}$ is much better than MF about $\frac{(\sum_{r=1}^{2} I_r)R}{\prod_{r=1}^{2} I_r}$. Thus, due to more related parameters for TF, the update rule and online learning for incremental data between MF and TF have a huge distinction to train the factor matrices. Meanwhile, after introducing the intermediate matrix of the Khatri-Rao product results $\mathbf{S}^{(n)}$, we observe that the update rules in (4) present a little similarity in a form with NMF; however, there are several major differences as following:

1. NMF only involves two factor matrices and matrix product; however, NTF with N modes has Khatri–Rao product for $N-1$ factor matrices and matrix product;
2. NTF needs more time and space complexities to building the intermediate matrix $\mathbf{S}^{(n)}$ than NMF;
3. NTF can represent multi-way data; however, NMF can only represent two-way data.

The difference between sparse TF and dense TF is that the nonzero data determine the involved feature elements. Thus, the update rule of each feature vector within a feature matrix $A^{(n)}$ of dense TF shares the same Hession matrix. However, the update rule of each feature vector within a feature matrix $A^{(n)}$ of sparse TF cannot share the same Hession matrix. The difference is illustrated in Fig. 6. Based on the difference between dense TF and sparse TF, the update rule of SNTF is introduced. The update rules in (4) only involve dense matrices operations; however, in real industrial applications, the tensor data is very sparse. Thus, the update rules in (4) cannot be applied in real low-rank representation for industrial data. Thus, the update rules in (4) cannot made available for generalized SNTF without appropriate adjustments. Consequently, the update rules in (4) should be revised. The state-of-the-art solvents for SNTF with appropriate adjustments of DNTF are presented as follows:

1. A solvent is proposed by for image processing and computer vision [43–45]. The time complexity of each intermediate matrix $\mathbf{S}^{(n)}$ is $O(|\Omega|(N-1)R)$, and the space require-

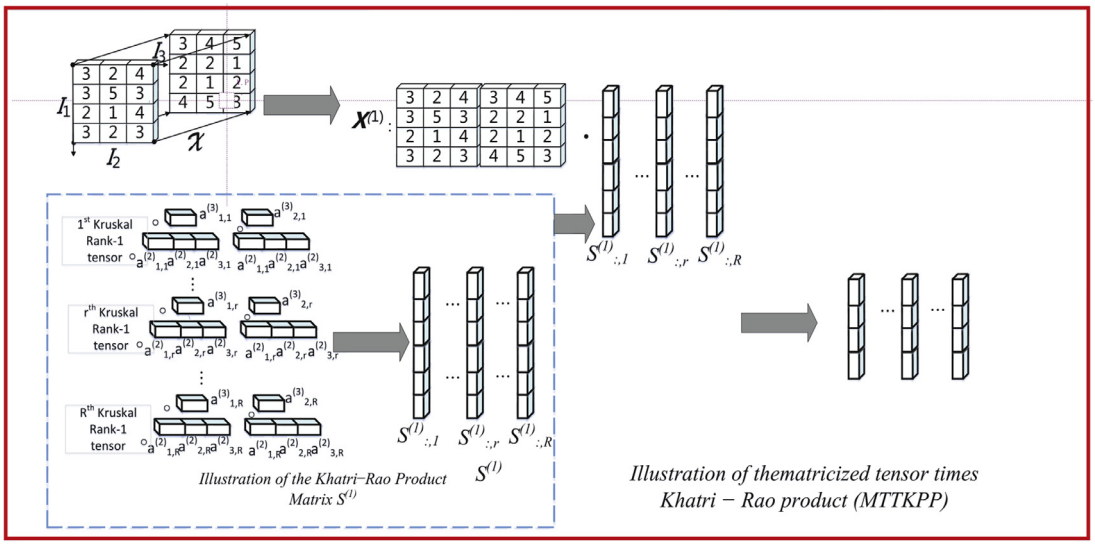


Fig. 4. Illustration of matricized tensor times Khatri–Rao Poduct (MTTKRP).

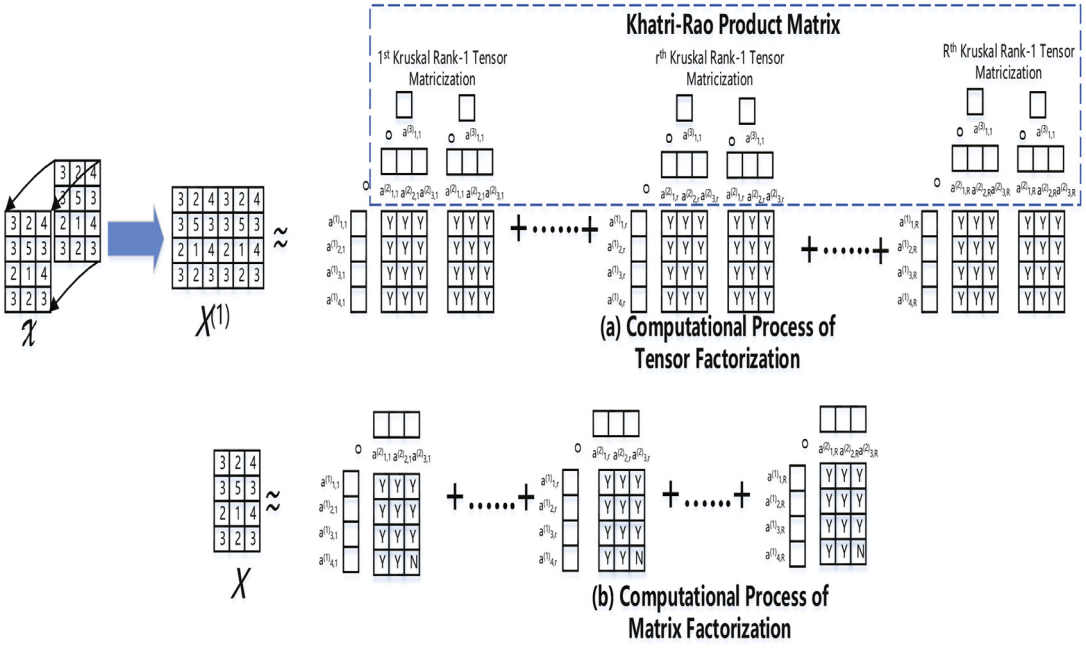


Fig. 5. Illustration of the difference between matrix factorization and tensor factorization.

ment is $O(\sum_{n=1}^{N} R\prod_{k\neq n} I_k)$. For these SNTF models, the time complexity of each intermediate matrix $\widehat{\mathbf{X}}^{(n)} = \mathbf{A}^{(n)}\mathbf{S}^{(n),T}$ is $O(|\Omega|NR)$, and the space requirement is $O(\prod_{n=1}^{N} I_n)$; the time complexity of each intermediate matrix $\mathbf{P}^{(n)} = \{\mathbf{X}^{(n)} / \mathbf{A}^{(n)}\mathbf{S}^{(n),T}, \mathbf{X}^{(n)}/(\mathbf{A}^{(n)}\mathbf{S}^{(n),T})^2, 1/\mathbf{A}^{(n)}\mathbf{S}^{(n),T}\}$ is $O(|\Omega|)$; the time complexity of MTTKRP for each intermediate matrix $\mathbf{Q}^{(n)} = \{\mathbf{X}^{(n)}\mathbf{S}^{(n)}, \widehat{\mathbf{X}}^{(n)}\mathbf{S}^{(n)}, \mathbf{P}^{(n)}\mathbf{S}^{(n)}\}$ is $O(|\Omega|R)$. Thus, the time complexity and space requirement for $\mathcal{D}_{Eu}$, $\mathcal{D}_{KL}$, and $\mathcal{D}_{IS}$ to a full SNTF are intolerable.

2. The other solvent [47] for $\mathbf{X}^{(n)}\mathbf{S}^{(n),T}\mathbf{S}^{(n)}$ is proposed for (3) to the update rules in (4). The update rules of DNTF refer to all row indexes of $\mathbf{A}^{(n)}$, and $\mathbf{G}_{i_n}^{(n)} = \mathbf{S}^{(n),T}\mathbf{S}^{(n)}$ is shared by $\{a_{i_n,:}^{(n)} | i_n \in \{1,\ldots,I_n\}\}$. However, the composite of $\mathbf{S}^{(n)}$ and $\mathbf{S}^{(n),T}\mathbf{S}^{(n)}$ in update rules (4) cannot follow the sparsity pattern of $\mathbf{X}^{(n)}$. Thus, the update rules (4) should be adjusted. The time complexity to building each Gram matrix $\mathbf{G}_{i_n}^{(n)} = \mathbf{S}^{(n),T}\mathbf{S}^{(n)}$ is $O(|\Omega_{i_n}^{(n)}|R^2)$, and $O(|\Omega|NR^2)$ for a full SNTF iteration. $\{(a_{i_n,:}^{(n)}\mathbf{G}_{i_n}^{(n)})|i_n \in \{1,\ldots,I_n\}\}$ need $O(I_nR^2)$. The time complexity of element division and multiplication operations are $O(2I_nR)$, and the space requirement for intermediate matrices of nominator and denominator is $O(2I_nR^2)$. The time complexity and space requirement of a full SNTF are linear with $O(R^2)$, which are not scalable linearly.

SSNTF[16] can solve the problem of non-linear time and space overhead that due to large-scale intermediate data. However, SSNTF concentrates mainly on the work of Euclidean distance, and it is not applied to the KL-divergence and IS-divergence. SSNTF is not a generalized solvent for that problem. Thus, a generalized single-thread-based model GSSNTF is proposed to solve the GPU computing obstacles on large-scale industrial data. At the same time, our method benefits from online processing.

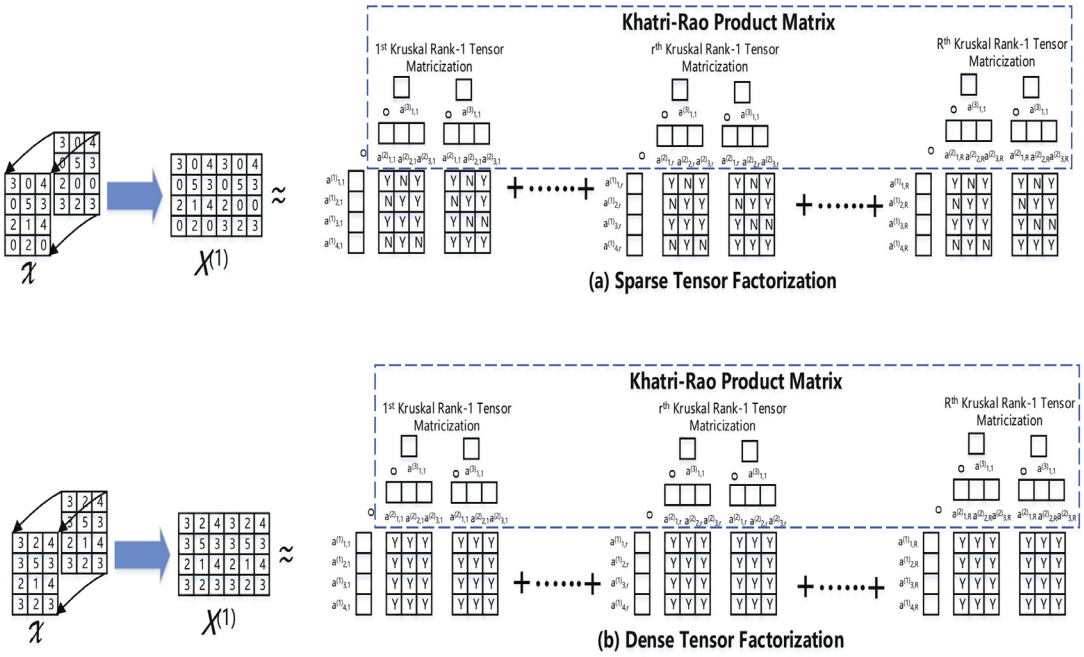


Fig. 6. Illustration of the Difference between the sparse TF and dense TF

3.4. GPU based computing

GPU is a highly efficient processor for data-intensive tasks, e.g., matrix operations, image processing, and graph mining, etc. For large-scale sparse tensor computing, there are the following superiorities: (1) Hardware: inside a GPU, each Stream Multi-processor (SM) which concludes several Stream Processor (SP) can access the global memory via highway bus. The faster the access speed on global memory a GPU has, the faster running speed the GPU has; (2) Programming: *kernels*, *thread*, and *thread block* are the basic terms in CUDA programming. *kernels* are functions that are executed on GPU. A kernel function is executed by a batch of threads. These threads are organized into blocks of threads to form grids, and the blocks are mapped to SMs. In a thread block, 32 threads are organized into a group to form a warp to execute the same instructions on the processor synchronously. Furthermore, CUDA provides abundant libraries, which help the users to perform Matrix Operations, Fast Fourier Transform (FFT), etc, fluently. Meanwhile, in CUDA programming communities, many pioneers have optimized the common approaches, e.g., vector product, Sparse Matrix and Vector Multiplication (SPMV), etc; (3) Communication and load balance: when a GPU cannot provide enough space for data, a multi-GPU system is a natural choice. The most important thing for sparse tensor computing is how to divide parallelization tasks in a load-balanced way, and map those tasks to threads and thread blocks in kernel in a fine-grained way. Multi-GPU system can simultaneously perform data transmission between the *devices* and *hosts*, and it provides synchronization instructions.

4. A generalized single-thread-based model, CUGSSNTF and MCUGSSNTF, and online model

The GSSNTF is presented in this section, and the needed elements follow the sparsity model of sparse tensor $\mathcal{X}$. The update rules of GSSNTF model with L_2 norm regularization are shown in Sections 4.1. Algorithm is analysed in Section 4.2. In Section 4.3, CUGSSNTF and MCUGSSNTF are presented. At last, in Section 4.4, the online learning model is illustrated, which includes an online learning algorithm and data merging.

4.1. Generalized single-thread-based model

The probabilistic distribution priors, i.e,. $\mathcal{G}aussian$, $\mathcal{P}oisson$, and $\mathcal{E}xponential$, are common assumptions in machine learning communities. We present the update rules based these three probabilistic distribution without L_2 norm regularization in Section 4.1.1, the L_2 norm regularized model in Section 4.1.2. Meanwhile, we represent the generalized update rules in Section 4.1.3.

4.1.1. Without L_2 norm regularization

The distance measurement tools, i.e., $\mathcal{D}_{Eu}$, $\mathcal{D}_{KL}$ and $\mathcal{D}_{IS}$, for the original $\boldsymbol{\mathcal{X}}$ and the approximation $\widehat{\boldsymbol{\mathcal{X}}}$ with non-negative constraints, are given as

$$\begin{cases} \underset{\mathbf{A}^{(n)},n\in\{1,\ldots,N\}}{\arg\min} \; d_{Eu} = \|\mathcal{P}_\Omega(\boldsymbol{\mathcal{X}}) - \mathcal{P}_\Omega(\widehat{\boldsymbol{\mathcal{X}}})\|^2; \\ \underset{\mathbf{A}^{(n)},n\in\{1,\ldots,N\}}{\arg\min} \; d_{KL} = \sum\limits_{\Omega}\left(\mathcal{P}_\Omega(\widehat{\boldsymbol{\mathcal{X}}}) - \mathcal{P}_\Omega(\boldsymbol{\mathcal{X}}).\log\left(\mathcal{P}_\Omega(\widehat{\boldsymbol{\mathcal{X}}})\right)\right); \\ \underset{\mathbf{A}^{(n)},n\in\{1,\ldots,N\}}{\arg\min} \; d_{IS} = \sum\limits_{\Omega}\left(\frac{\mathcal{P}_\Omega(\boldsymbol{\mathcal{X}})}{\mathcal{P}_\Omega(\widehat{\boldsymbol{\mathcal{X}}})} + \log\left(\mathcal{P}_\Omega(\widehat{\boldsymbol{\mathcal{X}}})\right)\right), \end{cases} \tag{5}$$

where $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)} \geq 0$. Supposing that $\mathbf{A}^{(n)}$ is fixed, all entries in Ω and their approximation are considered in (5). This problem can be split into multiple independent parts, i.e., $d = \sum_{i_n=1}^{I_n} d_{i_n}^{(n)}$, $I_n \in \{I_1, I_2, \ldots, I_N\}$, where $(d_{Eu}^{(n)})_{i_n} = \sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} (x_{i_n,j}^{(n)} - \widehat{x}_{i_n,j}^{(n)})^2$, $(d_{KL}^{(n)})_{i_n} = \sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} (\widehat{x}_{i_n,j}^{(n)} - x_{i_n,j}^{(n)} \log(\widehat{x}_{i_n,j}^{(n)}))$, and $(d_{IS}^{(n)})_{i_n} = \sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} (\frac{x_{i_n,j}^{(n)}}{\widehat{x}_{i_n,j}^{(n)}} + \log(\widehat{x}_{i_n,j}^{(n)}))$. The above decomposition analysis for the optimization problems (5) brings out the following truths: (1) the alternative optimization method which is that fixes $N-1$ factor matrices and optimizes a factor matrix can be decomposed into I_n parallel parts; (2) non-negative constrained optimization problems should be solved by determining the training step of gradient descent to maintain the monotonically decreasing and non-negative nature of the factor matrices. For $d_{i_n}^{(n)}$, its form of gradient descent is given

as

$$a_{i_n,r}^{(n)} \leftarrow a_{i_n,r}^{(n)} + \eta_{i_n,r}^{(n)}\left(-\frac{\partial d_{i_n}^{(n)}}{\partial a_{i_n,r}^{(n)}}\right), \tag{6}$$

where

$$\begin{cases} \dfrac{\partial (d_{Eu}^{(n)})_{i_n}}{\partial a_{i_n,r}^{(n)}} = \displaystyle\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \left((\widehat{x}_{i_n,j}^{(n)} - x_{i_n,j}^{(n)})\mathcal{S}_{i_n,j,r}^{(n)}\right); \\ \dfrac{\partial (d_{Eu}^{(n)})_{i_n}}{\partial a_{i_n,r}^{(n)}} = \displaystyle\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \left(\left(1 - \frac{x_{i_n,j}^{(n)}}{\widehat{x}_{i_n,j}^{(n)}}\right)\mathcal{S}_{i_n,j,r}^{(n)}\right); \\ \dfrac{\partial (d_{IS}^{(n)})_{i_n}}{\partial a_{i_n,r}^{(n)}} = \displaystyle\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \left(\left(\frac{1}{\widehat{x}_{i_n,j}^{(n)}} - \frac{x_{i_n,j}^{(n)}}{(\widehat{x}_{i_n,j}^{(n)})^2}\right)\mathcal{S}_{i_n,j,r}^{(n)}\right), \end{cases} \tag{7}$$

and

$$\mathcal{S}_{i_n,j,r}^{(n)} = \prod_{k\neq n, k\in\{1,2,\ldots,N\}} a_{i_k,r}^{(k)}, \tag{8}$$

$\widehat{x}_{i_n,j}^{(n)}$ and $x_{i_n,j}^{(n)}$ can be obtained from Definition 2. The three expressions that appear in all the formulas below are also obtained in this way.

Set

$$\begin{cases} (\eta_{Eu}^{(n)})_{i_n,r} = \dfrac{a_{i_n,r}^{(n)}}{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \widehat{x}_{i_n,j}^{(n)}\mathcal{S}_{i_n,j,r}^{(n)}}; \\ (\eta_{KL}^{(n)})_{i_n,r} = \dfrac{a_{i_n,r}^{(n)}}{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \mathcal{S}_{i_n,j,r}^{(n)}}; \\ (\eta_{IS}^{(n)})_{i_n,r} = \dfrac{a_{i_n,r}^{(n)}}{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \frac{1}{\widehat{x}_{i_n,j}^{(n)}}\mathcal{S}_{i_n,j,r}^{(n)}}, \end{cases} \tag{9}$$

where $\{\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}\}$ is initialized non-negatively. Since the negative item $\{-\widehat{x}_{i_n,j}^{(n)}\mathcal{S}_{i_n,j,r}^{(n)}, -\mathcal{S}_{i_n,j,r}^{(n)}, -\frac{1}{\widehat{x}_{i_n,j}^{(n)}}\mathcal{S}_{i_n,j,r}^{(n)}\}$ in $-\frac{\partial d_{i_n}^{(n)}}{\partial a_{i_n,r}^{(n)}}$ of update rules (12) can be eliminated, the update rules for optimization problems (5) are reformated into

$$\begin{cases} a_{i_n,r}^{(n)} \leftarrow a_{i_n,r}^{(n)} \dfrac{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} x_{i_n,j}^{(n)}\mathcal{S}_{i_n,j,r}^{(n)}}{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \widehat{x}_{i_n,j}^{(n)}\mathcal{S}_{i_n,j,r}^{(n)}}; \\ a_{i_n,r}^{(n)} \leftarrow a_{i_n,r}^{(n)} \dfrac{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \frac{x_{i_n,j}^{(n)}}{\widehat{x}_{i_n,j}^{(n)}}\mathcal{S}_{i_n,j,r}^{(n)}}{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \mathcal{S}_{i_n,j,r}^{(n)}}; \\ a_{i_n,r}^{(n)} \leftarrow a_{i_n,r}^{(n)} \dfrac{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \frac{x_{i_n,j}^{(n)}}{(\widehat{x}_{i_n,j}^{(n)})^2}\mathcal{S}_{i_n,j,r}^{(n)}}{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \frac{1}{\widehat{x}_{i_n,j}^{(n)}}\mathcal{S}_{i_n,j,r}^{(n)}}, \end{cases} \tag{10}$$

respectively. A single-thread-based model for GSSNTF can be obtained by the strategy with auto modified learning step, which can keep non-negativity with non-negative and original N factor matrices. L_2 norm regularization can make a more smoothed and accurate result and can avoid the over-fitting problem. Thus, in Section 4.1.2, the L_2 norm regularization method is presented.

4.1.2. L_2 Norm regularization

With L_2 norm regularization, the optimization problem for *Gaussian*, *Poisson*, and *Exponential* distribution styles can be written as

$$\begin{cases} \underset{\mathbf{A}^{(n)}, n\in\{1,\ldots,N\}}{\arg\min} \; d_{Eu} = \sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} (x_{i_n,j} - \widehat{x}_{i_n,j})^2 \\ \qquad\qquad + \sum_{n=1}^{N} (\lambda_{\mathbf{A}^{(n)}} \sum_{r=1}^{R} a_{i_n,r}^2); \\ \underset{\mathbf{A}^{(n)}, n\in\{1,\ldots,N\}}{\arg\min} \; d_{KL} = \sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \left(\widehat{x}_{i,j} - x_{i,j}\log(\widehat{x}_{i,j})\right) \\ \qquad\qquad + \sum_{n=1}^{N} (\lambda_{\mathbf{A}^{(n)}} \sum_{r=1}^{R} a_{i_n,r}^2); \\ \underset{\mathbf{A}^{(n)}, n\in\{1,\ldots,N\}}{\arg\min} \; d_{IS} = \sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \left(\frac{x_{i,j}}{\widehat{x}_{i,j}} + \log(\widehat{x}_{i,j})\right) \\ \qquad\qquad + \sum_{n=1}^{N} (\lambda_{\mathbf{A}^{(n)}} \sum_{r=1}^{R} a_{i_n,r}^2). \end{cases} \tag{11}$$

Applying GD on optimization objectives $\{d_{Eu}, d_{KL}, d_{IS}\}$ without constant 2 is given as

$$a_{i_n,r}^{(n)} \leftarrow a_{i_n,r}^{(n)} + \eta_{i_n,r}^{(n)}\left(-\frac{\partial d_{i_n}^{(n)}}{\partial a_{i_n,r}^{(n)}}\right), \tag{12}$$

where

$$\begin{cases} \dfrac{\partial (d_{Eu}^{(n)})_{i_n}}{\partial a_{i_n,r}^{(n)}} = \displaystyle\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} (\widehat{x}_{i_n,j}^{(n)} - x_{i_n,j}^{(n)})\mathcal{S}_{i_n,j,r}^{(n)} + \lambda_{\mathbf{A}^{(n)}} a_{i_n,r}^{(n)}; \\ \dfrac{\partial (d_{Eu}^{(n)})_{i_n}}{\partial a_{i_n,r}^{(n)}} = \displaystyle\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \left(1 - \frac{x_{i_n,j}^{(n)}}{\widehat{x}_{i_n,j}^{(n)}}\right)\mathcal{S}_{i_n,j,r}^{(n)} + \lambda_{\mathbf{A}^{(n)}} a_{i_n,r}^{(n)}; \\ \dfrac{\partial (d_{IS}^{(n)})_{i_n}}{\partial a_{i_n,r}^{(n)}} = \displaystyle\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \left(\frac{1}{\widehat{x}_{i_n,j}^{(n)}} - \frac{x_{i_n,j}^{(n)}}{(\widehat{x}_{i_n,j}^{(n)})^2}\right)\mathcal{S}_{i_n,j,r}^{(n)} + \lambda_{\mathbf{A}^{(n)}} a_{i_n,r}^{(n)}. \end{cases} \tag{13}$$

Set

$$\begin{cases} (\eta_{Eu}^{(n)})_{i_n,r} = \dfrac{a_{i_n,r}^{(n)}}{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \widehat{x}_{i_n,j}^{(n)}\mathcal{S}_{i_n,j,r}^{(n)} + \lambda_{\mathbf{A}^{(n)}} a_{i_n,r}^{(n)}}; \\ (\eta_{KL}^{(n)})_{i_n,r} = \dfrac{a_{i_n,r}^{(n)}}{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \mathcal{S}_{i_n,j,r}^{(n)} + \lambda_{\mathbf{A}^{(n)}} a_{i_n,r}^{(n)}}; \\ (\eta_{IS}^{(n)})_{i_n,r} = \dfrac{a_{i_n,r}^{(n)}}{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \frac{1}{\widehat{x}_{i_n,j}^{(n)}}\mathcal{S}_{i_n,j,r}^{(n)} + \lambda_{\mathbf{A}^{(n)}} a_{i_n,r}^{(n)}}, \end{cases} \tag{14}$$

where $\{\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}\}$ is initialized non-negatively. Since the negative item $\{-\widehat{x}_{i_n,j}^{(n)}\mathcal{S}_{i_n,j,r}^{(n)} - a_{i_n,r}^{(n)}, -\mathcal{S}_{i_n,j,r}^{(n)} - a_{i_n,r}^{(n)}, -\frac{1}{\widehat{x}_{i_n,j}^{(n)}}\mathcal{S}_{i_n,j,r}^{(n)} - a_{i_n,r}^{(n)}\}$ in $-\frac{\partial d_{i_n}^{(n)}}{\partial a_{i_n,r}^{(n)}}$ of update rules in (14) can be cancelled out, and the update rules for (11) are reformulated into

$$\begin{cases} a_{i_n,r}^{(n)} \leftarrow a_{i_n,r}^{(n)} \dfrac{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} x_{i_n,j}^{(n)}\mathcal{S}_{i_n,j,r}^{(n)}}{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \widehat{x}_{i_n,j}^{(n)}\mathcal{S}_{i_n,j,r}^{(n)} + \lambda_{\mathbf{A}^{(n)}} a_{i_n,r}^{(n)}}; \\ a_{i_n,r}^{(n)} \leftarrow a_{i_n,r}^{(n)} \dfrac{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \frac{x_{i_n,j}^{(n)}}{\widehat{x}_{i_n,j}^{(n)}}\mathcal{S}_{i_n,j,r}^{(n)}}{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \mathcal{S}_{i_n,j,r}^{(n)} + \lambda_{\mathbf{A}^{(n)}} a_{i_n,r}^{(n)}}; \\ a_{i_n,r}^{(n)} \leftarrow a_{i_n,r}^{(n)} \dfrac{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \frac{x_{i_n,j}^{(n)}}{(\widehat{x}_{i_n,j}^{(n)})^2}\mathcal{S}_{i_n,j,r}^{(n)}}{\sum_{(i_n,j)\in\Omega_{i_n}^{(n)}} \frac{1}{\widehat{x}_{i_n,j}^{(n)}}\mathcal{S}_{i_n,j,r}^{(n)} + \lambda_{\mathbf{A}^{(n)}} a_{i_n,r}^{(n)}}. \end{cases} \tag{15}$$

respectively. For weighted L_2 norm regularization, the update rules are presented as

$$\begin{cases} a^{(n)}_{i_n,r} \leftarrow a^{(n)}_{i_n,r} \dfrac{\sum_{(i_n,j)\in\Omega^{(n)}_{i_n}} x^{(n)}_{i_n,j}\mathcal{S}^{(n)}_{i_n,j,r}}{\sum_{(i_n,j)\in\Omega^{(n)}_{i_n}} \widehat{x}^{(n)}_{i_n,j}\mathcal{S}^{(n)}_{i_n,j,r} + \lambda_{\mathbf{A}^{(n)}}|\Omega^{(n)}_{i_n}|a^{(n)}_{i_n,r}}; \\ a^{(n)}_{i_n,r} \leftarrow a^{(n)}_{i_n,r} \dfrac{\sum_{(i_n,j)\in\Omega^{(n)}_{i_n}} \frac{x^{(n)}_{i_n,j}}{\widehat{x}^{(n)}_{i_n,j}}\mathcal{S}^{(n)}_{i_n,j,r}}{\sum_{(i_n,j)\in\Omega^{(n)}_{i_n}} \mathcal{S}^{(n)}_{i_n,j,r} + \lambda_{\mathbf{A}^{(n)}}|\Omega^{(n)}_{i_n}|a^{(n)}_{i_n,r}}; \\ a^{(n)}_{i_n,r} \leftarrow a^{(n)}_{i_n,r} \dfrac{\sum_{(i_n,j)\in\Omega^{(n)}_{i_n}} \frac{x^{(n)}_{i_n,j}}{(\widehat{x}^{(n)}_{i_n,j})^2}\mathcal{S}^{(n)}_{i_n,j,r}}{\sum_{(i_n,j)\in\Omega^{(n)}_{i_n}} \frac{1}{\widehat{x}^{(n)}_{i_n,j}}\mathcal{S}^{(n)}_{i_n,j,r} + \lambda_{\mathbf{A}^{(n)}}|\Omega^{(n)}_{i_n}|a^{(n)}_{i_n,r}}, \end{cases} \tag{16}$$

respectively.

4.1.3. Generalized update rules

Then, a generalized formation is presented to merge these update rules-based in a generalized way, which can drastically simplify the formulas and analysis. Let the intermediate parameters $\{p^{Eu}_{i_n,j,r}, p^{KL}_{i_n,j,r}, p^{IS}_{i_n,j,r}\}$ denote $\{x^{(n)}_{i_n,j}\mathcal{S}^{(n)}_{i_n,j,r}, \frac{x^{(n)}_{i_n,j}}{\widehat{x}^{(n)}_{i_n,j}}\mathcal{S}^{(n)}_{i_n,j,r}, \frac{x^{(n)}_{i_n,j}}{(\widehat{x}^{(n)}_{i_n,j})^2}\mathcal{S}^{(n)}_{i_n,j,r}\}$, respectively, and let the intermediate parameters $\{\overline{p}^{Eu}_{i_n,j,r}, \overline{p}^{KL}_{i_n,j,r}, \overline{p}^{IS}_{i_n,j,r}\}$ denote $\{\widehat{x}^{(n)}_{i_n,j}\mathcal{S}^{(n)}_{i_n,j,r}, \mathcal{S}^{(n)}_{i_n,j,r}, \frac{1}{\widehat{x}^{(n)}_{i_n,j}}\mathcal{S}^{(n)}_{i_n,j,r}\}$, respectively. Thus, the generalized forms for the update rules (15) and (16) are reformulated into

$$\begin{cases} a^{(n)}_{i_n,r} \leftarrow a^{(n)}_{i_n,r} \dfrac{\sum_{(i_n,j)\in\Omega^{(n)}_{i_n}} p_{i_n,j,r}}{\sum_{(i_n,j)\in\Omega^{(n)}_{i_n}} \overline{p}_{i_n,j,r} + \lambda_{\mathbf{A}^{(n)}}a^{(n)}_{i_n,r}}; \\ a^{(n)}_{i_n,r} \leftarrow a^{(n)}_{i_n,r} \dfrac{\sum_{(i_n,j)\in\Omega^{(n)}_{i_n}} p_{i_n,j,r}}{\sum_{(i_n,j)\in\Omega^{(n)}_{i_n}} \overline{p}_{i_n,j,r} + \lambda_{\mathbf{A}^{(n)}}|\Omega^{(n)}_{i_n}|a^{(n)}_{i_n,r}}, \end{cases} \tag{17}$$

respectively. Algorithm 1 describes a complete iteration to analyze the serial version of single-thread-based GSSNTF, which alternately updates $a^{(n)}_{i_n,:}$, $n \in \{1,\ldots,N\}$. The generalized parameters $\{\{p^{Eu}_{i_n,j,r}, p^{KL}_{i_n,j,r}, p^{IS}_{i_n,j,r}\}, \{\overline{p}^{Eu}_{i_n,j,r}, \overline{p}^{KL}_{i_n,j,r}, \overline{p}^{IS}_{i_n,j,r}\}\}$ of NTF are similar to that of NMF in [17]. We observe that NTF involves $N \geq 3$ factor matrices. However, the updated parameters of NMF only involves 2 factor matrices. Meanwhile, due to N-way data presentation of tensor, the form of the incremental data may present different forms, which can obtain the same conclusion with Section 3.3. In this work, we only present dynamic form in one way data of online learning for incremental data.

Algorithm 1: Serial Version of GSSNTF.

Input: factor matrices $\mathbf{A}^{(n)}$ of all n,$\mathcal{X}$,*Down*, *Up*
Output: $\mathbf{A}^{(n)}$ for all n

1 **for** *n from 1 to N* **do**
2 set $Down^{(n)} = 0$, $Up^{(n)} = 0$;
3 **for** i_n *from 1 to* I_n **do**
4 **for** $(i_n, j) \in \Omega^{(n)}_{i_n}$ **do**
5 **for** *r from 1 to R* **do**
6 $\mathcal{S}^{(n)}_{i_n,j,r} = \prod_{k\neq n} a^{(k)}_{i_k,r}$;
7 $\widehat{x}^{(n)}_{i_n,j} = \sum_{r=1}^{R} a^{(n)}_{i_n,r}\mathcal{S}^{(n)}_{i_n,j,r}$;
8 **for** *r from 1 to R* **do**
9 $up^{(n)}_{i_n,r} \leftarrow up^{(n)}_{i_n,r} + p_{i_n,j,r}$;
10 $down^{(n)}_{i_n,r} \leftarrow down^{(n)}_{i_n,r} + \overline{p}_{i_n,j,r}$
11 $down^{(n)}_{i_n,r} \leftarrow down^{(n)}_{i_n,r} + \lambda_{\mathbf{A}^{(n)}}a^{(n)}_{i_n,r} down^{(n)}_{i_n,r} \leftarrow down^{(n)}_{i_n,r} + \lambda_{\mathbf{A}^{(n)}}|\Omega^{(n)}_{i_n}|a^{(n)}_{i_n,r}$% Weight update rule
12 **for** *r from 1 to R* **do**
13 $a^{(n)}_{i_n,r} \leftarrow a^{(n)}_{i_n,r}(Up_{i_n,r}/Down_{i_n,r})$;

4.2. Theoretical analysis

The time complexity and space requirement of GSSNTF will be analyzed next.

Theorem 1 (Time complexity and space overhead of GSSNTF)**.** *The time complexity of (17), to a full GSSNTF are* $O(|\Omega|N^2R + 2|\Omega|NR + 2\sum_{n=1}^{N} I_nR)$, $O(|\Omega|N^2R + 2|\Omega|NR + 2|\Omega|N + 2\sum_{n=1}^{N} I_nR)$, *and* $O(|\Omega|N^2R + 2|\Omega|NR + 3|\Omega|N + 2\sum_{n=1}^{N} I_nR)$, *respectively, and the space requirement is* $O(3\sum_{n=1}^{N} I_nR)$, *on a full iteration.*

Theorem 2 (Space overhead of CUGSSNTF)**.** *The memory* overhead *of CUGSSNTF is* $O(\sum_{n=1}^{N} I_nR)$.

Normally, since $N \ll R$, the time complexity $O(|\Omega|N^2R + 2|\Omega|NR + 2\sum_{n=1}^{N} I_nR)$ of (17) is less than $O(|\Omega|(N^2R + NR^2) + \sum_{n=1}^{N} I_nR^2 + 2\sum_{n=1}^{N} I_nR)$, and the space requirement $O(3\sum_{n=1}^{N} I_nR)$ is less than $O(\prod_n I_n + \sum_{n=1}^{N} R\prod_{k\neq n} I_k + 3\sum_{n=1}^{N} I_nR)$, and $O(\sum_{n=1}^{N} I_nR^2 + 3\sum_{n=1}^{N} I_nR)$. GSSNTF has linear time complexity and space requirements relative to rank R.

4.3. CUDA parallelization approach

CUGSSNTF is presented in Section 4.3.1, then MCUGSSNTF is presented in Section 4.3.2.

4.3.1. Parallelization approach

As shown in Fig. 7, CSF has the hierarchy and fiber-centralization features; meanwhile, CSF can save memory cost and improve addressing efficiency for CUGSSNTF. Thus, CSF is an efficient sparse compressive format. In CUGSSNTF, a thread block can update $a^{(n)}_{i_n,:}$ by $\{(a^{(n)}_{i_n,:}, x^{(n)}_{i_n,j})|(i_1,\ldots,i_N) \in \Omega^{(n)}\}$, and the *fids* and *val* vectors store index $(i_1,\ldots,i_N)$ and $x^{(n)}_{i_n,j}$ in order, respectively. Thus, by sequentially accessing the element in *fptr*, *fids*, and *val* vectors, multiple thread blocks update the *n*th feature matrix, and a thread block updates a feature vector; meanwhile, a thread within the thread block updates a feature element within the feature vector (Lines 3 - 13, Algorithm 1). Supposing that there are *Tb* thread blocks, $a^{(n)}_{i_n,:}$, $i_n \in \{1,\ldots,I_n\}$ can be updated by the *Tb* thread blocks in parallel, and the thread block $mod(i_n, Tb)$ can update the$a^{(n)}_{i_n,:}$ without contention for resources or interlocks.

4.3.2. Multi-GPU parallelization

When a GPU cannot load a real dataset, the multi-GPU is an appropriate substitute choice. These GPUs make up a $(J_1 \times ,\ldots, \times J_N)$ array for processing the tiled $\mathcal{X}$ and its tiled corresponding factor matrix $\mathbf{A}^{(n)}$. The sub-tensor and factor submatrices $\{\mathcal{X}_{j_1,\ldots,j_N}, \mathbf{A}^{(1)}_{j_1},\ldots,\mathbf{A}^{(N)}_{j_N}\}$ is distributed to GPU $(j_1,\ldots,j_N)$, $j_n \in \{1,\ldots,J_n\}$, and $n \in \{1,\ldots,N\}$. The communication strategy is derived from the tiling strategy, and the communication group is defined by the alternative update rule of GSSNTF. Then, GPU (j_n, j) broadcasts local nominator and denominator to other GPUs, while updating its local $\mathbf{A}^{(n)}_{j_n,j}$. At last, GPU (j_n, j) broadcasts $\mathbf{A}^{(n)}_{j_n,j}$ to other GPUs within the same group. Fig. 8 shows a toy example, 8 GPUs (a $2 \times 2 \times 2$ GPU array) update $\mathbf{A}^{(1)}$. 8 GPUs are divided into two groups. GPUs {(1, 1, 1), (1, 2, 1), (1, 2, 2), (1, 1, 2)} update $\{\mathbf{A}^{(1)}_{1,1},$

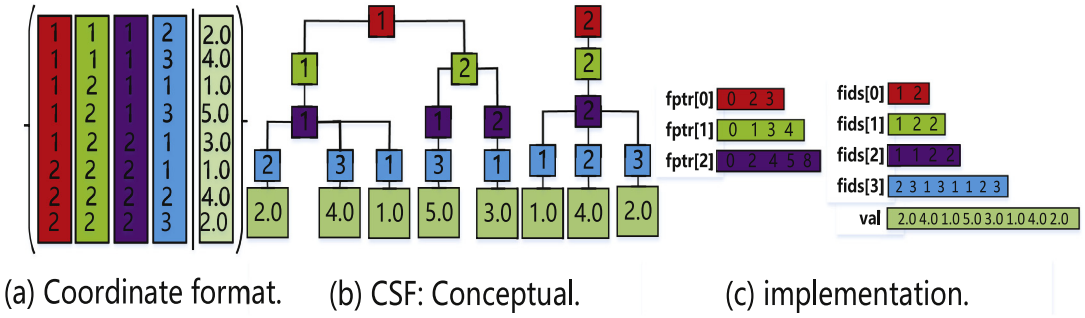


Fig. 7. Compressed Sparse Fiber.

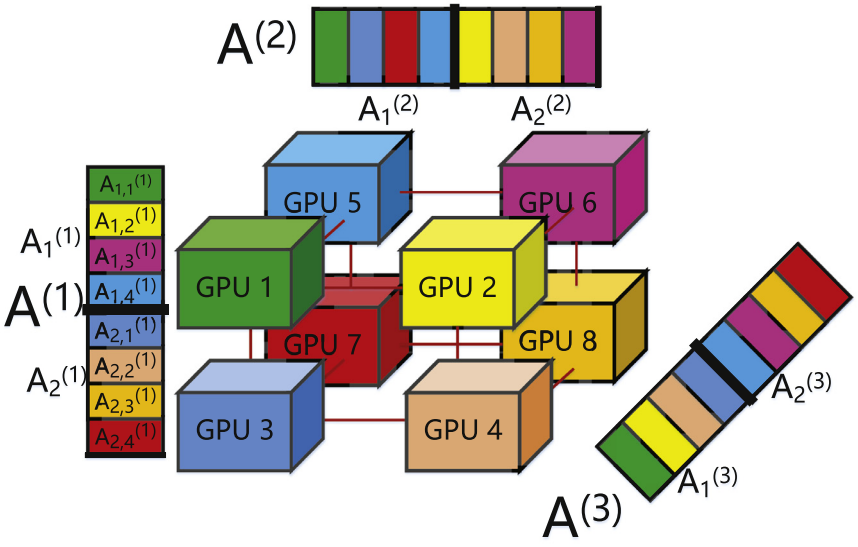


Fig. 8. Multi-GPU structure of MCUGSSNTF.

$\mathbf{A}_{1,2}^{(1)}$, $\mathbf{A}_{1,3}^{(1)}$,}, GPUs {(2, 1, 1), (2, 2, 1), (2, 2, 2), (2, 1, 2)} update { $\mathbf{A}_{2,1}^{(1)}$, $\mathbf{A}_{2,2}^{(1)}$, $\mathbf{A}_{2,3}^{(1)}$, }, The space overhead is $O(\frac{|\Omega|}{\prod_{n=1}^{N} J_n} + 6\sum_{n=1}^{N} \frac{I_n}{J_n} R)$ of a GPU, which consists of $\{\mathcal{X}_{j_1,\ldots,j_N}, Up_{j_n}^{(n)}, Down_{j_n}^{(n)}, \mathbf{A}_{j_n}^{(n)}\}$, and the cache area for $\{Up_{j_n}^{(n)}, Down_{j_n}^{(n)}, \mathbf{A}_{j_n}^{(n)}\}$, and the communication overhead is $O(3\sum_{n=1}^{N} \frac{I_n(\prod_{k\neq n} J_k - 1)}{J_n \prod_{k\neq n} J_k} R)$, which consists of the communication cost for { $Up_{j_n}^{(n)}$, $Down_{j_n}^{(n)}$, $\mathbf{A}_{j_n}^{(n)}$ }, $j_n \in \{1, \ldots, J_n\}$, and $n \in \{1, \ldots, N\}$. The space requirement and communication overhead scale with the rank R linearly.

4.4. Data merging and online learning

In this section, online learning for the incremental data processing is presented. Tree storage structure for sparse tensor has higher access efficiency than the common data structure styles, e.g., COO, etc. Furthermore, in large-scale data systems, online learning for GSSNTF should cooperate with the incremental building process of a tree storage structure for sparse tensor. Thus, an efficient online approach should include:

1. How to update the factor matrices in an online and real-time way and keep high accuracy? (Sections 4.4.1–4.4.3);
2. How to merge the newly arrived data into the built data structure? (Section 4.4.4).

Online learning is used to track the low-rank style of incremental data.

4.4.1. Online learning algorithm

The single-thread-based processing style gives the GSSNTF the ability of first-coming-first-computing. In this section, online learning for stream-like sparse tensor data is presented, which is derived from the single-thread-based processing style [17,48]. First, Supposing that the length of the Nth ($N \geq 3$) dimension grows over time, which follows the assumption in. Some notations about online learning are introduced. An online sparse tensor $\mathcal{X} \in \mathbb{R}_{+}^{I_1 \times \cdots \times I_N^{t_1+t_2}}$ comprises of $\mathcal{X}^{t_1} \in \mathbb{R}_{+}^{I_1 \times \cdots \times I_N^{t_1}}$ and $\mathcal{X}^{t_2} \in \mathbb{R}_{+}^{I_1 \times \cdots \times I_N^{t_2}}$, which represents the built tensor in t_1 time and new data in t_2 time, respectively. $\{\Omega_{i_N}^{(N),t_1}, \{\Omega_{i_n}^{(n),t_1} | n \in \{1, \ldots, N-1\}\}\}$ and $\{\Omega_{i_N}^{(N),t_2}, \{\Omega_{i_n}^{(n),t_2} | n \in \{1, \ldots, N-1\}\}\}$ are the corresponding index for $\mathcal{X}^{t_1} \in \mathbb{R}_{+}^{I_1 \times \cdots \times I_N^{t_1}}$ and $\mathcal{X}^{t_2} \in \mathbb{R}_{+}^{I_1 \times \cdots \times I_N^{t_2}}$, respectively. In real online systems, the data scale in t_2 is much smaller than the scale in t_1. Thus, we assume that $I_N^{t_2} \ll I_N^{t_1}$ and $|\Omega_{i_N}^{(N),t_2}| \ll |\Omega_{i_N}^{(N),t_1}|$. The online GSSNTF finds the temporal factor matrix $\mathbf{A}^{(N),t_2}$ for the time parameter t_2, which are based on the following data $\{\mathbf{X}^{(n),(t_2)}, \mathbf{A}^{(N),t_2}, \{\mathbf{A}^{(n),t_1} | n \in \{1, \ldots, N-1\}\}\}$. The update process for temporal factor vectors is obvious; Because the update process should consider the changes of the temporal data, the update process for the un-temporal factor vectors is more complex than

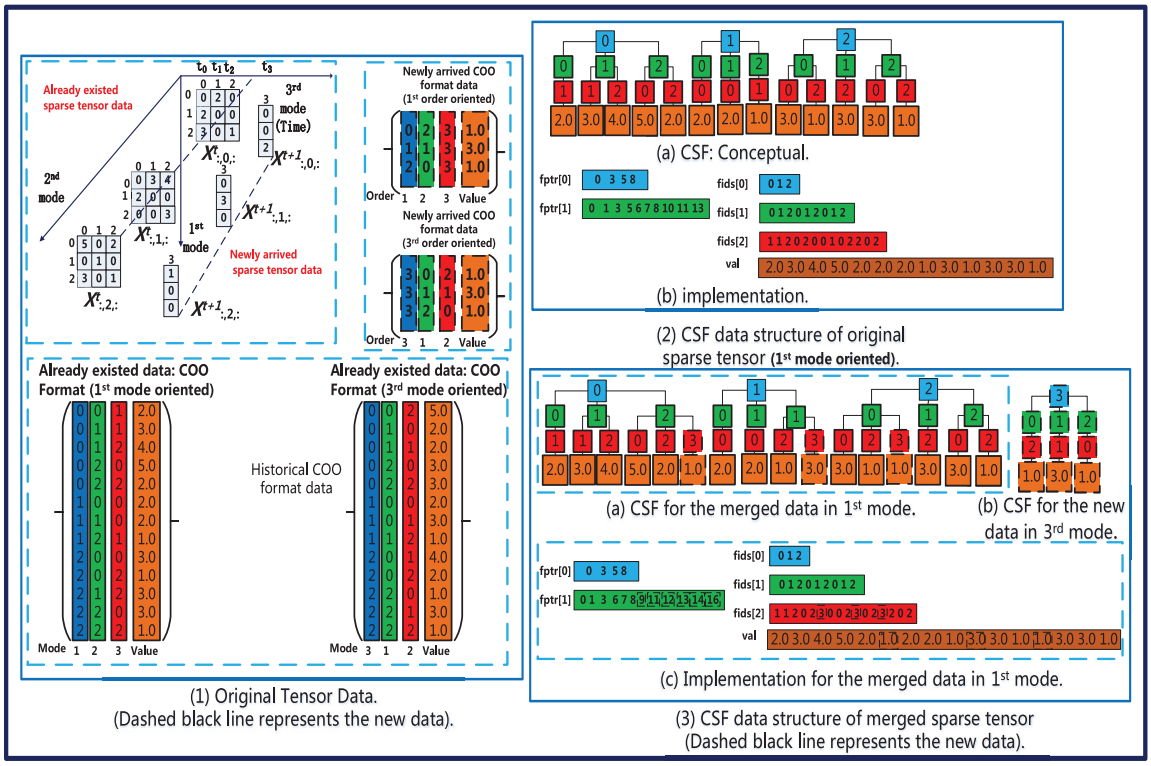


Fig. 9. Data merging process of OnCSF.

the temporal conditions. Thus, the approximation problem is give by $\mathcal{D}(\mathbf{X}^{(n)}\|\widehat{\mathbf{X}}^{(n)}) = \mathcal{D}(\mathbf{X}^{(n),t_1}\|\widehat{\mathbf{X}}^{(n),t_1}) + \eta\mathcal{D}(\mathbf{X}^{t_2}_{(n)}\|\widehat{\mathbf{X}}^{t_2}_{(n)})$, $n \in \{1,\ldots,N-1\}$, where η is a coefficient and it can represent the effect of the incoming elements at time t_2.

4.4.2. Update temporal factor matrix $\boldsymbol{A}^{(N)}$

By fixing the non-temporal factor matrices $\{\mathbf{A}^{(n)}|n \in \{1,\ldots,N-1\}\}$, if the divergence $\mathcal{D}(\mathbf{X}^{(n),t_1}\|\widehat{\mathbf{X}}^{(n),t_1})$ is minimized, then the factor matrices $\{\mathbf{A}^{(n)}|n \in \{1,\ldots,N\}\}$ are updated. Thus, the problem of updating $\mathbf{A}^{(N),t_2}$ is equivalent to minimize $\mathcal{D}(\mathbf{X}^{(N),t_2}\|\widehat{\mathbf{X}}^{(N),t_2})$. By appending the projection $\mathbf{A}^{(N),t_2}$ of $\mathbf{X}^{(n),t_2}$ via loading the factor matrices $\{\mathbf{A}^{(n),t_1}|n \in \{1,\ldots,N-1\}\}$ of the previous time step, $\mathbf{A}^{(N),t_2}$ is updated. The update rule for $\mathbf{A}^{(N),t_2}$ in generalized element-wise and weighted forms for online learning are given as

$$\begin{cases} a^{t_2}_{i_N,r} \leftarrow \dfrac{a^{t_2}_{i_N,r}\sum_{j\in\Omega^{(N),t_2}_{i_N}} p^{t_2}_{i_N,j,r}}{\sum_{j\in\Omega^{(N),t_2}_{i_N}} \overline{p}^{t_2}_{i_N,j,r} + \lambda_{\mathbf{A}^{(N)}} a^{t_2}_{i_N,r}}; \\ a^{t_2}_{i_N,r} \leftarrow \dfrac{a^{t_2}_{i_N,r}\sum_{j\in\Omega^{(N),t_2}_{i_N}} p^{t_2}_{i_N,j,r}}{\sum_{j\in\Omega^{(N),t_2}_{i_N}} \overline{p}^{t_2}_{i_N,j,r} + \lambda_{\mathbf{A}^{(N)}}|\Omega^{(N),t_2}_{i_N}|a^{t_2}_{i_N,r}}, \end{cases} \tag{18}$$

respectively.

4.4.3. Update non-temporal factor matrix $\{\mathbf{A}^{(n)}|n \in \{1,\ldots,N-1\}\}$

The online problem is rewrited as

$$\arg\min_{\mathbf{A}^{(n)}} \mathcal{D}\left(\mathbf{X}^{(n)}\|\widehat{\mathbf{X}}^{(n)}\right) = \mathcal{D}\left(\mathbf{X}^{(n),t_1}\|\widehat{\mathbf{X}}^{(n),t_1}\right) + \eta\mathcal{D}\left(\mathbf{X}^{(n),t_2}\|\widehat{\mathbf{X}}^{(n),t_2}\right). \tag{19}$$

Applying GD to minimize the error $\overline{d}_j(h_j)$ can be written as

$$a^{t_1}_{i_n,r} \leftarrow a^{t_1}_{i_n,r} + \gamma_{\mathbf{W}}\left(\frac{\partial\mathcal{D}\left(\mathbf{X}^{(n)}\|\widehat{\mathbf{X}}^{(n)}\right)}{\partial a^{t_1}_{i_n,r}} + \eta\frac{\partial\mathcal{D}\left(\mathbf{X}^{(n),t_2}\|\widehat{\mathbf{X}}^{(n),t_2}\right)}{\partial a^{t_1}_{i_n,r}}\right). \tag{20}$$

The update rules of non-temporal factor matrix $\left\{\mathbf{A}^{(n)}|n \in \{1,\ldots,N-1\}\right\}$ in element-wise and weighted form are given as

$$a^{t_1}_{i_n,r} \leftarrow \frac{a^{t_1}_{i_n,r}\left(\sum_{j\in\Omega^{(n),t_1}_{i_n}} p^{t_1}_{i_n,j,r} + \eta\sum_{j\in\Omega^{(n),t_2}_{i_n}} p^{t_2}_{i_n,j,r}\right)}{\left(\sum_{j\in\Omega^{(n),t_1}_{i_n}} \overline{p}^{t_1}_{i_n,j,r} + \eta\sum_{j\in\Omega^{(n),t_2}_{i_n}} \overline{p}^{t_2}_{i_n,j,r}\right) + \lambda_{\mathbf{A}^{(n)}} a^{t_2}_{i_n,r}}, \tag{21}$$

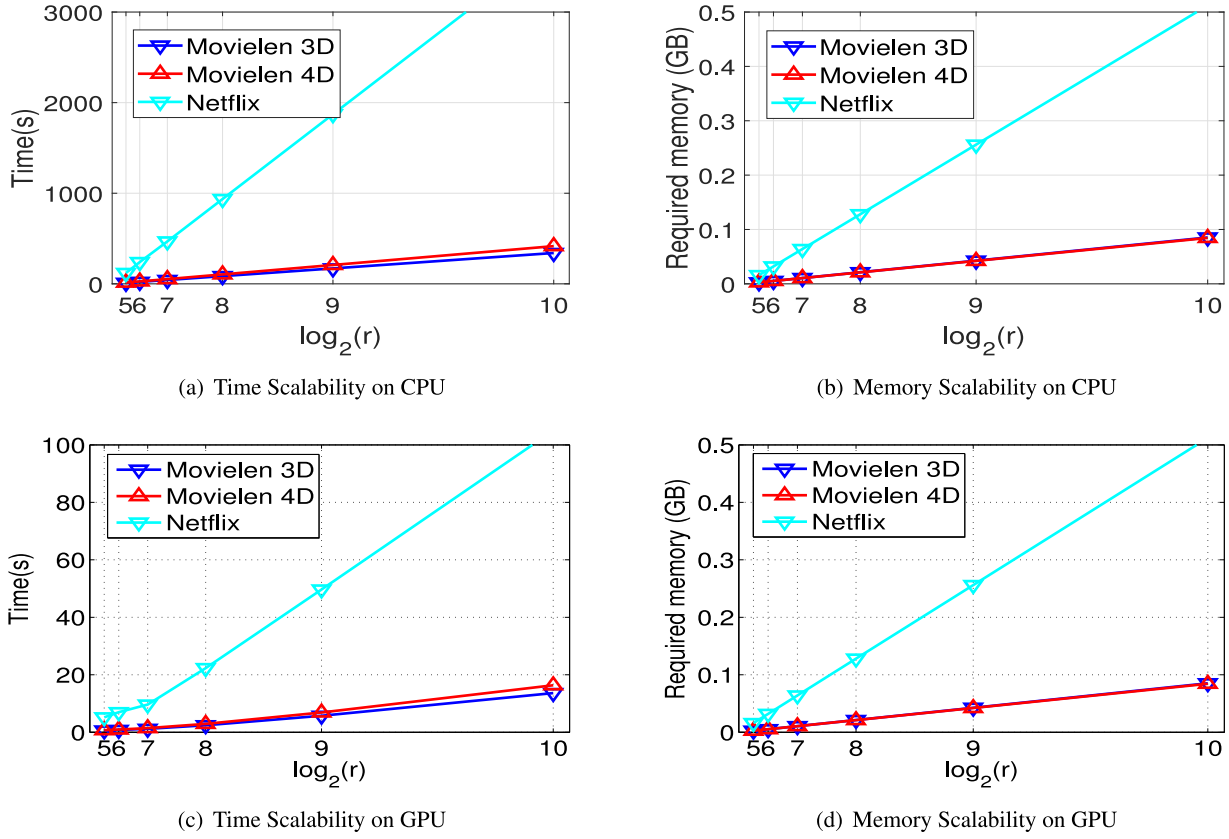
and

$$a^{t_1}_{i_n,r} \leftarrow \frac{a^{t_1}_{i_n,r}\left(\sum_{j\in\Omega^{(n),t_1}_{i_n}} p^{t_1}_{i_n,j,r} + \eta\sum_{j\in\Omega^{(n),t_2}_{i_n}} p^{t_2}_{i_n,j,r}\right)}{\left(\sum_{j\in\Omega^{(n),t_1}_{i_n}} \overline{p}^{t_1}_{i_n,j,r} + \eta\sum_{j\in\Omega^{(n),t_2}_{i_n}} \overline{p}^{t_2}_{i_n,j,r}\right) + \lambda_{\mathbf{A}^{(n)}}|\Omega^{(n),t_2}_{i_n}|a^{t_2}_{i_n,r}}, \tag{22}$$

respectively.

4.4.4. Data merging

CSF structure has more access efficiency than COO format; however, when the new data arrive in the system, the original CSF can hardly adopt the data immediately. OnCSF can process the stream-like data, and transform the online COO data into the built CSF data structure. In this section, the OnCSF is introduced, which includes the details about building the new CSF data structure in a real-time and online way. Fig. 9 shows the principle of OnCSF. As shown in Fig. 9, the 1th mode tensor unfolding matrix $\mathbf{X}^{(1),2}$ and



(a) Time Scalability on CPU

(b) Memory Scalability on CPU

(c) Time Scalability on GPU

(d) Memory Scalability on GPU

Fig. 10. Scalability and memory requirement of GSSNTF$_{Eu}$ are scalable with R on CPU (Fig. 10(a) and 10(b),time on second (s)). Scalability and memory requirement of CUGSSNTF$_{Eu}$ are scalable with R on GPU (Fig. 10(c) and 10(d), time on second (s)).

Table 3
The details for real data sets.

Data set	Movielens$_{3D}$	Movielens$_{4D}$	Netflix
I_1	71, 567	71, 567	480, 189
I_2	10, 681	10, 681	17, 770
I_3	1, 038	169	2, 150
I_4	/–	24	/–
♯*Train*	9, 301, 274	9, 301, 274	99, 072, 112
♯*Test*	698, 780	698, 780	1, 408, 395

$\mathbf{X}^{(1),3}$ follow the

$$\mathbf{X}^{(1),2} = \left[\begin{array}{cccc|cccc} 0 & 0 & 5 & 2 & 3 & 0 & 0 & 4 & 2 \\ 2 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 3 & 0 & 3 & 0 & 0 & 0 & 1 & 3 & 1 \end{array}\right],$$

$$\mathbf{X}^{(1),3} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 3 & 0 \\ 2 & 0 & 0 \end{bmatrix},$$

respectively. The Fig. 9 reveals the following 3 features of online learning:

1. As the Fig. 9 (2) and (3) show, the merged CSF data structure can only has small modification compared with original CSF data structure. The amount of newly arrived data is much less than the amount of the existed data. However, the time cost of rebuilding the CSF data structure cannot satisfy the real-time requirement;
2. Meanwhile, if only one mode is dynamic, e.g., time (t), the modification scope is limited in three arrays in 1th mode unfolding pattern, e.g., *fptr*[1], *fids*[2], and *val*[]. The pointer array *fptr*[1] should change the values in $\{fptr[1][6], \ldots, fptr[1][11]\}$. The identity array *fids*[2] should change the values { *fids*[2][5], *fids*[2][9], *fids*[2][12] }. The value array *val*[] should change the values {*val*[5], *val*[9], *val*[12]}. The time cost of OnCSF is much less the rebuilding a new CSF data structure;
3. In Fig. 9, we observe that the merging process do not be illustrated for the unfolded matrix of the sparse tensor in the 2nd and 3rd mode. We conclude that the reasons are that (1) the merging process styles of the CSF data structure for 2nd and 3rd mode sparse tensor are similar; (2) As shown in the Fig. 9 (3b),the OnCSF for the time mode (3rd mode) only involves in building a new sub-tree with the vertex labelled by 3, and the pointer arrays, e.g., {*fptr*[0], *fptr*[1]} and the identity arrays {*fids*[0], *fids*[1], *fids*[2], *val*} only need add elements in the tail.

5. Experiments

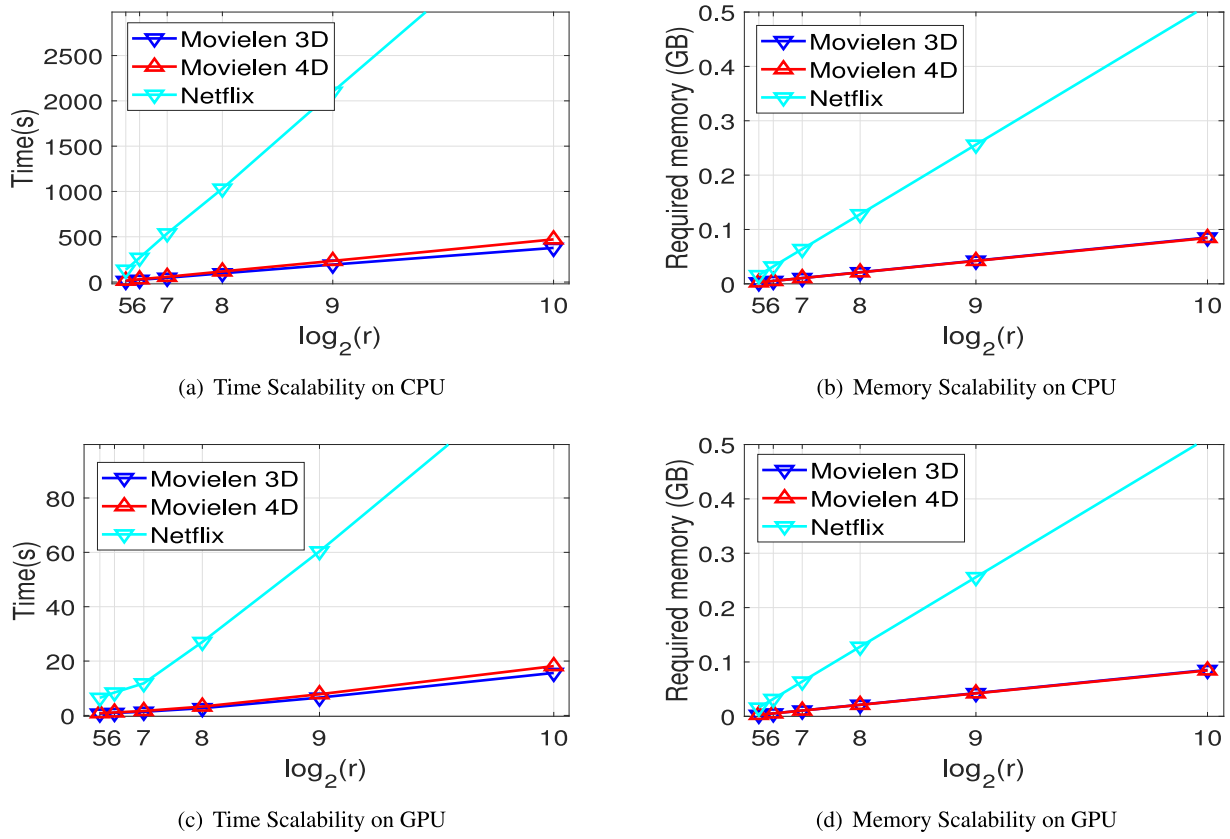
Multiple sets of experiments will be constructed to analyze the following issues: (1) Memory scalability with the value of rank and computing time scalability (Section 5.2); (2) Convergence analysis and the performances on multi-GPU and online learning (Section 5.3).

5.1. Experimental settings

The accuracy of tensor factorization is evaluated by Root Mean Square Error (RMSE)[47], which is defined as

$$RMSE = \sqrt{\frac{\sum_{(i_1,\ldots,i_n)\in\Phi}(\mathcal{X}_{(i_1,\ldots,i_n)} - \widehat{\mathcal{X}}_{(i_1,\ldots,i_n)})^2}{|\Phi|}} \tag{23}$$

where $|\Phi|$ denotes the amount of the non-zero entries in test sets. In order to evaluate the experimental performances, we



(a) Time Scalability on CPU

(b) Memory Scalability on CPU

(c) Time Scalability on GPU

(d) Memory Scalability on GPU

Fig. 11. Scalability and memory requirement of GSSNTF$_{KL}$ are scalable with R on CPU (Fig. 11(a) and 11(b),time on second (s)). Scalability and memory requirement of CUGSSNTF$_{KL}$ are scalable with R on GPU (Fig. 11(c) and 11(d), time on second (s)).

adopt two data sets to demonstrate the performances of tensor factorization, e.g., MovieLens[1], Netflix[2], which is shown in Table 3. We conduct our experiments and comparison codes on 8 P100 GPUs, each of them has 56 SMs and OpenMP. On each GPU, there are 128 SPs per SM, and each NVIDIA Tesla P100 GPU works at a 1.33 GHz clock rate. OpenMP platform has two 16 cores CPUs. The global memory, bandwidth and memory clock rate are 16 GB, 4096 bits and 715 MHz, respectively. For simplification, we use GSSNTF$_{Eu}$, GSSNTF$_{KL}$, and GSSNTF$_{IS}$ to denote the GSSNTF approach on Euclidean distance, KL- and IS-divergence, respectively, and we use CUGSSNTF$_{Eu}$, CUGSSNTF$_{KL}$, and CUGSSNTF$_{IS}$ to denote the responding CUDA parallelization approaches. We compare the accuracy of GSSNTF$_{Eu}$, GSSNTF$_{KL}$ and ALS on 3-way tensor data ($1000 \times 1000 \times 100$ with sparsity 0.1%) with $\mathcal{P}oisson$ distribution which is generated by MATLAB and we compare the accuracy of GSSNTF$_{Eu}$, GSSNTF$_{IS}$ and ALS on 3-way tensor data ($1,000 \times 1,000 \times 100$ with sparsity 0.1%) with $\mathcal{E}xponential$ distribution which is generated by MATLAB. We use simulation datasets of $\mathcal{P}oisson$ and $\mathcal{E}xponential$ distribution styles to test the accuracy of GSSNTF$_{Eu}$, GSSNTF$_{KL}$, GSSNTF$_{IS}$ and ALS [49]. Meanwhile, We use the corresponding MATLAB code of [49] for tensor operations which are proposed on Section 3.3. We test the scalability with the rank of feature matrices. Thus, we set the rank value R as {32, 64, 128, 256, 512, 1024} for scalability test, which is the multiple of the *warp*. We choose uniform distribution as the initial value for factor matrices $\{\mathbf{A}^{(i)}|i \in \{1,\ldots,N\}\}$, which are the same as the comparison approaches.

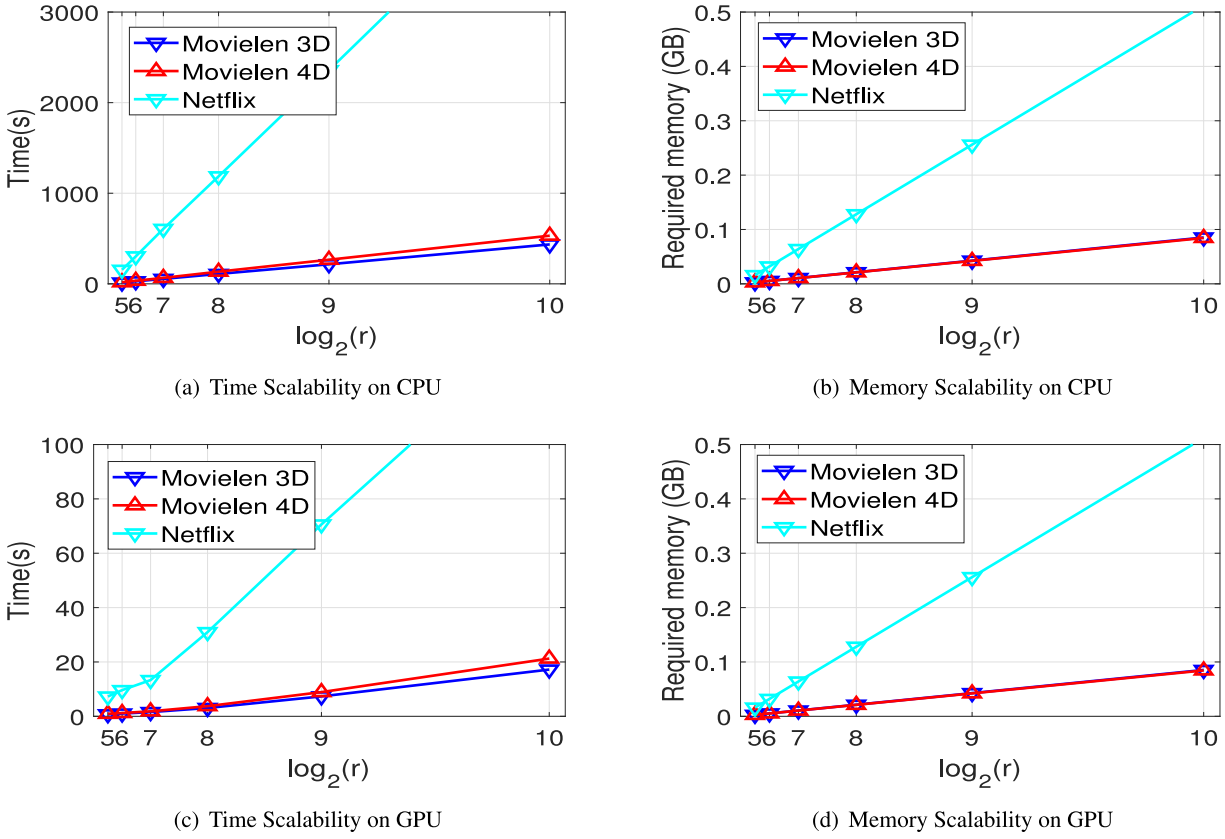
[1] http://files.grouplens.org/datasets/movieLens

[2] http://www.netflixprize.com

5.2. Scalability and parameter selection

We test the scalability for computational time and space overhead on this section. The scalability performances of computational time on CPU for GSSNTF$_{Eu}$, GSSNTF$_{KL}$ and GSSNTF$_{IS}$ are presented on Figs. 10(a),11(a) and 12(a), respectively. The scalability performances of space overhead on CPU for GSSNTF$_{Eu}$, GSSNTF$_{KL}$ and GSSNTF$_{IS}$ are presented on Figs. 10(b),11(b) and 12(b), respectively. The scalability performances of computational time on GPU for CUGSSNTF$_{Eu}$, CUGSSNTF$_{KL}$ and CUGSSNTF$_{IS}$ are presented on Figs. 10(c),11(c) and 12(c), respectively. The scalability performances of space overhead on GPU for CUGSSNTF$_{Eu}$, CUGSSNTF$_{KL}$ and CUGSSNTF$_{IS}$ are presented on Figs. 10(d),11(d) and 12(d), respectively. With the rapid increasing of web data, processing scalability becomes a core index. Scalability in the research communities of tensor factorization includes time scalability and memory scalability. Time scalability comprises of: (1) The scalability performance with the scale of a data set; (2) The scalability with the value of R. The choice of R can influence the approximation degree. The small value of R cannot approximate the original tensor appropriately; However, a big value of R may make an over-fitting problem due to more number of training parameters. In this work, only the scalability performance of tensor factorization is considerd, and the R is selected as {32, 64, 128, 256, 512, 1024} or $\log_2(R) \in \{5, 6, 7, 8, 9, 10\}$.

The testing results of time scalability (Figs. 10(a), 11(a) and 12(a)) on CPU show that GSSNTF is scalable. However, we found that the results of time scalability (Figs. 10(c), 11(c) and 12(c)) on GPU are not scalable between $R = 64$ and $R = 128$. We conclude the reason is that when GPU is running on $R = \{32, 64\}$, the SPs are more idle than the GPU runs on $R = \{128, 256, 512, 1024\}$. From



(a) Time Scalability on CPU

(b) Memory Scalability on CPU

(c) Time Scalability on GPU

(d) Memory Scalability on GPU

Fig. 12. Scalability and memory requirement of GSSNTF$_{IS}$ are scalable with R on CPU (Fig. 12(a) and 12(b), time on second (s)). Scalability and memory requirement of CUGSSNTF$_{IS}$ are scalable with R on GPU (Fig. 12(c) and 12(d), time on second (s)).

Table 4
The computational time (minute) on a training iteration (Netflix data set), with $R = \{32, 64, 128, 256, 512, 1024\}$ on 32-core shared memory platform. Speedup refers to the ratio of SPLATT to GSSNTF$_{Eu}$.

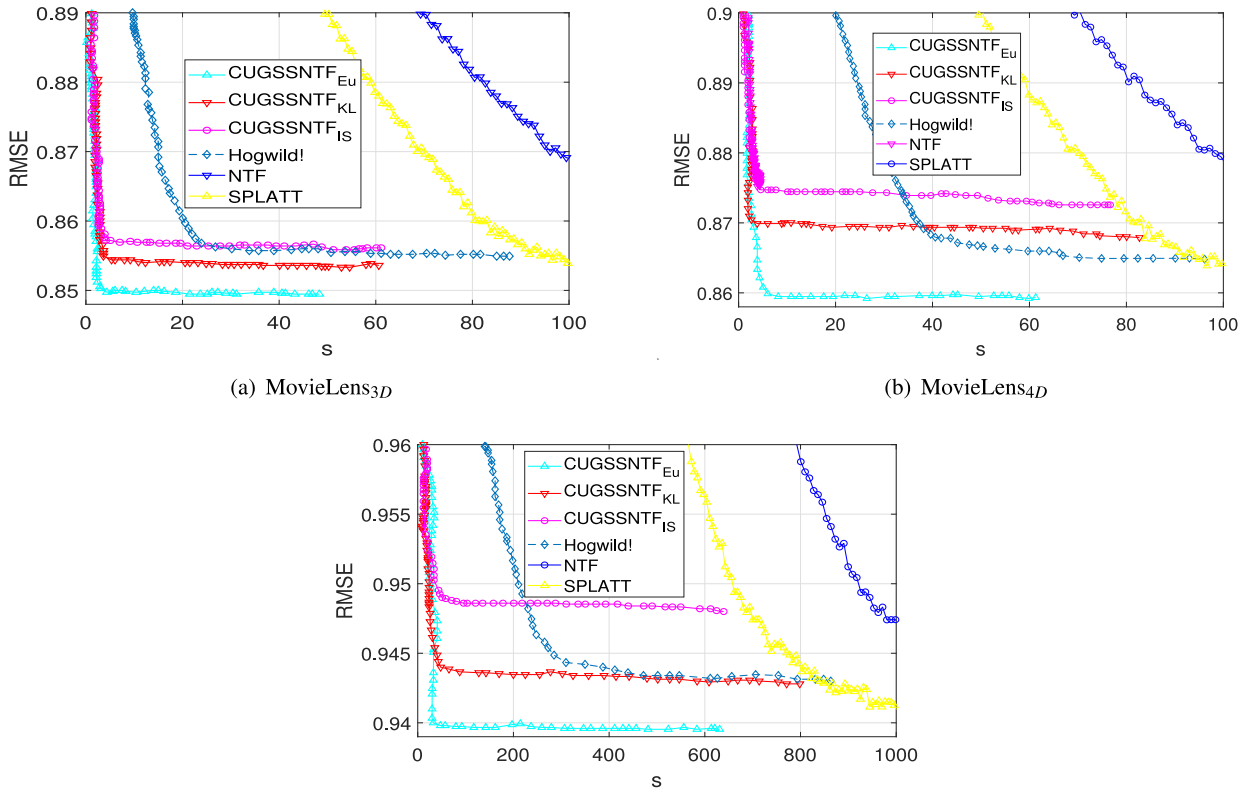
Rank	GSSNTF$_{Eu}$	NTF	SPLATT	Speedup
$R = 32$	0.10	0.87	0.62	6.24
$R = 64$	0.20	3.30	2.36	11.81
$R = 128$	0.39	12.58	8.99	22.93
$R = 256$	0.78	49.36	35.26	45.21
$R = 512$	1.56	195.99	139.99	89.74
$R = 1024$	3.12	781.05	557.89	178.81

the results of time scalability (Figs. 10(a), 11(a) and 12(a)) on CPU and the results of time scalability (Figs. 10(c), 11(c) and 12(c)) on GPU, the running time of $GSSNTF_{IS}$ is longer than $GSSNTF_{KL}$; Meanwhile the running time of $GSSNTF_{KL}$ is longer than $GSSNTF_{Eu}$. Because more number of parameters need longer time of data fetching, the running results support that more parameters output more computational time. The Figs. 10(b), 11(b) and 12(b) show that the memory requirement on CPU for GSSNTF is increased linearly with R. Meanwhile, the Figs. 10(d), 11(d) and 12(d) show that the memory requirement on GPU for CUGSSNTF is increased linearly with R. Table 4 shows the speedup. Table 4 demonstrates that GSSNTF has linear scalability with R. Meanwhile, we observe that, due to the computational process of the intermediate Hession matrix $\mathbf{G}$ with the discussion on Section 3.3, the computational overhead of NTF and SPLATT for SNTF on Euclidean Distance is scalable with $O(R^2)$.

5.3. Convergence, multi-GPU and online learning

We compare the convergence performances of CUGSSNTF, NTF [11], Hogwild! [40] and SPLATT [8,9]. From the results on Fig. 13 (a-c), CUGSSNTF has the same convergence performances with NTF and SPLATT, and CUGSSNTF$_{Eu}$ can converge to the same RMSE value as NTF, and SPLATT. Hogwild! is a framework of parallelization SGD, which can avoid the over-writing problem with convergence promise and initial parameters sensitivity. However, Hogwild! cannot obtain the same accuracy level as NTF, SPLATT, and GSSNTF. We conclude the reason is that we choose the original version of Hogwild! and don't consider other information. In the future, we may improve this condition. The Fig. 13 (a-c) illustrate that: 1) Three prior assumption styles, i.e., $\mathcal{G}aussian$, $\mathcal{P}oisson$, and $\mathcal{E}xponential$, present different accuracy results; 2) CUGSSNTF$_{Eu}$ has the best RMSE performance result. We conclude the result is that the data follow the $\mathcal{G}aussian$. Meanwhile, we found that in the discussion of [50], if we carefully choose the distance styles for a matrix data, the methods of non-negative and low-rank factor for matrix data on $\mathcal{P}oisson$ and $\mathcal{E}xponential$ may obtain higher accuracy performance than the matrix data on $\mathcal{G}aussian$. In this work, we not only explore the linear scalability improvement for the generalized algorithm but also conduct the further explorations for the detail performance. From the experimental results, because the accuracy results demonstrate the convergency performance of GSSNTF, the proposed GSSNTF is correct and linear scalable. Thus, the GSSNTF can be adopted by large-scale industrial data for low-rank analysis of tensor data. At last, CUGSSNTF runs faster than the NTF and SPLATT. We conclude the reason is that CUGSSNTF can avoid frequent matrix manipulations and the formations of intermediate matrices.

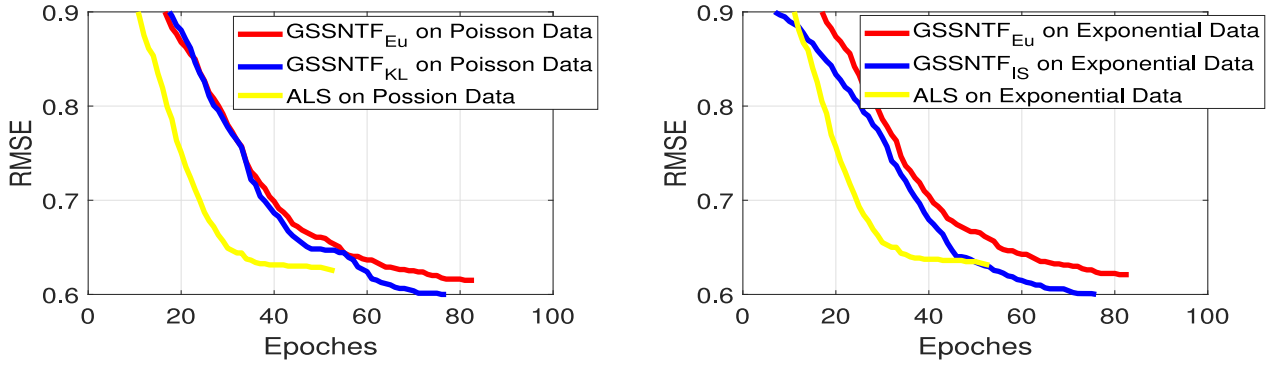
Due to the higher sparsity of MovieLens4D than MovieLens3D, the RMSE of CUGSSNTF on MovieLens4D obtains a somewhat higher value than the RMSE of CUGSSNTF on MovieLens3D, which means that CUGSSNTF on MovieLens4D obtains somewhat



(a) MovieLens$_{3D}$ (b) MovieLens$_{4D}$

(c) Netflix

Fig. 13. RMSE versus computational time on a 32-core shared memory system for NTF, SPLATT, and Hogwild!. A P100 GPU for CUGSSNTF (Time on Seconds (s)).

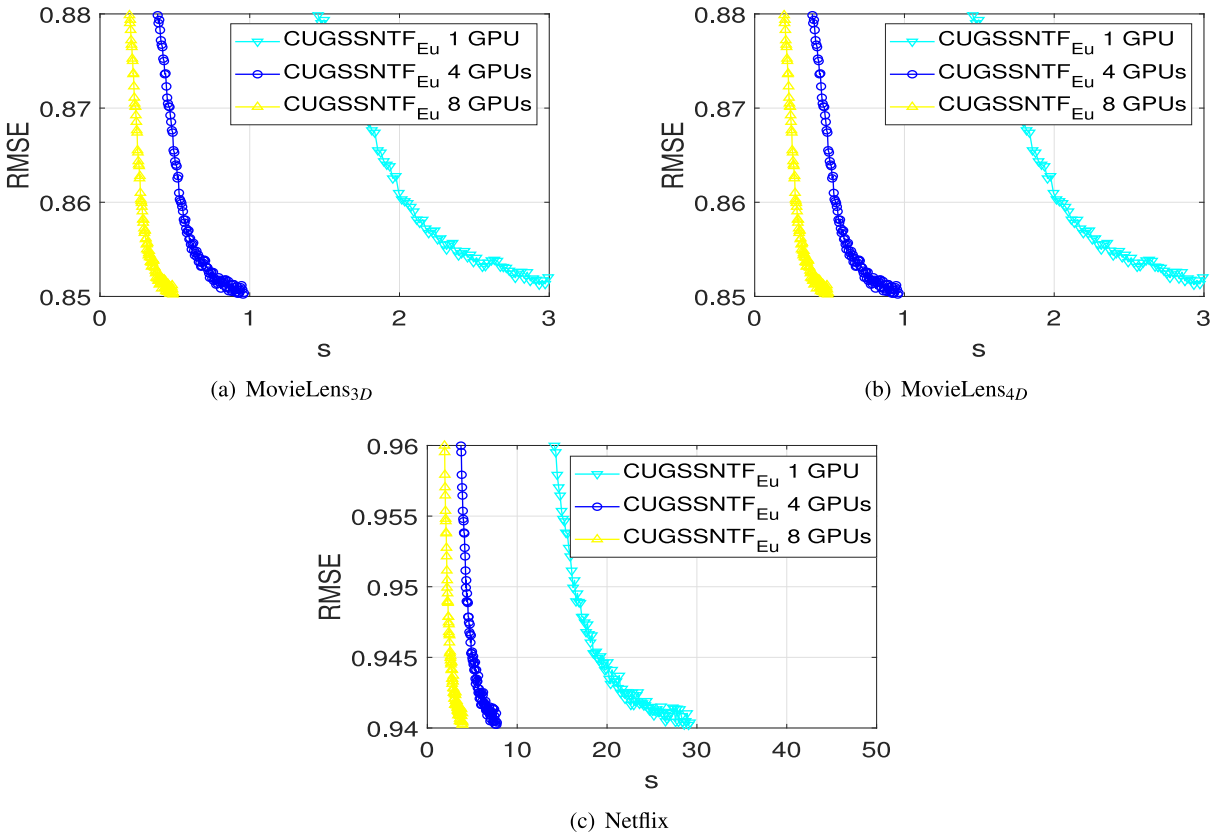


(a) Comparison on Possion Data (b) Comparison on Exponential Data

Fig. 14. RMSE vs training epoches for ALS, GSSNTF$_{Eu}$ and GSSNTF$_{KL}$ are tested on Possion simulation data (Fig. 14(a)). RMSE vs training epoches for ALS, GSSNTF$_{Eu}$ and GSSNTF$_{IS}$ are tested on Exponential simulation data (Fig. 14(b)).

higher accuracy than CUGSSNTF on MovieLens3D. We observe that the sparsity on MovieLens3D is $\left(9,301,274/(71,567*10,681*1,038)\right)=0.0017\%$ and the sparsity on MovieLens4D is $\left(9,301,274/(71,567*10,681*169*24)\right)=0.0004\%$. However, the number of training parameters of CUGSSNTF on MovieLens4D (71, 567*10, 681*169*24*R) is much larger than on the number of training parameters of MovieLens3D (71, 567*10, 681*1, 038*R). Although the higher sparsity results in the decreasing of accuracy, 4-way tensor factorization present more robust than 3-way tensor factorization in the condition of information missing.

We compare GSSNTF$_{Eu}$, GSSNTF$_{KL}$ and GSSNTF$_{IS}$ on $\mathcal{P}oisson$ and $\mathcal{E}xponential$ distribution styles. As shown in Fig. 14(a), the accuracy performance of GSSNTF$_{KL}$ outperforms the accuracy performance of GSSNTF$_{Eu}$ and ALS on $\mathcal{P}oisson$ distribution style. Meanwhile, as shown in Fig. 14(b), the accuracy performance of GSSNTF$_{IS}$ outperforms the accuracy performance of GSSNTF$_{Eu}$ and ALS on $\mathcal{E}xponential$ distribution style. The results demonstrate the versatility of the proposed generalized model. From the illustration of Fig. 14(a) and Fig. 14(b), ALS runs faster than GSSNTF$_{Eu}$, GSSNTF$_{KL}$ and GSSNTF$_{IS}$. However, ALS involves the inversion operations of Hession matrix and the computational complexity is $O(R^3)$.



(a) MovieLens$_{3D}$ (b) MovieLens$_{4D}$

(c) Netflix

Fig. 15. The time overhead of MCUGSSNTF on one, four, and eight GPUs.

Table 5
Training time influence of η on GSSNTF$_{Eu}$ in obtaining a baseline RMSE (Time on second (s)).

Data sets	CUGSSNTF$_{Eu}$	SPLATT	Online $\eta = 0.5$	Online $\eta = 1.0$	Online $\eta = 1.5$	Online $\eta = 2.0$
Movielen$_{3D}$	2.130	9.163	0.00596	0.00603	0.00636	0.00589
Movielen$_{4D}$	2.369	10.960	0.00693	0.00856	0.00785	0.00586
Netflix-100M	86.69	635.34	0.5212	0.6370	0.5720	0.6806

Table 6
Training time influence of η on GSSNTF$_{KL}$ in obtaining a baseline RMSE (Time on second (s)).

Data sets	CUGSSNTF$_{KL}$	Online $\eta = 0.5$	Online $\eta = 1.0$	Online $\eta = 1.5$	Online $\eta = 2.0$
Movielen$_{3D}$	2.562	0.00698	0.00663	0.00636	0.00609
Movielen$_{4D}$	2.639	0.00783	0.00757	0.00774	0.00701
Netflix-100M	91.43	0.6013	0.6040	0.6532	0.6924

Table 7
Training time influence of η on GSSNTF$_{IS}$ in obtaining a baseline RMSE (Time on second (s)).

Data sets	CUGSSNTF$_{IS}$	Online $\eta = 0.5$	Online $\eta = 1.0$	Online $\eta = 1.5$	Online $\eta = 2.0$
Movielen$_{3D}$	2.940	0.00756	0.00785	0.00754	0.00742
Movielen$_{4D}$	2.876	0.00743	0.00790	0.00774	0.00743
Netflix-100M	92.75	0.7365	0.7453	0.7425	0.7123

MCUGSSNTF is the multi-GPU version of CUGSSNTF. MCUGSSNTF considers the data with 3-way and 4-way. As shown in Fig. 15, due to the sparsity of Movielens and Netflix data sets, there is load unbalance within each GPU. Thus, MCUGSSNTF on 4 and 8 GPU can only obtain the {3.78*X*, 3.68*X*, 3.76*X*} on 4 GPUs for Movielens$_{3D}$, Movielens$_{4D}$, and Netflix, respectively, and {7.4*X*, 7.2*X*, 7.41*X*} on 8 GPUs for Movielens$_{3D}$, Movielens$_{4D}$, and Netflix, respectively. MCUGSSNTF defines more complex communication pattern on 4-way data than on 3-way data. Thus, the speedup performance of MCUGSSNTF on Movielens$_{4D}$ is lower than the speedup performance of MCUGSSNTF on Movielens$_{3D}$.

The results of online learning performance reveal the real-time performances and parameter influences. In Table 5, η represents the weight of the new data, which is used in Formulas (21) and

(22); Columns 2 to 3 represent the total time cost for GSSNTF$_{Eu}$ and SPLATT [8] to process new and old data without an online way, respectively; Columns 4 to 7 represent the time cost required for GSSNTF$_{Eu}$ to process new data under different values of η in an online way. It can be observed that GSSNTF$_{Eu}$ has less time cost than SPLATT. As described in sections III and IV, GSSNTF$_{Eu}$ and SSNTF have the same time cost as GSSNTF$_{Eu}$ inherits the advantages of SSNTF based on single thread method. Obviously, if we use our online processing algorithm, it takes very little time to process new data according to Columns 4 to 7. And the parameter η influences the time cost of GSSNTF$_{Eu}$ for online version. Meanwhile, we extend the work of online processing to the KL-divergence, and IS-divergence. We perform the same experiment on GSSNTF$_{KL}$ and GSSNTF$_{IS}$, and the experimental results can also draw similar conclusions as shown in Tables 6 and 7. From the results of Tables 5–7, the strategy of online learning for GSSNTF is workable.

6. Conclusion and future work

This paper is an extension of our previous work [16] from a single Euclidean distance to generalized divergence styles, and the main works are presented as follows:

1. We analyze that the main computing bottlenecks for SNTF are frequent matrix manipulation and large-scale intermediate matrices;
2. The GSSNTF model is discussed, which can reduce to linear time complexity and space requirement with convergence promise. And we present a more generalized update rule for factor matrices;
3. An online model for GSSNTF is presented, which includes data merging for built data structure and newly arrived data and online learning algorithm;
4. When the GSSNTF faces a large-scale dataset, a CUDA parallelization model for GSSNTF is presented on GPU (CUGSSNTF) and multi-GPU (MCUGSSNTF).

CUGSSNTF and online learning approach are the deep extensions of the GSSNTF. Experimental results demonstrate the perfect performances of the CUGSSNTF and the online learning approach of the GSSNTF.

In the future works, we will mine the further performance of the parallelization features on cloud platforms, i.e., Flink, Spark and so on. We will combine the accelerating approach, i.e., Alternative Direction Method (ADM) to improve the convergence rate. Meanwhile, we will explore how to combine the manifold learning and NTF to improve the accuracy and we will extend the MCUGSSNTF to the sparse data styles with 5 or more modes. Furthermore, deep learning is a new perspective that can capture the inherent information via a neural network. Thus, in the future, we explore how to combine deep learning and tensor learning.

Authors' Contributions

Non-negative Tensor Factorization models (NTF) are effective and efficient in extracting useful knowledge from various types of probabilistic distribution with multi-way information. Current NTF models are mostly designed for problems in computer vision which involve the whole *Matricized Tensor Times Khatri - Rao Product* (MTTKRP). Meanwhile, a Sparse NTF (SNTF) proposed to solve the problem of sparse Tensor Factorization (TF) can results in large-scale intermediate data. A model of Single-thread-based SNTF (SSNTF) is proposed to solve the problem of non-linear time and space overhead caused by large-scale intermediate data. However, the SSNTF is not a generalized model. Furthermore, the above methods cannot describe the stream-like data from industrial applications in mainstream processors, e.g, Graphics Processing Unit (GPU) and multi-GPU in an online way. To address these two issues, a Generalized SSNTF (GSSNTF) is proposed, which extends the works of SSNTF to the Euclidean distance, KullbackLeibler (KL)-divergence, and Itakura-Saito (IS)-divergence. The GSSNTF only involves the feature elements rather than on the whole factor matrices, and can avoid the formation of large-scale intermediate matrices with convergence and accuracy promises. Furthermore, GSSNTF can merge the new data into the state-of-the-art built tree dataset for sparse tensor, and then the online learning has the promise of correct data format. At last, a model of Compute Unified Device Architecture (CUDA) parallelizing GSSNTF (CUGSSNTF) is proposed on GPU and Multi-GPU (MCUGSSNTF). Thus, CUGSSNTF has linear computing complexity and space requirement, and linear communication overhead on multi-GPU. We implement CUGSSNTF and MCUGSSNTF on 8 P100 GPUs, and the experimental results from real-world industrial data sets demonstrate the linear scalability and 40X speedup performances of CUGSSNTF than the state-of-the-art parallel and distributed methods.

The main contributions of our study are as follows:

(1) **Algorithm analysis**. The GSSNTF model is derived theoretically, which obeys various common probabilistic styles, e.g., $\mathscr{G}$aussian, $\mathscr{P}$oisson, and $\mathscr{E}$xponential distribution styles, and fine-grained parallelization inherence. And GSSNTF represents a more generalized update rule for factor matrices.

(2) **CUDA Parallelization** and **Multi-GPU**. CUGSSNTF harnesses local memory more instead of increasing the global memory overhead of the GPU, and CUGSSNTF has linear scalability of computation and space overhead. MCUGSSNTF adopts a multi-GPU model with mode scalability for tensor data. GSSNTF and CUGSSNTF have linear time and space complexities.

(3) **Online Learning**. The stream-like computing style of the single-thread-based model gives the GSSNTF the online learning ability. Online learning and high efficient CUDA parallelization are the two byproducts of the singlethread-based model.

Declaration of Competing Interest

We declare that we have no conflict of interest.

Acknowlgedgments

Thanks for the Editor-in-chief (EiC) and Associate editor (AE) to find the suitable reviewers for improving the paper quality.

References

[1] Y. Zhao, L.T. Yang, J. Sun, Privacy-preserving tensor-based multiple clusterings on cloud for industrial Iot, IEEE Trans. Ind. Inf. 15 (4) (2018) 2372–2381.
[2] Z. Zhang, C. Hawkins, Variational bayesian inference for robust streaming tensor factorization and completion, in: Proceedings of the IEEE International Conference on Data Mining (ICDM), IEEE, 2018, pp. 1446–1451.
[3] J. Hatvani, A. Basarab, J.-Y. Tourneret, M. Gyöngy, D. Kouamé, A tensor factorization method for 3-d super resolution with application to dental ct, IEEE Trans. Med. Imaging 38 (6) (2018) 1524–1531.
[4] J.L. Hinrich, S.F. Nielsen, K.H. Madsen, M. Mørup, Variational Bayesian partially observed non-negative tensor factorization, in: Proceedings of the IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP), IEEE, 2018, pp. 1–6.
[5] I. Perros, E.E. Papalexakis, R. Vuduc, E. Searles, J. Sun, Temporal phenotyping of medically complex children via parafac2 tensor factorization, J Biomed Inform 93 (2019) 103125.
[6] Q. Jia, Y. Zhang, W. Chen, Image-based process monitoring using projective nonnegative-tensor factorization, IEEE Trans. Ind. Electron. (2018) 1, doi:10.1109/TIE.2018.2833027.
[7] X. Luo, H. Wu, H. Yuan, M. Zhou, Temporal pattern-aware QoS prediction via biased non-negative latent factorization of tensors, IEEE Trans. Cybern. (2019), doi:10.1109/TCYB.2019.2903736.
[8] S. Smith, G. Karypis, A medium-grained algorithm for distributed sparse tensor factorization, in: Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2016.

[9] S. Smith, N. Ravindran, N.D. Sidiropoulos, G. Karypis, Splatt: Efficient and parallel sparse tensor-matrix multiplication, in: Proceedings of the IEEE International Parallel and Distributed Processing Symposium, 2015, pp. 61–70, doi:10.1109/IPDPS.2015.27.
[10] V.T. Chakaravarthy, S.S. Pandian, S. Raje, Y. Sabharwal, On optimizing distributed non-negative tucker decomposition, in: Proceedings of the ACM International Conference on Supercomputing, ACM, 2019, pp. 238–249.
[11] K. Shin, L. Sael, U. Kang, Fully scalable methods for distributed tensor factorization, in: Proceedings of the IEEE Transactions on Knowledge and Data Engineering, 29, IEEE, 2017, pp. 100–113.
[12] Z. Blanco, B. Liu, M.M. Dehnavi, CSTF: Large-scale sparse tensor factorizations on distributed platforms, in: ICPP Proceedings of the 47th International Conference on Parallel Processing, 2018.
[13] H. Li, K. Li, J. An, K. Li, MSGD: a novel matrix factorization approach for large-scale collaborative filtering recommender systems on GPUs, IEEE Trans. Parallel Distrib. Syst. 29 (7) (2018) 1530–1544.
[14] S. Ona, T. Kasai, Efficient constrained tensor factorization by alternating optimization with primal-dual splitting, in: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2018, pp. 3379–3383.
[15] J.B. Smith, M. Goto, Nonnegative tensor factorization for source separation of loops in audio, in: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2018, pp. 171–175.
[16] H. Li, K. Li, J. An, K. Li, CUSNTF: a scalable sparse non-negative tensor factorization model for large-scale industrial applications on multi-GPU, in: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, ACM, 2018, pp. 1113–1122.
[17] H. Li, K.G. Li, J. An, K.G. Li, An online and scalable model for generalized sparse non-negative matrix factorization in industrial applications on multi-GPU, IEEE Trans. Ind. Inf. (2019) 1, doi:10.1109/TII.2019.2896634.
[18] J. Ye, K. Chen, T. Wu, J. Li, Z. Zhao, R. Patel, M. Bae, R. Janardan, H. Liu, G. Alexander, et al., Heterogeneous data fusion for Alzheimer's disease study, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2008, pp. 1025–1033.
[19] Y. Sun, J. Gao, X. Hong, B. Mishra, B. Yin, Heterogeneous tensor decomposition for clustering via manifold optimization, IEEE Trans. Pattern Anal. Mach. Intell. 38 (3) (2015) 476–489.
[20] F. Xiong, Y. Qian, J. Zhou, Y.Y. Tang, Hyperspectral unmixing via total variation regularized nonnegative tensor factorization, IEEE Trans. Geosci. Remote Sens. 57 (4) (2019) 2341–2357, doi:10.1109/TGRS.2018.2872888.
[21] X. Luo, M. Zhou, S. Li, M. Shang, An inherently nonnegative latent factor model for high-dimensional and sparse matrices from industrial applications, IEEE Trans. Ind. Inf. 14 (5) (2017) 2011–2022.
[22] Z. Yang, Y. Zhang, Y. Xiang, W. Yan, S. Xie, Non-negative matrix factorization with dual constraints for image clustering, IEEE Trans. Syst. Man Cybern. Syst. (2018) 1–10, doi:10.1109/TSMC.2018.2820084.
[23] G. Cui, X. Li, Y. Dong, Subspace clustering guided convex nonnegative matrix factorization, Neurocomputing 292 (2018) 38–48.
[24] L. Zhao, Z. Chen, Y. Yang, Z.J. Wang, V.C. Leung, Incomplete multi-view clustering via deep semantic mapping, Neurocomputing 275 (2018) 1053–1062.
[25] X. Luo, M. Zhou, S. Li, Y. Xia, Z.-H. You, Q. Zhu, H. Leung, Incorporation of efficient second-order solvers into latent factor models for accurate prediction of missing QoS data, IEEE Trans. Cybern. 48 (4) (2017) 1216–1228.
[26] X. Luo, Z. Wang, M. Shang, An instance-frequency-weighted regularization scheme for non-negative latent factor analysis on high-dimensional and sparse data, IEEE Trans. Syst. Man Cybern. Syst. (2019), doi:10.1109/TSMC.2019.2930525.
[27] X. Wang, T. Zhang, X. Gao, Multiview clustering based on non-negative matrix factorization and pairwise measurements, IEEE Trans. Cybern. 49 (9) (2018) 3333–3346.
[28] X.-R. Feng, H.-C. Li, J. Li, Q. Du, A. Plaza, W.J. Emery, Hyperspectral unmixing using sparsity-constrained deep nonnegative matrix factorization with total variation, IEEE Trans. Geosci. Remote Sens. 56 (10) (2018) 6245–6257.
[29] B. Bauer, M. Kohler, et al., On deep learning as a remedy for the curse of dimensionality in nonparametric regression, Ann. Stat. 47 (4) (2019) 2261–2285.
[30] H. Li, K. Li, J. An, W. Zheng, K. Li, An efficient manifold regularized sparse non-negative matrix factorization model for large-scale recommender systems on GPUs, Inf. Sci. (Ny) 496 (2019) 464–484.
[31] X. Wei, W. Zhu, B. Liao, L. Cai, Scalable one-pass self-representation learning for hyperspectral band selection, IEEE Trans. Geosci. Remote Sens. (2019).
[32] D. Liu, P. Xu, L. Ren, Tpflow: progressive partition and multidimensional pattern extraction for large-scale spatio-temporal data analysis, IEEE Trans. Vis. Comput. Graph. 25 (1) (2018) 1–11.
[33] R. Tomioka, T. Suzuki, K. Hayashi, H. Kashima, Statistical performance of convex tensor decomposition, in: Advances in Neural Information Processing Systems, 2011, pp. 972–980.
[34] X. Luo, M. Zhou, S. Li, D. Wu, Z. Liu, M. Shang, Algorithms of unconstrained non-negative latent factor analysis for recommender systems, IEEE Trans. Big Data (2019), doi:10.1109/TBDATA.2019.2916868.
[35] Y.-D. Kim, A. Cichocki, S. Choi, Nonnegative tucker decomposition with alpha–divergence, in: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2008, pp. 1829–1832.
[36] Y.-D. Kim, S. Choi, Weighted nonnegative matrix factorization, in: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2009, pp. 1541–1544.
[37] Y. Chen, K. Li, W. Yang, G. Xiao, X. Xie, T. Li, Performance-aware model for sparse matrix-matrix multiplication on the sunway taihulight supercomputer, IEEE Trans. Parallel Distrib. Syst. 30 (4) (2018) 923–938.
[38] J. Li, B. Uçar, Ü.V. Çatalyürek, J. Sun, K. Barker, R. Vuduc, Efficient and effective sparse tensor reordering, in: Proceedings of the ACM International Conference on Supercomputing, ACM, 2019, pp. 227–237.
[39] X. Luo, Z. Liu, S. Li, M. Shang, Z. Wang, A fast non-negative latent factor model based on generalized momentum method, IEEE Trans. Syst. Man. Cybern. Syst. (2018), doi:10.1109/TSMC.2018.2875452.
[40] B. Recht, C. Re, S. Wright, F. Niu, Hogwild!: A lock-free approach to parallelizing stochastic gradient descent, in: Advances in Neural Information Processing Systems, 2011, pp. 693–701.
[41] R. Gemulla, E. Nijkamp, P.J. Haas, Y. Sismanis, Large-scale matrix factorization with distributed stochastic gradient descent, in: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2011, pp. 69–77.
[42] O. Kaya, B. Uçar, Scalable sparse tensor decompositions in distributed memory systems, in: Proceedings of the SC-International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2015, pp. 1–11.
[43] J. Antikainen, J. Havel, R. Josth, A. Herout, P. Zemcik, M. Hautakasari, Nonnegative tensor factorization accelerated using GPGPU, in: Proceedings of the IEEE Transactions on Parallel and Distributed Systems, 22, 2011, pp. 1135–1141.
[44] A. Shashua, T. Hazan, Non-negative tensor factorization with applications to statistics and computer vision, in: Proceedings of the International Conference on Machine Learning, 2005, pp. 792–799.
[45] M. Welling, M. Weber, Positive tensor factorization, in: Pattern Recognition Letters, 22, 2001, pp. 1255–1261.
[46] D.D. Lee, H.S. Seung, Algorithms for non-negative matrix factorization, in: Advances in Neural Information Processing Systems, 2001, pp. 556–562.
[47] W. Zhang, H. Sun, X. Liu, X. Guo, Temporal qos-aware web service recommendation via non-negative tensor factorization, in: Proceedings of the 23rd International Conference on World Wide Web, ACM, 2014, pp. 585–596.
[48] K. Xie, X. Li, X. Wang, J. Cao, G. Xie, J. Wen, D. Zhang, Z. Qin, On-line anomaly detection with high accuracy, IEEE/ACM Trans. Netw. 26 (3) (2018) 1222–1235.
[49] T.G. Kolda, B.W. Bader, Tensor decompositions and applications, SIAM Rev. 51 (3) (2009) 455–500.
[50] X. Luo, Y. Yuan, M. Zhou, Z. Liu, M. Shang, Non-negative latent factor model based on β-divergence for recommender systems, IEEE Trans. Syst. Man Cybern. Syst. (2019), doi:10.1109/TSMC.2019.2931468.

Linlin Zhuo is currently working toward the Ph.D. degree at Hunan University, China. His research interests are mainly in clustering algorithm, object detection and multi-GPU computing.

Kenli Li received the Ph.D. degree in computer science from Huazhong University of Science and Technology, China, in 2003. He was a visiting scholar at University of Illinois at Urbana Champaign from 2004 to 2005. He is currently a full professor of computer science and technology at Hunan University 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 IEEE-TC, IEEE-TPDS, IEEE-TSP, JPDC, ICPP, CCGrid. He is an outstanding member of CCF. He is a senior member of the IEEE and serves on the editorial board of IEEE Transactions on Computers.

Hao Li is currently working toward the Ph.D. degree at Hunan University, China. His research interests are mainly in large-scale sparse matrix and tensor factorization, recommender systems, social network, data mining, machine learning, and GPU and multi-GPU computing. He has published 6 journal and conference papers in IEEE TPDS, InforSci, IEEE-TII, CIKM, and ISPA.

Jiwu Peng is currently a Ph.D. candidate in Computer Science, Hunan University, China. His research main interest includes mobile cloud/edge computing, high-performance computing, artificial intelligence. He has published several research articles in international conference and journals of machine learning algorithms and parallel computing.Jiwu Peng is currently a PhD candidate in Computer Science, Hunan University, China. His research main interest includes mobile cloud/edge computing, high-performance computing, artificial intelligence. He has published several research articles in international conference and journals of machine learning algorithms and parallel computing.

Dr. Keqin Li is a SUNY Distinguished Professor of computer science with the State University of New York. His current research interÈts include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has published over 620 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He currently serves or has served on the editorial boards of the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Computers, the IEEE Transactions on Cloud Computing, the IEEE Transactions on Services Computing, and the IEEE Transactions on Sustainable Computing. He is an IEEE Fellow.