Multi-Aspect Incremental Tensor Decomposition Based on Distributed In-Memory Big Data Systems

Hye-Kyung Yang¹, Hwan-Seung Yong^{2†}

¹Department of Computer Software, Korean Bible University, Seoul, 01757, Republic of Korea

²Department of Computer Science and Engineering, Ewha Womans University, Seoul, 03760, Republic of Korea

Abstract

Purpose: We propose InParTen2, a multi-aspect parallel factor analysis three-dimensional tensor decomposition algorithm based on the Apache Spark framework. The proposed method reduces re-decomposition cost and can handle large tensors.

Design/methodology/approach: Considering that tensor addition increases the size of a given tensor along all axes, the proposed method decomposes incoming tensors using existing decomposition results without generating sub-tensors. Additionally, InParTen2 avoids the calculation of Khari–Rao products and minimizes shuffling by using the Apache Spark platform.

Findings: The performance of InParTen2 is evaluated by comparing its execution time and accuracy with those of existing distributed tensor decomposition methods on various datasets. The results confirm that InParTen2 can process large tensors and reduce the re-calculation cost of tensor decomposition. Consequently, the proposed method is faster than existing tensor decomposition algorithms and can significantly reduce re-decomposition cost.

Research limitations: There are several Hadoop-based distributed tensor decomposition algorithms as well as MATLAB-based decomposition methods. However, the former require longer iteration time, and therefore their execution time cannot be compared with that of Spark-based algorithms, whereas the latter run on a single machine, thus limiting their ability to handle large data.

Practical implications: The proposed algorithm can reduce re-decomposition cost when tensors are added to a given tensor by decomposing them based on existing decomposition results without re-decomposing the entire tensor.

Originality/value: The proposed method can handle large tensors and is fast within the limited-memory framework of Apache Spark. Moreover, InParTen2 can handle static as well as incremental tensor decomposition.

Keywords PARAFAC; Tensor decomposition; Incremental tensor decomposition; Apache Spark; Big data

Citation: Yang, Hye-Kyung, and Hwan-Seung Yong. “Multi-aspect incremental tensor decomposition based on distributed in-memory big data systems.” *Journal of Data and Information Science*, vol. 5, no. 2, 2020, pp. 13–32. <https://doi.org/10.2478/jdis-2020-0010>

Received: Oct. 30, 2019

Revised: Feb. 22, 2020

Accepted: Mar. 6, 2020

[†] Corresponding author: Hwan-Seung Yong (E-mail: hsyong@ewha.ac.kr).

1 Introduction

Recommender systems based on multi-dimensional matrices, that is, tensors, have recently attracted significant attention, as they can find more latent factors and patterns compared with conventional systems. However, real-world tensor datasets are sparse and large. Further, tensors are more complicated to analyze than matrices. Therefore, tensor decomposition algorithms, such as parallel factor analysis (PARAFAC) decomposition, Tucker decomposition, and high order singular value decomposition (HOSVD), are used to analyze tensors (Yang & Yong, 2019b). These algorithms can find latent patterns and factors in tensors, but they are limited to static tensor datasets. Specifically, in the case of dynamic tensor datasets or tensor streaming, that is, when new tensors are added to an existing tensor, these algorithms should re-decompose all the tensors because the original decomposition results change. Unfortunately, tensor decomposition is expensive owing to its computational complexity and the large number of iterations involved. Recently, incremental tensor decomposition algorithms, such as online CANDECOMP/PARAFAC (CP) (Zhou, Erfani, & Bailey, 2018) and SamBaTen (Gujral, Pasricha, & Papalexakis, 2018), have been developed; these algorithms assume that only the time axis increases. For example, if a three-dimensional tensor is composed of users, items, time indexes, and rating values, the user and item indexes do not increase. However, the number of users and items generally increase in the real world. Few incremental tensor decomposition methods that consider this multiaxial growth have been proposed. One of these algorithms is a multi-aspect dynamic tensor completion method called MAST (Song et al., 2017). In MAST, new tensors are partitioned into sub-tensors and are subsequently decomposed. Partitioning into sub-tensors affects the results of adjacent parts, and sub-tensor decomposition should be performed sequentially. Therefore, we propose a method for reducing re-decomposition cost without partitioning new tensors into sub-tensors, where all dimensions increase. The computation uses the indexes of an added tensor and existing decomposition results. To this end, we propose InParTen2, which is a distributed multi-aspect incremental PARAFAC three-dimensional tensor decomposition algorithm. PARAFAC is a widely used tensor decomposition algorithm, which, in this study, was transformed into an incremental decomposition algorithm. Most tensor decomposition algorithms are based on MATLAB or Hadoop. MATLAB-based algorithms run on a single machine, and therefore they cannot handle large tensors. The MATLAB method by Song et al. (2017) cannot handle large data either. Hadoop-based algorithms have longer execution time because Hadoop has high disk I/O cost when it performs iterative operations such as tensor decomposition. We use Apache Spark, which is a distributed in-memory big-data system and is faster than

Hadoop for iterative operations (Zaharia et al., 2012). However, Apache Spark runs on limited in-memory systems; therefore, we used it with the improved form of the PARAFAC decomposition algorithm.

The proposed algorithm decomposes incoming tensors using existing decomposition results without generating sub-tensors. In addition, we propose a memory-efficient method to reduce the intermediate data explosion that occurs during PARAFAC decomposition. This method can handle large tensors and is fast within the limited-memory framework of Apache Spark. Moreover, InParTen2 can handle static as well as incremental tensor decomposition.

We evaluated the performance of InParTen2 by comparing its execution time and accuracy with those of existing distributed tensor decomposition methods based on Apache Spark using various tensor datasets. The results confirm that InParTen2 reduces execution time, and its accuracy is similar to that of the other methods.

The rest of the paper is organized as follows. Section 2 provides relevant notations and definitions. Existing PARAFAC tensor decomposition and incremental tensor decomposition algorithms are also discussed in this section. The proposed algorithm is described in Section 3, and the experimental results are presented in Section 4. Section 5 concludes the paper.

2 Preliminaries and related work

In this section, we present relevant notation and tensor matricization concepts. Existing PARAFAC tensor decomposition and incremental tensor decomposition algorithms are also discussed in this section. Table 1 lists the symbols used in this paper.

Table 1. Table of symbols.

Symbols	Definitions
X	Tensor
$X_{(n)}$	Mode- n unfolding matrix of tensor X
$\ X\ _F$	Frobenius norm of tensor X
$\text{nnz}(X)$	Number of non-zero elements in tensor X
$\otimes$	Kronecker product
$\odot$	Khatri-Rao product
$*$	Hadamard product (Element-wise product)
$\dagger$	Pseudo inverse

2.1 Notations and tensor matricization

Before describing PARAFAC decomposition, we will present the relevant notations. The Kronecker product is denoted as $\otimes$. Given two matrices A and B , where the size of A is $I \times K$ and the size of B is $J \times R$, the Kronecker product

Research Paper

generates a matrix of size $IJ \times KR$. The Khatri–Rao product is the column-wise Kronecker product of two matrices with the same number of columns. Given two matrices A and B , where the size of A is $I \times R$ and that of B is $J \times R$, the Khatri–Rao product $A \odot B$ generates a matrix of size $IJ \times R$. The Hadamard product is the element-wise product, denoted as $A * B$, where the two matrices have the same size.

We will now describe tensor matricization, which unfolds a tensor into matrix format. Figure 1 shows a representative example, where a three-dimensional tensor X of size $I \times J \times K$ is unfolded into three matrices. The unfolded matrices along the I , J , and K axes are denoted by $X_{(1)}$, $X_{(2)}$, and $X_{(3)}$, respectively. The sizes of the three unfolded matrices are $X_{(1)} \in \mathbb{R}^{I \times JK}$, $X_{(2)} \in \mathbb{R}^{J \times IK}$, and $X_{(3)} \in \mathbb{R}^{K \times IJ}$.

Figure 1. Schematic representation of tensor matricization.

However, when a new tensor is added (i.e., streaming or dynamic tensors), the total size of the tensor changes and the results of tensor matricization also change. Figure 2 shows the effect on matricization of the addition of a new tensor.

Figure 2. Schematic representation of incremental tensor matricization.

As tensors generally grow over time, in Figure 2, it is assumed that the K axis (time axis) is increasing and the other axes can also increase. The gray regions represent new values corresponding to the arrival of a new tensor. When the entire tensor is unfolded, the new tensor is located behind or under the existing tensor. Therefore, there is a need for a method in which only newly added tensors are decomposed, and the results of existing tensor decompositions are updated. The proposed method decomposes new values without decomposing the entire tensor; thereby, existing tensors are not unfolded, and new unfolded tensors are added to the existing matrix. The proposed method uses tensor indexing to concatenate new tensors and existing results.

2.2 PARAFAC decomposition

PARAFAC decomposition, also known as CP decomposition, is one of the most widely used tensor decomposition algorithms. It decomposes a tensor into a sum of rank-one tensors (Gujral et al., 2018; Jeon et al., 2015; Kang et al., 2012; Yang & Yong, 2019a). A given three-dimensional tensor X is decomposed into R components to yield three factor matrices. This can be expressed as follows:

$$X \approx \sum_{r=1}^R \lambda_r a_r \circ b_r \circ c_r \quad (1)$$

The alternating least squares (ALS) method is used for computing the factor matrices in PARAFAC decomposition. In the case of a three-dimensional tensor, this method fixes two factor matrices to obtain the third factor matrix (Yang & Yong, 2019a; 2019b). The PARAFAC-ALS method for computing the factor matrices is expressed as follows:

$$\begin{aligned} A &= X_{(1)} (C \odot B) (C^\top C * B^\top B)^\dagger \\ B &= X_{(2)} (C \odot A) (C^\top C * A^\top A)^\dagger \\ C &= X_{(3)} (B \odot A) (B^\top B * A^\top A)^\dagger \end{aligned} \quad (2)$$

In Equation (2), $X_{(1)}(C \odot B)$, $X_{(2)}(C \odot A)$, and $X_{(3)}(B \odot A)$ (i.e., the Khatri–Rao products) lead to intermediate data explosions. To alleviate this, several methods have been proposed for avoiding the calculation of Khatri–Rao products. For example, Kang et al. (2012), Jeon et al. (2015), and Park et al. (2016) calculated only non-zero values and avoided direct Khatri–Rao product calculations by multiplying the column vectors of the factor matrices. These methods frequently use map-and-reduce processes for each operation, and thus excessive shuffling may result in slow execution. Papalexakis, Faloutsos, and Sidiropoulos (2012)

used random sampling for tensor decomposition. Jeon et al. (2016) proposed SCouT, which is a large and scalable coupled matrix–tensor factorization algorithm. However, all these methods are based on Hadoop. As Hadoop-based tensor decomposition methods are not suitable for iterative operations, several methods based on Spark have been proposed. For example, the algorithms in Yang and Yong (2018, 2019a) are both based on Apache Spark and reduce the required number of map-and-reduce operations. In the method in Yang and Yong (2018), memory overflow occurs when big data are processed in small distributed systems. This issue is resolved in Yang and Yong (2019a), where tensors are divided into sub-tensors, which are subsequently decomposed. In addition to tensor decomposition, methods for tensor completion are being extensively investigated. Ge et al. (2018) proposed a method called DisTenC; it is a distributed large-scale tensor completion algorithm based on Apache Spark. However, the main goals of tensor decomposition significantly differ from those of tensor completion. Tensor completion is based on low-rank approximations to obtain an accurate estimate of missing values in tensors, whereas tensor decomposition focuses on latent patterns and factors. Therefore, our study focuses on tensor decomposition rather than tensor completion.

2.3 Incremental tensor decomposition

Incremental tensor decomposition algorithms are used to decompose dynamic tensors. Recalculation time can be reduced by such algorithms. Most incremental tensor decomposition algorithms assume that only one axis increases over time. The most representative example is online CP (Zhou et al., 2018), which is an incremental CP decomposition algorithm in which it is assumed that when new tensors are added to an existing tensor, only the time axis increases. Ma, Yang, and Wang (2018) proposed the randomized online CP decomposition algorithm, which is based on random sampling. This algorithm avoids calculating Khatri-Rao products and reduces memory usage. However, the algorithms mentioned above are based on MATLAB routines running on a single machine. Recently, distributed incremental tensor decomposition methods have been introduced. Gujral et al. (2018) proposed a sampling-based batch incremental CP tensor decomposition algorithm, which has been implemented in both MATLAB (Gujral et al., 2018) and Apache Spark (this can be download from <https://github.com/lucasjliu/SamBaTen-Spark>). The Spark version is a re-implementation of the MATLAB version. Yang and Yong (2019b) proposed InParTen, which is also an incremental PARAFAC decomposition algorithm based on Apache spark. Yang and Yong (2019b) and Gujral et al. (2018) both assume that a tensor grows only along one axis. However, real-world tensors can grow along all axes.

Recently, a few incremental tensor decomposition methods that consider multiaxial growth have been developed. Song et al. (2017) proposed a multi-aspect dynamic tensor completion framework called MAST which is based on CP decomposition. This method decomposes added tensors by partitioning them into sub-tensors. This affects the results of adjacent parts, implying that sub-tensor decomposition must be performed sequentially. Najafi, He, and Yu (2019) proposed an outlier-robust multi-aspect streaming tensor completion and factorization algorithm called OR-MSTC (Najafi et al., 2019). The incremental tensor decomposition method in OR-MSTC generates sub-tensors from incoming tensors. Nimishakavi et al. (2018) proposed a side-information-infused incremental tensor analysis method called SIITA which incorporates side information and can process general incremental tensors by Tucker decomposition. However, this method cannot handle scalable tensor datasets because it does not consider scalability issues (Nimishakavi et al., 2018). Additionally, most multi-aspect tensor decomposition algorithms either are based on MATLAB or focus on tensor completion.

In this study, we developed an incremental tensor decomposition algorithm called InParTen2. It considers both dynamic and static tensor datasets. Furthermore, the proposed method is implemented in Apache Spark and can therefore handle distributed in-memory big-data systems without dividing new tensors into sub-tensors. It can process new tensors based on their indexes and the results of previous tensor decompositions. Additionally, InParTen2 avoids the calculation of Khari–Rao products and minimizes shuffling by using the Apache Spark platform. Table 2 provides a summary of various incremental tensor decomposition algorithms.

Table 2. Summary of different incremental PARAFAC tensor decomposition algorithms.

	Static tensor decomposition	Incremental (Single-aspect)	Incremental (Multi-aspect)	Distributed	Scalability
Zhou et al. (2018)		√			
Ma et al. (2018)		√			
Gujral et al. (2018)		√		√	
Yang and Yong (2019b)		√		√	√
Song et al. (2017)		√	√		
Najafi et al. (2019)		√	√		
InParTen2	√	√	√	√	√

3 Methodology

In this section, we describe InParTen2. We assume that an existing tensor increases in all dimensions over time. Figure 3 shows a schematic representation of the multi-aspect incremental tensor decomposition process.

Figure 3. Schematic representation of the multi-dimensional incremental PARAFAC tensor decomposition method.

Here, the size of an existing tensor X_{old} and the new tensor X_{new} is $I \times J \times K$ and $(I + I') \times (J + J') \times K'$, respectively, and X is the sum of the two tensors. X_{old} is decomposed into three factor matrices: A_1 , B_1 , and C_1 . The decomposition of X_{new} yields the new factor matrices A_1 , A_2 and B_1 , B_2 , and C_2 . A and B include a portion of the old factor matrices.

X_{new} can be divided into three types of sub-tensors, as shown in Figure 4. Subsequently, each sub-tensor is decomposed as shown in Figures 4(a), 4(b), and 4(c). The decomposition of each sub-tensor affects the decomposition of the others. In Figure 4(a), as only the K axis is increasing in the current sub-tensor, the factor matrices A_1 and B_1 obtained by the decomposition of this sub-tensor have the same size as that of the existing factor matrices A_1 and B_1 . The size of the factor matrix C_2 is different from that of the existing factor matrix C_2 because the matrices have different indexes. However, the old factor matrix C_2 affects the value of the new factor matrix C_2 in Figures 4(b) and 4(c). Additionally, the old factor matrix B_1 affects the value of the new factor matrix B_1 in Figure 4(b). Therefore, the decomposition results of the new tensor affect each new factor matrix and the existing factor matrices. If we sequentially decompose the sub-tensors, Khatri–Rao products should be calculated more than three times to obtain each factor matrix. Therefore, this method is computationally expensive. In the proposed method, we resolve this issue by a decomposition process that does not generate sub-tensors, as well as a technique that avoids Khatri–Rao products.

Figure 4. Factor matrices obtained by the decomposition of the incoming tensors.

3.1 Incremental PARAFAC tensor decomposition method

InParTen2 decomposes new tensors without generating sub-tensors. We first perform initialization prior to executing incremental PARAFAC-ALS. In the decomposition of the added tensors, there are certain parts that affect the existing

factor matrices. Therefore, we add the existing factor matrices when A and B are updated. For example, to update A , we add the existing factor matrix A_1 and the new factor matrix A . This process can be expressed as follows:

$$\begin{aligned} P &= X_{old(1)}(C_1 \odot B_1) \\ P_{new} &= X_{new(1)}(C_2 \odot B_1) \\ P &= P + P_{new} \end{aligned} \quad (3.1)$$

$$\begin{aligned} Q &= C_1^\top C_1 * B_1^\top B_1 \\ Q_{new} &= C_2^\top C_2 * B_1^\top B_1 \end{aligned} \quad (3.2)$$

$$\begin{aligned} Q &= Q + Q_{new} \\ A &= PQ^\dagger \end{aligned} \quad (3.3)$$

The initial values of P and Q are set to $X_{old(1)}(C_1 \odot B_1)$ and $C_1^\top C_1 * B_1^\top B_1$, respectively. Here, we can calculate the Khatri–Rao products by using the existing tensor to obtain an initial value of P . We can avoid repeated calculations by using the existing factor matrices A_1 , B_1 , and C_1 because they were obtained from the existing tensor. This method is expressed by the following equation:

$$X_{old(1)}(C_1 \odot B_1) = A_1(C_1 \odot B_1)^\top (C_1 \odot B_1) = A_1(C_1^\top C_1 * B_1^\top B_1) = A_1 Q \quad (4)$$

To calculate $X_{old(1)}(C_1 \odot B_1)$, we can solve the equation above by multiplying Q by the existing matrix A_1 . Equation (4) can effectively reduce execution time and improve memory efficiency because the existing tensor is not unfolded, and intermediate data explosion is avoided by using the Hadamard product. Time complexity is thus reduced from $O(RIJK)$ to $O(R^2(I + J + K))$. Additional values, such as $X_{old(2)}(C_1 \odot A_1)$ and $X_{old(3)}(B_1 \odot C_1)$, can also be obtained by a similar method. Subsequently, we initialize the factor matrices $C_{updated}$ and A based on the existing factor matrices and unfold the new tensors prior to decomposing them. In the initialization process, two cases are considered. In the first case, all axes increase over time, and in the second, the time axis does not change or exhibits relatively small changes as the other axes increase. We will now explain the initialization process for the first case. In Figure 4 above, all dimensions increase over time. It is evident that the factor matrices obtained from each sub-tensor have overlapping parts: C_2 , A_1 , A_2 , and B_1 . These parts affect each other. In particular, the factor matrix C_2 affects all decomposed sub-tensors in the new tensors. Therefore, the initial value of C_2 can be obtained from the unfolded tensor $X_{new(3)}$ and the existing factor matrices A_1 and B_1 . In this case, the size of K for the added tensor is greater than zero; thus, we initialize $C_{updated}$ to the initial value of C_2 , as shown in Equation (5.1). In the

Research Paper

second case, where the time axis (K axis) does not change or changes little, we initialize $C_{updated}$ using C_1 , as shown in Equation (5.2).

$$C_{updated} = \begin{cases} C_2 = X_{new(3)}(B_1 \odot A_1)(B_1^\top B_1 * A_1^\top A_1)^\dagger & \text{if } K' > 0 \\ C_1 & \text{if } K' \leq 0 \end{cases} \quad (5.1)$$

Subsequently, we solve for A using the unfolded tensors $X_{new(1)}$ and $C_{updated}$ combined with the existing factor matrices B_1 and A_1 as follows:

$$A = A_1 + X_{new(1)}(C_{updated} \odot B_1)(C_{updated}^\top C_{updated} * B_1^\top B_1)^\dagger = (P + P_{new})(Q + Q_{new})^\dagger \quad (6)$$

As mentioned previously, we update the factor matrices A based on Equation (3). However, when the size of the I axis of $X_{new(1)}$ increases by I' , the size of $X_{new(1)}(C_{updated} \odot B_1)$ is $(I + I') \times R$. Therefore, $X_{new(1)}(C_{updated} \odot B_1)$ overlaps with the initial P , and the value of the initial P should be added to that of P_{new} . To this end, we use indexing to add the parts that should be updated in the existing P ; subsequently, the part of size $I' \times R$ is appended to the bottom rows of the initial P because P and $X_{new(1)}(C_{updated} \odot B_1)$ have different size. Additionally, $C_{updated}^\top C_{updated} * B_1^\top B_1$ and the existing Q have the same size, and therefore simple matrix addition can be performed.

Following this initial process, we use the incremental PARAFAC-ALS process to update the factor matrices B , A , and C . This update method can be expressed as follows:

$$\begin{aligned} B &= B_1 + X_{new(2)}(C_{updated} \odot A)(C_{updated}^\top C_{updated} * A^\top A)^\dagger \\ A &= A_1 + X_{new(1)}(C_{updated} \odot B)(C_{updated}^\top C_{updated} * B^\top B)^\dagger \\ C_{update} &= X_{new(3)}(B \odot A)(B^\top B * A^\top A)^\dagger \end{aligned} \quad (7)$$

First, to update the factor matrix B , we use the factor matrices $C_{updated}$ and A from the new tensors. Similarly, to update A , we use indexing to add only the parts that should be updated in the factor matrix B_1 . At the end of the iteration, we finally update the factor matrix C when the value of the initialized $C_{updated}$ is C_2 . In this case, $C_{updated}$ can be sequentially added to the rows of the existing matrix C_1 , as shown in the following equation:

$$C = \begin{bmatrix} C_1 \\ C_{updated} \end{bmatrix} = \begin{bmatrix} C_1 \\ X_{new(3)}(B \odot A)(B^\top B * A^\top A)^\dagger \end{bmatrix} \quad (8)$$

3.2 Implementation of incremental tensor decomposition method in Spark

In this section, we describe the implementation of InParTen2 in Apache Spark to achieve fast and memory-efficient execution. First, we use the coordinate format

to load a new tensor as an input because real-word tensors are generally sparse. The coordinate format is a resilient distributed data (RDD) type consisting of three indexes and values. Subsequently, this format is used to load existing three-factor matrices. These are constructed from row indexes and row vectors. Owing to the limited memory space, only non-zero values of the tensor and three-factor matrices are loaded. Prior to the execution of PARAFAC-ALS, we initialize the factor matrices $C_{updated}$ and A using the existing tensor. This can be carried out based on Equations (5.1) and (6). However, as mentioned previously, computing Khatri–Rao products can lead to intermediate data explosions. We modified the HaTen2-DRI method for implementation in Apache Spark. In HaTen2-DRI, intermediate data explosions occur $2nnz(x)R$ times. However, InParTen2 can reduce this to $2nnz(x)$. Khatri–Rao products can be avoided by using Equation (9). Thereby, all steps involving Khatri–Rao product operations can be omitted.

$$X_{(l)}(C \odot B) = \sum_{(j,k)=(1,1)}^{JK} X_{(l)(i,j,k)} B(j,:) C(k,:) \quad (9)$$

Here, the non-zero tensor values are multiplied by the two-factor matrices corresponding to the non-zero indexes in the tensor. For example, if there is a non-zero value in $X_{(1)(2,1,2)}$, then all column values in row 1 of B and row 2 of C are multiplied. In the parts where tensor decomposition involving Khatri–Rao products was initially required, we use Equation (7) instead. We use the RDD functions *join*, *map*, and *reduceByKey* to implement Equation (7). To update A and B , the existing factor matrix should be added. As mentioned previously, we do not generate a zero matrix; rather, we use indexing to add to the values of the existing factor matrix. If the existing factor matrix does not require updating, the addition operation is not performed. These processes use the *join* and *map* functions.

Overall, the time and space complexity of InParTen2 is $O((6nnz(X_{new})+R^2((I+I')+(J+J')+K')))$ and $O(R(I+J+K')+2nnz(X_{new}))$, respectively. Where $nnz(X_{new})$ is the number of non-zero in new tensor, and R is the decomposition rank.

4 Experiment and results

In this section, we evaluate the performance of InParTen2 by comparing its execution time and accuracy with those of existing tensor decomposition methods using various tensor datasets. We consider execution time and relative error under the following scenarios: different density using synthetic tensors, increasing data size using synthetic tensors, and real-world datasets.

We used the experimental environments Hadoop v2.7 and Apache Spark v2.2. The experiments were conducted using six node workers with a total memory of 48 GB.

4.1 Tensor datasets

We used synthetic and real-world datasets to evaluate InParTen2. We used 10 types of synthetic datasets for each tensor size. In the synthetic datasets, we generated random tensors of dimension $I = J = K$, where $I \in [100, 500, 800]$, and used different initialized densities: 20% and 60% in each initial tensor. Further, we generated incoming tensors with densities 5%, 10%, and 20% in the initial tensors, which are half of that in the total tensor. In addition, we used different initial size and density.

We used three real-world datasets containing sparse three-dimensional tensors: YELP (Jeon et al., 2016), MovieLens (Jeon et al., 2016), and Netflix (Bennett & Lanning, 2007). Table 3 lists these datasets. YELP contains users, locations, time indexes, and rating values from 1 to 5. MovieLens and Netflix contain users, movies, time indexes, and rating values from 1 to 5. In the incremental tensor decomposition by InParTen2, it is assumed that the three dimensions increase, and the initial dimension is fixed. In the SamBaTen (Gujral et al., 2018) algorithm, which considers the increase in a single dimension, only the time axis of the initial dimension is set to be equal to that of InParTen2.

Table 3. Real-world tensor datasets.

Data Name	Data type	Total Dimension	Initial Dimension	Non-zero	File size
YELP	User×Location×Time	70,817×15,579×108	70,815×15,572×20	334,166	5.6MB
MovieLens	User×Movie×Time	71,567×10,681×157	71,559×9,717×5	10,000,054	187MB
Netflix	User×Movie×Time	2,649,429×17,770×2,182	2,648,623×17,764×50	98,754,343	1.8GB

4.2 Evaluation measures

We compared the execution time and accuracy of InParTen2 with those of existing tensor decomposition methods based on Apache Spark. Execution time is an important measure because one of our goals is to reduce time cost. We compared the execution time using synthetic and real-world datasets. In the former case, we performed various measurements. In the first measurement, the initial tensors were set to a density of 20% and 60%, and the incoming tensors had densities of 5%, 10%, and 20%. This experiment was conducted to study the effect of the density of the incoming data on performance. In the second measurement, the size of the initial tensor and incoming tensors were varying. Here, the density of the incoming tensor was set to 20%, and that of the initial tensor to 20% and 60%. In this experiment, the incoming data had the same density, but performance was evaluated in terms of the size of the incoming tensor data. In the real-world datasets, we set the initial size. We tested 10 iterations and 10 ranks in each dataset.

The second comparison is based on accuracy. We reconstructed tensors through decomposition and used the relative error to investigate their similarity to the

original tensors. We compared these results to those obtained by existing tensor decomposition methods, particularly static tensor decomposition. Relative error can be defined as follows:

$$Relative\ Error = \frac{\|\hat{X}_{reconstruct} - X_{original}\|_2}{\|X_{original}\|_2} \quad (10)$$

where $\hat{X}_{reconstruct}$ denotes the reconstructed tensor using the decomposition results. A small relative error implies better approximation.

4.3 Existing tensor decomposition methods

We used Spark-based tensor decomposition methods. There are several Hadoop-based distributed tensor decomposition algorithms, such as HaTen2 (Jeon et al., 2015) and BigTensor (Park et al., 2016), as well as MATLAB-based decomposition tools. However, Hadoop-based decomposition tools require longer iteration time, and therefore their execution time cannot be compared with that of Spark-based algorithms. MATLAB-based algorithms run on a single machine, thus limiting their ability to handle large data. Therefore, we used IM-PARAFAC and SamBaTen (Gujral et al., 2018), which run in the Apache Spark framework, for the comparison. IM-PARAFAC (Yang & Yong, 2019a) can decompose static tensors and handle large datasets. SamBaTen can decompose both static and incremental tensors. The incremental tensor decomposition of SamBaTen considers the increase in the time-axis only; accordingly, the initial size of the tensor datasets was set only for the time axis. The incremental tensor decomposition of SamBaTen is based on sampling. However, in the case of the static CP-ALS provided by SamBaTen, the HaTen2-Naive algorithm was re-implemented instead of the sampling algorithm. As InParTen2 also performs both static and incremental decomposition, both methods were tested. In the tests, the static tensor decomposition of InParTen2 is denoted as InParTen2-Static.

4.4 Evaluation results

Initially, we considered the case in which initial tensors were set to a density of 20% and 60%, and the incoming tensors to 5%, 10%, and 20%. This test demonstrated the effect of density on computational cost for fixed size. For this experiment, the size of the initial tensor was set to half of the total tensor size. Table 4 and Figure 5 show the comparison of execution time for different densities of the incoming tensors. The best results are highlighted. Overall, it was demonstrated that sparse datasets exhibit shorter execution time. Further, InParTen2 was faster in every respect. InParTen2 expediently handled both static and incremental tensor decomposition. The latter was at least twice as fast as that by SamBaTen.

Research Paper

Table 4. Comparison of execution time (min) according to the different densities of incoming tensor.

Tensor Synthetic Datasets			100 × 100 × 100			500 × 500 × 500			800 × 800 × 800		
Density(%)			5%	10%	20%	5%	10%	20%	5%	10%	20%
Initial Density 60%	Static	IM-PARAFAC	1.19	1.37	1.62	78.5	95.5	131.5	318.9	402.3	543.6
		SamBaTen (CP-ALS)	1.55	1.74	1.98	134.9	183.8	211.9	657.6	836.2	1323.9
		InParTen2-Static	<u>0.56</u>	<u>0.64</u>	<u>0.77</u>	<u>26.9</u>	<u>34.2</u>	<u>48.3</u>	<u>106.2</u>	<u>130.2</u>	<u>177.6</u>
	Dynamic	SamBaTen	1.07	1.07	1.13	35.5	45.7	66.5	301.5	354.2	477.2
		InParTen2	<u>0.5</u>	<u>0.46</u>	<u>0.57</u>	<u>10.9</u>	<u>17.6</u>	<u>29.8</u>	<u>37.18</u>	<u>60.95</u>	<u>111.2</u>
Initial Density 20%	Static	IM-PARAFAC	0.87	0.99	1.27	40.9	58.5	98.03	167.9	246.3	397.8
		SamBaTen (CP-ALS)	1.42	1.48	1.98	41.9	116.2	194.66	345.6	554.3	973.1
		InParTen2-Static	<u>0.64</u>	<u>0.72</u>	<u>0.86</u>	<u>14.9</u>	<u>21.4</u>	<u>35.6</u>	<u>59.8</u>	<u>84.32</u>	<u>108.1</u>
	Dynamic	SamBaTen	0.92	0.96	1.04	18.3	24.9	35.2	87.7	125.9	250.7
		InParTen2	<u>0.36</u>	<u>0.44</u>	<u>0.54</u>	<u>9.3</u>	<u>16.7</u>	<u>24.1</u>	<u>30.2</u>	<u>51.5</u>	<u>95.4</u>

Figure 5. Comparison of relative errors according to the different densities of incoming tensor datasets.

Table 5 shows the comparison of the relative errors for different densities of the incoming tensor datasets. InParTen2 exhibits similar relative errors to those of the

other methods. In general, datasets with an initial density of 60% exhibit better relative error than those with an initial density of 20%. If the original datasets are sparse, the reconstructed tensor is filled with values, thus increasing the relative error.

Table 5. Comparison of relative errors according to the different densities of incoming tensor.

Tensor Synthetic Datasets			100 × 100 × 100			500 × 500 × 500			800 × 800 × 800		
Density			5%	10%	20%	5%	10%	20%	5%	10%	20%
Initial Density 60%	Static	IM-PARAFAC	0.63	0.67	0.69	0.64	0.72	0.73	0.62	0.70	0.70
		SamBaTen (CP-ALS)	0.65	0.70	0.69	0.66	0.71	0.72	0.66	0.71	0.72
		InParTen2-Static	0.63	0.67	0.71	0.60	0.73	0.72	0.61	0.69	0.72
	Dynamic	SamBaTen (Incremental)	0.67	0.69	0.78	0.67	0.75	0.80	0.67	0.74	0.79
		InParTen2	0.61	0.67	0.69	0.65	0.73	0.71	0.63	0.72	0.72
Initial Density 20%	Static	IM-PARAFAC	0.87	0.87	0.82	0.90	0.89	0.83	0.89	0.88	0.83
		SamBaTen (CP-ALS)	0.88	0.87	0.82	0.89	0.89	0.83	0.89	0.89	0.83
		InParTen2-Static	0.87	0.87	0.82	0.87	0.89	0.83	0.90	0.88	0.83
	Dynamic	SamBaTen (Incremental)	0.88	0.89	0.85	0.91	0.93	0.98	0.89	0.93	0.88
		InParTen2	0.86	0.89	0.82	0.91	0.90	0.84	0.87	0.90	0.86

Subsequently, we compared the execution time and relative error when data size increased. We considered initial tensors of different sizes under the same tensor size. Here, the density of the initial tensor was set to 20% and 60%, and that of the incoming tensors to 20%. Table 6 and Figure 6 show the comparison of the execution time for different initial size. The best results are highlighted. When the initial density was 60%, the execution time of the static decomposition tools increased gradually, but the dataset with an initial density of 20% exhibited the best performance when the initial size was half. The incremental decomposition by SamBaTen exhibited increased execution time. When the initial density was 60%, its execution time increased with the initial dimension, but when the initial density was 20%, the execution time decreased as the initial dimension increased. However, the incremental tensor decomposition by InParTen2 was faster regardless of the initial density when the initial dimension was larger and the amount of added data was small.

Table 7 shows the comparison of the relative errors for different initial dimensions. It can be seen that the relative errors by InParTen2 and the other tensor decomposition tools are similar. Overall, the relative error increases with density. Datasets with an initial density of 20% were overall sparse, and therefore they had similar accuracy regardless of the initial size.

Research Paper

Table 6. Comparison of execution time (min) according to the different initial size.

Tensor Synthetic Datasets			100×100×100			500×500×500			800×800×800		
Initial dimension size (I=J=K)			10	50	80	100	250	400	100	400	640
Initial Density 60%	Static	IM-PARAFAC	1.76	1.62	2.66	126.84	131.49	270.04	540.79	543.60	1050.21
		SamBaTen (CP-ALS)	1.97	1.98	2.77	235.55	211.94	553.88	1398.33	1323.99	2416.36
		InParTen2-Static	0.74	0.77	1.08	36.94	48.28	72.14	180.63	177.64	357.48
	Dynamic	SamBaTen (Incremental)	1.16	1.13	1.21	42.97	66.47	89.53	351.21	477.21	652.03
		InParTen2	<u>0.68</u>	<u>0.57</u>	<u>0.46</u>	<u>37.08</u>	<u>29.79</u>	<u>17.11</u>	<u>185.99</u>	<u>111.21</u>	<u>83.01</u>
Initial Density 20%	Static	IM-PARAFAC	1.76	1.27	1.39	125.80	98.03	110.03	570.94	397.80	459.99
		SamBaTen (CP-ALS)	1.97	1.98	2.72	264.09	194.66	265.96	1403.27	973.12	990.81
		InParTen2-Static	0.97	0.86	0.88	46.03	35.58	39.55	146.53	108.07	119.39
	Dynamic	SamBaTen (Incremental)	1.19	1.04	0.99	39.69	35.18	30.91	293.2	250.71	182.81
		InParTen2	<u>0.67</u>	<u>0.54</u>	<u>0.44</u>	<u>37.17</u>	<u>24.11</u>	<u>12.40</u>	<u>158.21</u>	<u>95.45</u>	<u>46.8</u>

Figure 6. Comparison of execution time (min) according to the different initial size.

Table 7. Comparison of relative errors according to the different initial size.

Tensor Synthetic Datasets			100 × 100 × 100			500 × 500 × 500			Tensor 800 × 800 × 800		
Initial size (I=J=K)			10	50	80	100	250	400	100	400	640
Initial Density 60%	Static	IM-PARAFAC	0.82	0.69	0.57	0.82	0.73	0.57	0.83	0.70	0.57
		SamBaTen (CP-ALS)	0.82	0.71	0.56	0.82	0.72	0.57	0.83	0.72	0.57
		InParTen2-Static	0.82	0.69	0.56	0.82	0.72	0.57	0.82	0.72	0.56
	Dynamic	SamBaTen (Incremental)	0.85	0.78	0.57	0.83	0.80	0.57	0.83	0.79	0.58
		InParTen2	0.82	0.69	0.56	0.82	0.71	0.57	0.83	0.72	0.56
	Initial Density 20%	IM-PARAFAC	0.82	0.82	0.82	0.83	0.83	0.83	0.83	0.83	0.83
		SamBaTen (CP-ALS)	0.82	0.82	0.82	0.82	0.83	0.83	0.83	0.83	0.83
		InParTen2-Static	0.82	0.82	0.82	0.83	0.83	0.83	0.83	0.83	0.83
		Dynamic	SamBaTen (Incremental)	0.84	0.85	0.83	0.85	0.98	0.87	1.02	0.88
			InParTen2	0.82	0.82	0.83	0.82	0.84	0.84	0.84	0.84

Finally, we compared the execution time and relative error using real-world datasets. Table 8 shows the results. It can be seen that InParTen2 is overall faster. InParTen2-static is approximately 1.03 to 3.6 times as fast as IM-PARAFAC, and 17 times as fast as CP-ALS of SamBaTen. InParTen2 exhibits smaller execution time than static tensor decomposition. In addition, it is approximately 1.5 to 18 times as fast as SamBaTen, and unlike SamBaTen, it can handle large tensors. Furthermore, InParTen2 exhibits similar relative errors to those of the other methods. SamBaTen-CPALS and SamBaTen exhibit a large relative error (larger than that of InParTen2) because they sample the additional data. However, InParTen2 has a difference of 0.04 to 0.05 from InParTen2-static and IM-PARAFAC in the MovieLens datasets. This apparently increases the relative error because the number of tensor values is increased to an excessive degree.

Table 8. Comparison of execution time and relative error using real world datasets.

		Execution Time(min)			Relative Error		
		YELP	MovieLens	Netflix	YELP	MovieLens	Netflix
Static	IM-PARAFAC	7.44	76.03	924.91	0.98	0.75	0.79
	SamBaTen (CP-ALS)	44.71	518.73	N.A.	0.98	0.99	N.A.
	InParTen2-Static	2.047	29.03	893.33	0.96	0.76	0.79
Dynamic	SamBaTen (incremental)	36.42	36.65	N.A.	0.98	0.91	N.A.
	InParTen2	1.94	23.50	246.63	0.97	0.80	0.80

5 Conclusion

In this paper, we proposed InParTen2, which is a distributed and incremental PARAFAC decomposition algorithm for multi-dimensional tensors. InParTen2 can reduce the re-calculation cost associated with the addition of new tensors. Furthermore, its accuracy is similar to that obtained by decomposing the entire tensor. The proposed method considers that all dimensions may grow over time. When new tensors are added, InParTen2 decomposes only the new tensors based on existing factor matrices. We implemented InParTen2 in the Apache Spark platform. InParTen2 uses an incremental tensor decomposition method suitable for Apache Spark so that fast execution may be achieved and large tensors may be handled. Moreover, InParTen2 uses a method for alleviating the memory overflow and data explosion problems that occur in in-memory big-data systems based on tensor decomposition algorithms. The performance of InParTen2 was evaluated by comparing its execution time and relative error with those of existing tensor decomposition methods. The results confirmed that InParTen2 can process large tensors and reduce re-calculation cost. Especially, InParTen2 is possible at a much faster than existing incremental tensor tools and can handle memory efficiency.

In the future, we will study Tucker-based incremental tensor decomposition based on Apache Spark. We will extend the proposed method to tensor completion, which can handle missing tensor entries. Furthermore, we will consider real-world applications, such as recommendation systems and social network analysis.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2016R1D1A1B03931529).

Author Contributions

Hye-Kyung Yang (hyekyung.db@gmail.com) proposed the research problems, algorithm designed, implementation, and performed experiments, and co-wrote the paper. Hwan-Seung Yong (hsyong@ewha.ac.kr) proposed the research problems, algorithm designed, supervised the research and co-wrote the paper.

References

- Bennett, J., & Lanning, S. (2007). The Netflix prize. In Proceedings of the KDD Cup Workshop 2007 (pp. 3–6). ACM. New York, NY, USA. <https://www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf>

- Ge, H., Zhang, K., Alfifi, M., Hu, X., & Caverle, J. (2018). DisTenC: A distributed algorithm for scalable tensor completion on spark. In *Proceedings of the 34th IEEE International Conference on Data Engineering* (pp. 137–148). IEEE. Paris, France. DOI: 10.1109/ICDE.2018.00022
- Gujral, E., Pasricha, R., & Papalexakis, E.E. (2018). SamBaTen: Sampling-based batch incremental tensor decomposition. In *Proceedings of the 2018 SIAM International Conference on Data Mining* (pp. 387–395). SIAM. San Diego Marriott Mission Valley, San Diego, CA, USA. <https://doi.org/10.1137/1.9781611975321.44>
- Jeon, B.S., Jeon, I., Lee, S., & Kang, U. (2016). SCouT: Scalable coupled matrix-tensor factorization—Algorithm and discoveries. In *Proceedings of the 32nd IEEE International Conference on Data Engineering* (pp. 811–822). IEEE. Helsinki, Finland.
- Jeon, I., Papalexakis, E.E., Kang, U., & Faloutsos, C. (2015). HaTen2: Billion-scale tensor decompositions. In *Proceedings of the 31st IEEE International Conference on Data Engineering* (pp. 1047–1058). IEEE. Seoul, South Korea. DOI: 10.1109/ICDE.2015.7113355
- Kang, U., Papalexakis, E., Harpale, A., & Faloutsos, C. (2012). GigaTensor: Scaling tensor analysis up by 100 times—Algorithms and Discoveries. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 316–324). ACM. Beijing, China. <https://doi.org/10.1145/2339530.2339583>
- Ma, C., Yang, X., & Wang, H. (2018). Randomized online CP decomposition. In *Proceedings of the Tenth International Conference on Advanced Computational Intelligence* (pp. 414–419). IEEE. Xiamen, China. DOI: 10.1109/ICACI.2018.8377495
- Najafi, M., He, L., & Yu, P.S. (2019). Outlier-robust multi-aspect streaming tensor completion and factorization. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (pp. 3187–3194). ijcai.org. Macao, China. <https://doi.org/10.24963/ijcai.2019/442>
- Nimishakavi, M., Mishra, B., Gupta, M., & Talukdar, P. (2018). Inductive framework for multi-aspect streaming tensor completion with side information. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (pp. 307–316). ACM. Torino, Italy. <https://doi.org/10.1145/3269206.3271713>
- Papalexakis, E.E., Faloutsos, C., & Sidiropoulos, N.D. (2012). ParCube: Sparse parallelizable tensor decomposition. In *Proceedings of Machine Learning and Knowledge Discovery in Databases—European Conference* (pp. 521–536). Springer. Bristol, UK. DOI: 10.1007/978-3-642-33460-3_39
- Park, N., Jeon, B., Lee, J., & Kang, U. (2016). BIGtensor: Mining billion-scale tensor made easy. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management* (pp. 2457–2460). ACM. Indianapolis, IN, USA. <http://dx.doi.org/10.1145/2983323.2983332>
- Song, Q., Huang, X., Ge, H., Caverlee, J., & Hu, X. (2017). Multi-aspect streaming tensor completion. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 435–443). ACM. Halifax, NS, Canada. <https://doi.org/10.1145/3097983.3098007>
- Yang, H.K., & Yong, H.S. (2018). S-PARAFAC: Distributed tensor decomposition using Apache Spark. *Journal of Korean Instituted of Information Scientists and Engineers*, 45(3), 280–287.

Research Paper

- Yang, H.K., & Yong, H.S. (2019a). Distributed PARAFAC decomposition method based on in-memory big data system. In Proceedings of Database Systems for Advanced Applications 24th International Conference (pp. 292–295). Springer. Chiang Mai, Thailand. https://doi.org/10.1007/978-3-030-18590-9_31
- Yang, H.K., & Yong, H.S. (2019b). Incremental PARAFAC decomposition for three-dimensional tensors using Apache Spark. In Proceedings of Web Engineering-19th International Conference (pp. 53–71). Springer. Daejeon, South Korea. https://doi.org/10.1007/978-3-030-19274-7_5
- Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M.,..., & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (pp. 15–28). USENIX. San Jose, CA, USA. <https://dl.acm.org/doi/10.5555/2228298.2228301>
- Zhou, S., Erfani, S.M., & Bailey, J. (2018). Online CP decomposition for sparse tensors. In Proceedings of the IEEE International Conference on Data Mining (pp. 1458–1463). IEEE. Singapore. DOI: 10.1109/ICDM.2018.00202

This is an open access article licensed under the Creative Commons Attribution-NonCommercial-NoDerivs License (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).