Accelerating Online CP Decompositions for Higher Order Tensors

Shuo Zhou[1], Nguyen Xuan Vinh[1], James Bailey[1], Yunzhe Jia[1], Ian Davidson[2]
[1]Dept. of Computing and Information Systems, The University of Melbourne, Australia
[1]{zhous@student., vinh.nguyen@, baileyj@, yunzhej@student.}unimelb.edu.au
[2]Dept. of Computer Science, University of California, Davis, USA
[2]davidson@cs.ucdavis.edu

ABSTRACT

Tensors are a natural representation for multidimensional data. In recent years, CANDECOMP/PARAFAC (CP) decomposition, one of the most popular tools for analyzing multi-way data, has been extensively studied and widely applied. However, today's datasets are often dynamically changing over time. Tracking the CP decomposition for such dynamic tensors is a crucial but challenging task, due to the large scale of the tensor and the velocity of new data arriving. Traditional techniques, such as Alternating Least Squares (ALS), cannot be directly applied to this problem because of their poor scalability in terms of time and memory. Additionally, existing online approaches have only partially addressed this problem and can only be deployed on third-order tensors. To fill this gap, we propose an efficient online algorithm that can incrementally track the CP decompositions of dynamic tensors with an arbitrary number of dimensions. In terms of effectiveness, our algorithm demonstrates comparable results with the most accurate algorithm, ALS, whilst being computationally much more efficient. Specifically, on small and moderate datasets, our approach is tens to hundreds of times faster than ALS, while for large-scale datasets, the speedup can be more than 3,000 times. Compared to other state-of-the-art online approaches, our method shows not only significantly better decomposition quality, but also better performance in terms of stability, efficiency and scalability.

Keywords

Tensor Decomposition; CP Decomposition; Online Learning

1. INTRODUCTION

Numerous types of data are naturally represented as multi-dimensional structures. The *Tensor*, a multi-way generalization of the matrix, is useful for representing such data. Similar to matrix analysis tools, such as PCA and SVD, *tensor decomposition* (TD) is a popular approach for feature extraction, dimensionality reduction and knowledge discovery on multi-way data. It has been extensively studied and widely applied in various fields of science, including chemometrics [1], signal processing [7], computer vision [14, 29], graph and network analysis [10, 17] and time series analysis [5].

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

As TD is usually the first and necessary step for analyzing multi-way data, in this work, we aim to address the problem of how to adaptively track the decompositions for such time-evolving tensors. Specifically, we are particularly interested in dynamic tensors that are incrementally growing over time, while the other dimensions remain unchanged. These are the most common type of dynamic tensors that occur in practice. We refer to such tensors as *online tensors*, also known as *tensor streams* and *incremental tensors* [27, 28].

Finding the decompositions for large-scale online tensors is challenging. The difficulty mainly arises from two factors. First, as online tensors are growing with time, their overall size is potentially unbounded. Thus, TD techniques for such tensors need to be highly efficient and scalable, from both time and space perspectives. Second, a high data generation rate demands decomposition methods providing real-time or near real-time performance [6]. However, traditional TD techniques, such as *Tucker* and *CANDECOMP/PARAFAC* (CP) decompositions, cannot be directly applied to this scenario because: (i) they require the availability of the full data for the decomposition, thus having a large memory requirement; and (ii) their fitting algorithms, *Higher-Order SVD* (HOSVD) for Tucker [9], and *Alternating Least Squares* (ALS) or other variants for CP [8], are usually computationally too expensive for large-scale tensors.

A recipe for addressing the above challenges is to adapt existing approaches using online techniques. In recent years, several studies have been conducted on tracking the Tucker decomposition of online tensors by incorporating online techniques, such as incremental SVD [14, 18, 26], and incremental update of covariance matrices [27, 28]. However, there is a limited amount of work on tracking the CP decomposition of an online tensor. The only work in the literature, pro-

KDD '16, August 13-17, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-4232-2/16/08...$15.00
DOI: http://dx.doi.org/10.1145/2939672.2939763

posed by Nion and Sidiropoulos [20], specifically only deals with third-order tensors and there is no provision for higher-order tensors with more than 3 dimensions. To fill this gap, we propose an efficient algorithm to find the CP decomposition of large-scale high-order online tensors, with low space and time usage. We summarize our contributions as follows:

- We propose a scalable algorithm for efficiently tracking the CP decompositions of online tensors. Not limited to basic third-order tensors, our model can also handle higher-order tensors that have more than 3 dimensions.
- Through experimental evaluation on seven real world datasets, we show that our approach can provide more accurate results and better efficiency, compared with state-of-the-art approaches.
- Based on empirical analysis on synthetic datasets, our algorithm produces more stable decompositions than existing online approaches, as well as better scalability.

The rest of this paper is organized as follows. Section 2 gives a review of current techniques and Section 3 introduces background knowledge. Our proposal is discussed in Section 4. We start from third-order tensors and then extend this model to general tensors that have an arbitrary number of dimensions. After that, the performance of our approach is evaluated on both real world and synthetic datasets in Section 5. Lastly, Section 6 concludes and discusses future research directions.

2. RELATED WORK

The problem of decomposing online tensors was originally proposed by Sun *et al.* [27, 28], wherein they refer to this problem as *Incremental Tensor Analysis* (ITA). Three variants of ITA are discussed in their work: (1) dynamic tensor analysis (DTA) modifies the covariance matrices calculation step in typical HOSVD in an incremental fashion; (2) stream tensor analysis (STA) is an approximation of DTA by the SPIRIT algorithm [21]; and (3) window-based tensor analysis (WTA) uses a sliding window strategy to improve the efficiency of DTA. The main issue of these techniques is that they have not fully optimized the most time consuming step, i.e., diagonalizing the covariance matrix for each mode, which limits their efficiency. To overcome this issue, Liu *et al.* [17] propose an efficient algorithm that enforces the diagonalization on the core tensors only. Additionally, another trend for improving the efficiency of HOSVD on online tensors is to replace SVD with incremental SVD algorithms. Several applications of this idea can be found in computer vision [14, 18, 26] and anomaly dectection [25] fields.

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

Unlike the existing extensive studies on online Tucker decomposition, there is a limited research reported for online CP decomposition. The most related work to ours was proposed by Nion and Sidiropoulos [20], which introduced two adaptive algorithms that specifically focus on CP decomposition: Simultaneous Diagonalization Tracking (SDT) that incrementally tracks the SVD of the unfolded tensor; and Recursive Least Squares Tracking (RLST), which recursively updates the decomposition factors by minimizing the mean squared error. However, the major drawback of this work is that they work on third-order tensors only, while in contrast we propose a general approach that can incrementally track the CP decompositions of tensors with arbitrary dimensions.

In another related area, among the studies that focus on improving CP decomposition for handling large-scale tensors, GridTF, a grid-based tensor factorization algorithm [22] is particularly related to our problem. The main idea of GridTF is to partition the large tensor into a number of small grids. These grids are then factorized by a typical ALS algorithm in parallel. Finally, the resulting decompositions of these sub-tensors are combined together using an iterative approach. In fact, as stated by the authors, if such partitioning is enforced on the time mode only, then GridTF can be used for tracking the CP decomposition of tensor streams.

3. PRELIMINARIES

3.1 Notation and Basic Operations

Following the notation in [16], vectors are denoted by boldface lowercase letters, e.g., $\mathbf{a}$, matrices by boldface uppercase letters, e.g., $\mathbf{A}$, and tensors by boldface Euler script letters, e.g., $\boldsymbol{\mathcal{X}}$. The *order* of a tensor, also known as the number of *ways* or *modes*, is its number of dimensions. For example, vectors and matrices are tensors of order 1 and 2. A tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is an N^{th}-order one consisting of real numbers and the cardinality of its i-th order, $i \in [1, N]$ is I_i. We refer to tensors with more than 3 modes as *higher*-order ones. The elements of a tensor are retrieved by their indices, and a *slice* of an N^{th}-order tensor is an $(N-1)^{th}$-order tensor with the index of a particular mode fixed.

Let $\mathbf{A}^\top$, $\mathbf{A}^{-1}$, $\mathbf{A}^\dagger$ and $\|\mathbf{A}\|$ denote the transpose, inverse, Moore-Penrose pseudoinverse and Frobenius norm of $\mathbf{A}$. Let $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$ represent a sequence of N matrices. The *Khatri-Rao* and *Hadamard* products [16] and element-wise division are denoted by $\odot$, $\circledast$ and $\oslash$, respectively. Furthermore, the Khatri-Rao and Hadamard products of a sequence of N matrices $\mathbf{A}^{(N)}, \mathbf{A}^{(N-1)}, \ldots, \mathbf{A}^{(1)}$ are denoted by $\bigodot^N \mathbf{A}^{(i)}$ and $\circledast^N A^{(i)}$. Note that the superscripts in the sequence are inverted and the subscript $i \neq n$ is used when the n-th matrix is not included in above operations, as $\bigodot_{i\neq n}^N \mathbf{A}^{(i)}$ and $\circledast_{i\neq n}^N A^{(i)}$.

Tensor unfolding, or *matricization*, is a process to transform a tensor into a matrix [15]. Generally, given an N^{th}-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, its mode-$n$ unfolding $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times \prod_{i\neq n}^N I_i}$ can be obtained by permuting the dimensions of $\boldsymbol{\mathcal{X}}$ as $[I_n, I_1, \ldots, I_{n-1}, I_{n+1}, \ldots, I_N]$ and then reshaping the permuted tensor into a matrix of size $I_n \times \prod_{i\neq n}^N I_i$.

3.2 CP Decomposition

CP decomposition is a widely used technique for exploring and extracting the underlying structure of the multi-way data. Basically, given an N^{th}-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, CP decomposition approximates this tensor by N *loading* matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$, such that

$$\begin{aligned}\mathbf{X}_{(n)} &\approx \mathbf{A}^{(n)}(\mathbf{A}^{(N)} \odot \cdots \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \cdots \odot \mathbf{A}^{(1)})^\top \\ &= \mathbf{A}^{(n)}(\textstyle\bigodot_{i\neq n}^N \mathbf{A}^{(i)})^\top \\ &= [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]\end{aligned} \tag{1}$$

where $[\![\bullet]\!]$ is defined as the CP decomposition operator and each loading matrices $\mathbf{A}^{(i)}, i \in [1, N]$ is of size $I_i \times R$, where R is the tensor rank, indicating the number of latent factors.

To find the CP decomposition $[\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$ for an N^{th}-order tensor $\mathfrak{X}$, the objective is to minimize the estimation error $\mathcal{L}$, which is defined as

$$\mathcal{L} = \frac{1}{2} \left\| \mathbf{X}_{(n)} - \mathbf{A}^{(n)} (\odot_{i \neq n}^{N} \mathbf{A}^{(i)})^\top \right\|^2$$

However, directly minimizing $\mathcal{L}$ over $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$ is difficult, since $\mathcal{L}$ is not convex w.r.t. $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$. As a result, a widely applied approach is ALS. The main idea is to divide the above optimization problem into N sub-problems and the n-th one, $n \in [1, N]$ fixes all variables but $\mathbf{A}^{(n)}$, and then minimizes the convex objective $\mathcal{L}$ w.r.t. $\mathbf{A}^{(n)}$, that is

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

3.3 Online CP Decomposition

Here we briefly introduce the main idea of existing online CP decomposition algorithms. A third-order online tensor $\mathfrak{X} \in \mathbb{R}^{I \times J \times (t_{old} + t_{new})}$ is used as a running example, where $\mathfrak{X}$ is expanded from $\mathfrak{X}_{old} \in \mathbb{R}^{I \times J \times t_{old}}$ by appending a new chunk of data $\mathfrak{X}_{new} \in \mathbb{R}^{I \times J \times t_{new}}$ at its last mode. Considering that in most online systems, the size of the new incoming data is usually much smaller than that of all existing historical data, thus we assume $t_{new} \ll t_{old}$. The CP decomposition of $\mathfrak{X}_{old}$ is written as $[\![\mathbf{A}_{old}, \mathbf{B}_{old}, \mathbf{C}_{old}]\!]$ and the aim is to find the CP decomposition $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of $\mathfrak{X}$.

SDT and RLST [20]: Both SDT and RLST transform the online tensor decomposition problem into an incremental matrix factorization problem, by letting $\mathbf{D} = \mathbf{B} \odot \mathbf{A}$, so that equation (1) can be written as $\mathbf{X}_{(3)} = \mathbf{C}\mathbf{D}^\top$. Then the problem is how to estimate $\mathbf{C}$ and $\mathbf{D}$.

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

In contrast, RLST follows a more direct approach to get $\mathbf{C}$ and $\mathbf{D}$. Recall that $\mathbf{X}_{(3)} = \mathbf{C}\mathbf{D}^\top$, firstly, $\mathbf{C}_{new}$ is calculated as $\mathbf{X}_{new(3)}(\mathbf{D}_{old}^\top)^\dagger$ and $\mathbf{C}$ is updated by appending $\mathbf{C}_{new}$ to $\mathbf{C}_{old}$. Then $\mathbf{D}$ is incrementally estimated with $\mathbf{X}_{new(3)}$ and $\mathbf{C}_{new}$ based on the matrix inversion and pseudo-inversion lemmas. Due to the page limit, we refer interested readers to the original papers [20] for more details.

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

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

GridTF [22]: As mentioned earlier, the basic idea of GridTF is to partition the whole tensor into smaller tensors and then combine their CP decompositions together. In regards to the example here, $\mathfrak{X}$ is the online tensor, $\mathfrak{X}_{old}$ and $\mathfrak{X}_{new}$ are the two partitions. To obtain the CP decomposition of $\mathfrak{X}$, the first step of GridTF is to decompose $\mathfrak{X}_{new}$ by ALS as $[\![\mathbf{A}_{new}, \mathbf{B}_{new}, \mathbf{C}_{new}]\!]$, while the decomposition of $\mathfrak{X}_{old}$ is already known from the last time step.

The combination step is a recursive update procedure such that in every iteration, each mode is updated with a modified ALS rule, until the whole estimation converges, or the maximum number of iterations has been reached. In our notation, the update rules are given as follows, of which further details can be found in [22].

1) For non-temporal modes $\mathbf{A}$ and $\mathbf{B}$

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

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

2) For temporal mode $\mathbf{C}$

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

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

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

4. OUR APPROACH

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

4.1 Third-order Tensors

Following the same notation introduced in Section 3.3, similar to the classic ALS algorithm, our approach handles the problem in an alternating update fashion. That is, we first fix $\mathbf{A}$ and $\mathbf{B}$, to update $\mathbf{C}$, and then sequentially update $\mathbf{A}$ and $\mathbf{B}$, by fixing the other two.

4.1.1 Update Temporal Mode C

By fixing $\mathbf{A}$ and $\mathbf{B}$, from (2) we have

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

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

$$\mathbf{C} = \begin{bmatrix}\mathbf{C}_{old}\\ \mathbf{C}_{new}\end{bmatrix} = \begin{bmatrix}\mathbf{C}_{old}\\ \mathbf{X}_{new(3)}((\mathbf{B}\odot\mathbf{A})^\top)^\dagger\end{bmatrix} \tag{4}$$

4.1.2 Update Non-temporal Modes A and B

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

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

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

$$\mathbf{A} = \mathbf{P}\mathbf{Q}^{-1} \tag{5}$$

Directly calculating $\mathbf{P}$ and $\mathbf{Q}$ is costly. This is mainly because the output of $(\mathbf{C}\odot\mathbf{B})$ is a huge matrix of size $J(t_{old} + t_{new}) \times R$, where R is the tensor rank. It further results in $\mathcal{O}(RIJ(t_{old}+t_{new}))$ and $\mathcal{O}(R^2J(t_{old}+t_{new}))$ operations to get $\mathbf{P}$ and $\mathbf{Q}$, respectively. Although for $\mathbf{Q}$, the Khatri-Rao product can be avoided by calculating it as $(\mathbf{C}^\top\mathbf{C})\circledast(\mathbf{B}^\top\mathbf{B})$ [16], which has a complexity of $\mathcal{O}(R^2(J + t_{old} + t_{new}))$, this is still expensive since t_{old} is usually quite large. As a result, in order to improve the efficiency, we need a faster approach.

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

$$\begin{aligned}\mathbf{P} &= \mathbf{X}_{(1)}(\mathbf{C}\odot\mathbf{B}) \\ &= \left[\mathbf{X}_{old(1)}, \mathbf{X}_{new(1)}\right]\left(\begin{bmatrix}\mathbf{C}_{old}\\ \mathbf{C}_{new}\end{bmatrix}\odot\mathbf{B}\right) \\ &= \left[\mathbf{X}_{old(1)}, \mathbf{X}_{new(1)}\right]\begin{bmatrix}\mathbf{C}_{old}\odot\mathbf{B}\\ \mathbf{C}_{new}\odot\mathbf{B}\end{bmatrix} \\ &= \mathbf{X}_{old(1)}(\mathbf{C}_{old}\odot\mathbf{B}) + \mathbf{X}_{new(1)}(\mathbf{C}_{new}\odot\mathbf{B})\end{aligned} \tag{6}$$

recall that $\mathbf{B}$ has been fixed as $\mathbf{B}_{old}$, so that the first part of the last line of equation (6) only contains components from the previous time step. Suppose we know this part already and denote it by $\mathbf{P}_{old}$, then (6) can be rewritten as

$$\mathbf{P} = \mathbf{P}_{old} + \mathbf{X}_{new(1)}(\mathbf{C}_{new}\odot\mathbf{B}) \tag{7}$$

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

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

$$\begin{aligned}\mathbf{Q} &= \mathbf{Q}_{old} + (\mathbf{C}_{new}\odot\mathbf{B})^\top(\mathbf{C}_{new}\odot\mathbf{B}) \\ &= \mathbf{Q}_{old} + (\mathbf{C}_{new}^\top\mathbf{C}_{new})\circledast(\mathbf{B}^\top\mathbf{B})\end{aligned} \tag{8}$$

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

$$\begin{aligned}&\mathbf{P} \leftarrow \mathbf{P} + \mathbf{X}_{new(1)}(\mathbf{C}_{new}\odot\mathbf{B})\\ &\mathbf{Q} \leftarrow \mathbf{Q} + (\mathbf{C}_{new}^\top\mathbf{C}_{new})\circledast(\mathbf{B}^\top\mathbf{B})\\ &\mathbf{A} \leftarrow \mathbf{P}\mathbf{Q}^{-1}\end{aligned} \tag{9}$$

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

$$\begin{aligned}&\mathbf{U} \leftarrow \mathbf{U} + \mathbf{X}_{new(2)}(\mathbf{C}_{new}\odot\mathbf{A})\\ &\mathbf{V} \leftarrow \mathbf{V} + (\mathbf{C}_{new}^\top\mathbf{C}_{new})\circledast(\mathbf{A}^\top\mathbf{A})\\ &\mathbf{B} \leftarrow \mathbf{U}\mathbf{V}^{-1}\end{aligned} \tag{10}$$

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

To sum up: For a third-order tensor that grows with time, we propose an efficient algorithm for tracking its CP decomposition on the fly. We name this algorithm as *OnlineCP*, comprising the following two stages:

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

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

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

a) for the temporal mode 3, $\mathbf{C}$ is updated with (4)

b) for non-temporal modes 1 and 2, $\mathbf{A}$ is updated with (9) and $\mathbf{B}$ is updated with (10), respectively.

4.2 Extending to Higher-Order Tensors

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

4.2.1 Update Temporal Mode

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

$$\mathbf{A}^{(N)} \leftarrow \arg\min_{\mathbf{A}^{(N)}} \frac{1}{2}\left\|\mathbf{X}_{(N)} - \mathbf{A}^{(N)}(\bigodot^{N-1}\mathbf{A}^{(i)})^\top\right\|^2$$

Basically, the above equation has the same structure as (3), so we have a similar update rule for $\mathbf{A}^{(N)}$

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

4.2.2 Update Non-temporal Modes

For each non-temporal mode $n \in [1, N-1]$, the estimation error $\mathcal{L}$ on mode n is $\frac{1}{2}\left\|\mathbf{X}_{(n)} - \mathbf{A}^{(n)}(\bigodot_{i\neq n}^{N}\mathbf{A}^{(i)})^{\top}\right\|^2$, and similar update rule as equation (9) can be applied, that is

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

where we denote the Khatri-Rao product of the first $N-1$ but the n-th loading matrices, $\bigodot_{i\neq n}^{N-1}\mathbf{A}^{(i)}$, as $\mathbf{K}^{(n)}$ and the Hadamard product, $\circledast_{i\neq n}^{N-1}{\mathbf{A}^{(i)}}^{\top}\mathbf{A}^{(i)}$, as $\mathbf{H}^{(n)}$.

4.2.3 Avoid Duplicated Computation

In fact, if we were to compute every $\mathbf{K}^{(n)}$ for each $n \in [1, N-1]$, there would be some redundant computation among them. Take a 5th-order tensor for example, where $\mathbf{K}^{(1)} = \mathbf{A}^{(4)} \odot \mathbf{A}^{(3)} \odot \mathbf{A}^{(2)}$, $\mathbf{K}^{(2)} = \mathbf{A}^{(4)} \odot \mathbf{A}^{(3)} \odot \mathbf{A}^{(1)}$, $\mathbf{K}^{(3)} = \mathbf{A}^{(4)}\odot\mathbf{A}^{(2)}\odot\mathbf{A}^{(1)}$, and $\mathbf{K}^{(4)} = \mathbf{A}^{(3)}\odot\mathbf{A}^{(2)}\odot\mathbf{A}^{(1)}$. It is clear that both $\mathbf{K}^{(1)}$ and $\mathbf{K}^{(2)}$ have computed $\mathbf{A}^{(4)}\odot\mathbf{A}^{(3)}$, and $\mathbf{K}^{(3)}$ and $\mathbf{K}^{(4)}$ share a common computation $\mathbf{A}^{(2)}\odot\mathbf{A}^{(1)}$. These redundant Khatri-Rao products are computationally expensive to calculate, and more importantly, the amount of redundancy will dramatically increase with the number of modes N, since more common components are shared.

To overcome this issue, we use a dynamic programming strategy to compute all the $\mathbf{K}^{(n)}$'s in one run, by making good use of the intermediate results and avoiding duplicated operations. This process is detailed in Algorithm 1 and an illustrating example of a 6th-order tensor is given in Figure 1. The main idea is to go through the loading matrix list $\mathbf{A}^{(N-1)}, \dots, \mathbf{A}^{(2)}, \mathbf{A}^{(1)}$ from both ends, until the algorithm reaches the results of $\mathbf{K}^{(1)}$ and $\mathbf{K}^{(N-1)}$ (lines 3 to 8). After that, for the rest of $\mathbf{K}^{(i)}$ where $i \in [2, N-2]$, they are computed as the Khatri-Rao products of the intermediate results from the last loop (lines 11 to 15).

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

Finally, by putting everything together, we obtain the general version of our OnlineCP algorithm[1], as presented in Algorithm 2 and 3.

4.3 Complexity Analysis

Following the same notation as Section 4.2, let R be the tensor rank, $S = \prod_{i=1}^{N-1} I_i$, and $J = \sum_{i=1}^{N-1} I_i$. To process a new chunk of data $\mathfrak{X}_{new}$, it takes up to $(N-1)S$ operations to get all the $\mathbf{K}^{(n)}, n \in [1, N-1]$, and $\mathbf{H}$ can be obtained in $R^2J + (N-2)R^2$ operations (lines 1 and 2 in Algorithm 3). To update the time mode, RS, St_{new}, and $RSt_{new} + R^2t_{new} + R^3$ operations are required to get $\mathbf{K}^{(N)}$, $\mathbf{X}_{new(N)}$, and $\mathbf{A}^{(N)}_{new}$, respectively (lines 3, 4). Note that the pseudoinverse $((\mathbf{K}^{(N)})^{\top})^{\dagger}$ can be replaced by $\mathbf{K}^{(N)}\mathbf{H}^{\dagger}$ as $(\mathbf{A} \odot \mathbf{B})^{\dagger} = ((\mathbf{A}^{\top}\mathbf{A}) \circledast (\mathbf{B}^{\top}\mathbf{B}))^{\dagger}(\mathbf{A} \odot \mathbf{B})^{\top}$ [16]. For each non-temporal mode n (lines 7 to 10), St_{new}, RSt_{new}/I_n ($\approx St_{new}$, since R is usually smaller than I_n), RSt_{new} and RI_n operations are required for the unfolding, Khatri-Rao, multiplication and addition in the step to update $\mathbf{P}^{(n)}$; and $\mathbf{Q}^{(n)}$ takes $R^2I_n + R^2t_{new} + 3R^2$ operations to update; then the updated $\mathbf{A}^{(n)}$ can be calculated in $R^3 + R^2I_n$ operations. Thus, to update the loading matrix $\mathbf{A}^{(n)}$ of mode n, $(2R^2+R)I_n + (R+2)St_{new} + R^3 + (3+t_{new})R^2$ operations is required and the whole update procedure for non-temporal modes takes $(2R^2 + R)J + (N-1)(R+2)St_{new} + (N-1)(R^3 + (3 + t_{new})R^2)$ operations. Overall, as S is usually much larger than other factors, the time complexity of OnlineCP can be written as $\mathcal{O}(NRSt_{new})$, which is constant w.r.t. the length of processed data t_{old}.

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

[1]We provide our Matlab implementation of `OnlineCP` at `http://shuo-zhou.info`.

Algorithm 1: Get a list of Khatri-Rao products

```
   Input: A list of loading matrices $[\mathbf{A}^{(N-1)}, \dots, \mathbf{A}^{(1)}]$
   Output: A list of Khatri-Rao products
           $[\mathbf{K}^{(1)}, \dots, \mathbf{K}^{(N-1)}]$
1  $left \leftarrow [\mathbf{A}^{(N-1)}]$
2  $right \leftarrow [\mathbf{A}^{(1)}]$
3  if $N > 3$ then
4      for $n \leftarrow 2$ to $N-2$ do
5          $left[n] \leftarrow left[n-1] \odot \mathbf{A}^{(N-n)}$
6          $right[n] \leftarrow \mathbf{A}^{(n)} \odot right[n-1]$
7      end
8  end
9  $\mathbf{K}^{(1)} \leftarrow left[N-2]$
10 $\mathbf{K}^{(N-1)} \leftarrow right[N-2]$
11 if $N > 3$ then
12     for $n \leftarrow 2$ to $N-2$ do
13         $\mathbf{K}^{(n)} \leftarrow left[N-n-1] \odot right[n-1]$
14     end
15 end
```

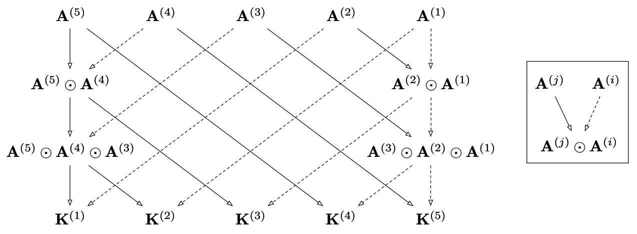


Figure 1: A 6th-order example to get all $\mathbf{K}^{(n)}$'s together. A Khatri-Rao product is represented by two arrows, of which the solid one linked to the first input. The two lists, *left* and *right*, are indicated by the two columns on the graph.

Table 1: Complexity comparison between OnlineCP and existing methods.

	Time	Space	Source
OnlineCP	$\mathcal{O}(NRSt_{new})$	$St_{new} + (2J + t_{old})R + (N-1)R^2$	
ALS	$\mathcal{O}(NRS(t_{old} + t_{new}))$	$S(t_{old} + t_{new})$	[4]
SDT	$\mathcal{O}(R^2(t_{old} + S))$	$St_{new} + (J + S + 2t_{old})R + 3R^2$	[20]
RLST	$\mathcal{O}(R^2 S)$	$St_{new} + (J + t_{old} + 2S)R + 2R^2$	[20]
GridTF	$\mathcal{O}(NRSt_{new} + R^2(J + t_{old} + t_{new}))$	$St_{new} + (J + t_{old})R$	[22]

Algorithm 2: Initialization stage of OnlineCP

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

Algorithm 3: Update stage of OnlineCP

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

We summarize the complexity of our approach in Table 1, along with other existing approaches. Note that the complexities of SDT and RLST are based on the exponential window [20], which considers all existing data while leverages their importance by a forgetting factor λ. In addition, as they only work on third-order tensors, when other methods are compared to them, N should be set to 3. Another remark is that the time complexities of ALS and GridTF are based on one iteration only, in reality they would take a few iterations until convergence.

5. EMPIRICAL ANALYSIS

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

5.1 Real World Datasets

5.1.1 Experimental Specifications

Datasets: The experiments are conducted on seven real world datasets of varying characteristics, all are naturally of multi-way structures. These are two image datasets: (i) Columbia Object Image Library (COIL); (ii) ORL Database of Faces (FACE); three human activity datasets: (iii) Daily and Sports Activities Data Set (DSA); (iv) University of Southern California Human Activity Dataset (HAD); (v) Daphnet Freezing of Gait Data Set (FOG); one chemical laboratory dataset: (vi) Gas sensor array under dynamic gas mixtures Data Set (GAS); and a (vii) road traffic dataset collected from loop detectors in Victoria, Australia (ROAD).

Each dataset is represented by a tensor with its most natural structure. For instance, FACE is represented by a $pixel \times pixel \times shot$ third-order tensor, while DSA is stored as an 4th-order tensor of $subject \times trail \times sensor \times time$. Furthermore, since some of our baselines can only work with third-order tensors, for tensors with higher-order, DSA, GAS, and HAD, we randomly extract third-order sub-tensors from them. Conversely, to enlarge the number of higher-order tensors, image datasets, COIL and FACE have been transformed into 4th-order tensors by treating each image as a collection of small *patches*, which forms the extra order. As a result, there are five datasets having two versions of representation: a third-order one, indicated by suffix *3D*, and a higher-order form with suffix *HD*. The details of these datasets can be found in Table 2.

Baselines: In this experiment, five baselines have been selected as the competitors to evaluate the performance.

(i) Batch Cold: an implementation of ALS algorithm in Tensor Toolbox [4] without special initialization.

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

(iii) SDT [20]: an adaptive algorithm based on incrementally tracking the SVD of the unfolded tensor.

(iv) RLST: another online approach proposed in [20]. Instead of tracking the SVD, recursive updates are performed to minimize the mean squared error on new data.

Table 2: Details of datasets

Datasets	Size	Slice Size $S = \prod_{i=1}^{N-1} I_i$	Source
COIL-3D	$128 \times 128 \times 240$	16,384	[19]
COIL-HD	$64 \times 64 \times 25 \times 240$	102,400	
DSA-3D	$8 \times 45 \times 750$	360	[2]
DSA-HD	$19 \times 8 \times 45 \times 750$	6,840	
FACE-3D	$112 \times 92 \times 400$	10,304	[23]
FACE-HD	$28 \times 23 \times 16 \times 400$	10,304	
FOG	$10 \times 9 \times 1000$	90	[3]
GAS-3D	$30 \times 8 \times 2970$	240	[12]
GAS-HD	$30 \times 6 \times 8 \times 2970$	1,440	
HAD-3D	$14 \times 6 \times 500$	64	[30]
HAD-HD	$14 \times 12 \times 5 \times 6 \times 500$	3,840	
ROAD	$4666 \times 96 \times 1826$	447,936	[24]

(v) GridTF [22]: an divide-and-conqure based algorithm. To find CP decompositions for online tensors, the partitioning is enforced on the time mode only.

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

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

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

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

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

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

In terms of method-specific parameters, for the two batch algorithms, the default settings, $\varepsilon = 1e - 4$ and $maxiters = 50$ are used. For GridTF, which contains an ALS procedure for the *new* data slice and a recursively update procedure for estimating the *whole* current tensor, the same default parameters are chosen for the ALS step; while $\varepsilon = 1e - 2$ and $maxiters = 50$ are used for the update phase. Additionally, since batch algorithms do not provide a weighting strategy to differentiate the importance of data, in order to make a fair comparison, all the data slices are equally treated and there is no difference between older and newer ones in terms of their weights, which means the exponential window is used with $\lambda = 1$ in SDT and RLST.

5.1.2 Results

Given a particular dataset and a specific algorithm, its fitness and processing time are two time series (averaged over 10 runs). We take the mean values of them and report the results for third-order tensors in Table 3 and the higher-order tensors in Table 4. In addition, for the four online approaches, SDT, RLST, GridTF and OnlineCP, their relative performances compared to Batch Hot are also shown in the parenthesis. Finally, the best results among these four are indicated by boldface.

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

As the earliest studies of online CP decomposition, both SDT and RLST address this efficiency issue very well. Compared with Batch Hot, they shorten the mean running time by up to 400 times. RLST, in particular, was the most efficient online algorithm on 4 out of 7 third-order tensor datasets. In fact, the efficiency of SDT is quite close to RLST, except for the GAS dataset, whose length of time mode is significantly higher than other datasets. This shows that SDT is more sensitive to the growth of time. However, the main issue of SDT and RLST is their estimation accuracy. For some datasets, such as COIL and HAD, they work fine, while for some others like DSA, they exhibit fairly poor accuracy, achieving only nearly half of the fitness of batch methods. The same accuracy problem can be observed in GridTF as well. In terms of efficiency, there is no significant difference between GridTF and Batch Hot, at least on the small size datasets. In fact, we notice that a substantial amount of time of GridTF is consumed by decomposing the new data slice and this cost is particularly dominant when the tensor size is not large enough. This can be confirmed by observing its generally better efficiency on higher-order datasets, compared to the third-order ones.

Table 3: Experimental results of third-order datasets

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

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

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

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

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

Table 4: Experimental results of higher-order datasets.

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

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

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

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

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

5.2 Sensitivity to Initialization

Throughout our experiments, an interesting observation was made: for all adaptive algorithms that make use of the previous step results, namely Batch Hot, SDT, RLST, GridTF, and OnlineCP, their best results are usually linked to a good initial fitness, while poor-quality initializations often lead them to subsequent under-fitting. To explore the impact of initialization to each algorithm, the following ex-

Table 5: The final fitness averaged over 200 runs with different initial fitness. Results are displayed as $mean \pm std$, where $mean$ is the average final fitness and std is the standard deviation, both in % (the higher the values the better).

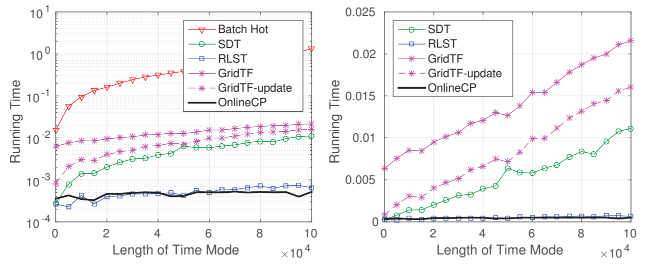
	Final Fitness
Batch Hot	82.57±10.1474
SDT	7.68±55.5205
RLST	33.15±36.9270
GridTF	57.43±15.2148
OnlineCP	67.67±12.9846

periment has been conducted. We generate a synthetic tensor $\mathcal{X} \in \mathbb{R}^{20\times 20\times 100}$ by constructing from random loading matrices and then downgrade it by a Gaussian noise with a Signal-to-Interference Rate (SIR) of 20 dB. The best fitness to $\mathcal{X}$ in 10 runs of ALS is 90.14%. This tensor is then repeatedly decomposed by the above five methods for 200 times. At the beginning of each run, half of the data is used for initialization by ALS with a random tolerance from $9e-1$ to $1e-4$, to produce different level of initial fitness. Then the rest of data is sequentially added and processed by each online algorithm. The averaged final fitness over all runs is used as the effectiveness indicator, as well as the standard deviation. Table 5 shows the experimental results with average initial fitness as 65.78% and standard deviation as 15.3704%.

As shown in Table 5, overall, the low quality initialization has a negative impact on all algorithms. Even the most powerful one, Batch Hot, cannot always reach the best fitness and shows a decline of 10%. For SDT and RLST, it turns out that both algorithms are significantly dependent on the initial fitness. When the initial fitness is lower than the best value, their performance can quickly drop to an unacceptable level. In addition, their results are also highly unstable, as demonstrated by a large variance. While both GridTF and OnlineCP exhibit much more stable performance, the final fitness of GridTF is considerably lower than our approach OnlineCP. This experimental evidence demonstrates that the proposed algorithm, OnlineCP, is less sensitive to the quality of the initialization, compared with exiting online methods. However, it can be seen that good initialization still plays an important role to our algorithm. Thus, for applying our method, we suggest to validate the goodness of the initialization at the beginning, in order to obtain the best subsequent effectiveness.

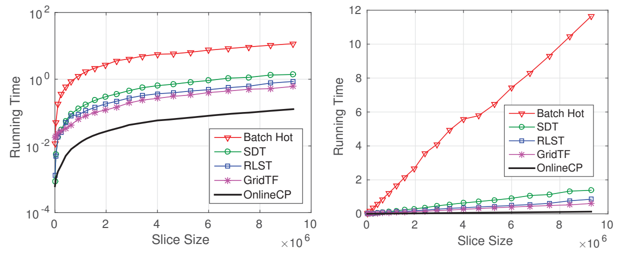
5.3 Scalability Evaluation

According to Section 4.3, the time complexity of each algorithm is mainly determined by the slice size and the length of processed data. To confirm our analysis and evaluate the scalability of our algorithm, firstly, a tensor $\mathcal{X} \in \mathbb{R}^{20\times 20\times 10^5}$ of small slice size but long time dimension is decomposed. After initializing with data of the first 100 timestamps, each method's running time for processing one data slice at each time step is measured and displayed in Figure 2. In addition, to examine the impact of slice size to efficiency, we fix the time mode to 100, and generate a group of tensors of different slice sizes, ranging from 100 to 9×10^6. For each tensor, its first 20% of data is used for initialization and the average running time for processing the rest data slices is



(a) (b)

Figure 2: Running time (in seconds) for adding one slice to a $20 \times 20 \times (t-1)$ tensor at time t. Two figure represents the same information, differing only in the y-axis scale.



(a) (b)

Figure 3: Running time (in seconds) for processing different size of tensors. Two figure represents the same information, differing only in the y-axis scale.

shown in Figure 3. For better comparison, both Batch Hot and GridTF are forced to execute 1 iteration only in these two experiments. Note that the y-axis of Figures 2a and 3a is displayed in log scale and in Figure 2b, Batch Hot has been removed for better visibility.

As can be seen from Figure 2, both RLST and OnlineCP show constant complexities and the increasing length of processed data has no impact on them. For the other approaches, a clear linear growth with time can be observed in Batch Hot and SDT, which makes them less feasible for online learning purposes. The change of time consumption in GridTF is less obvious compared with Batch Hot and SDT. However, after removing the time used by its inner ALS procedure, similar linear trend can be seen, marked as GridTF-update in the figure.

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

6. CONCLUSIONS AND FUTURE WORK

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

There is still room for improving our method. One direction is to further extend it for more general dynamic tensors that may be changed on any modes [11]. Another potential direction is to incorporate constraints, such as non-negativity [13], so that our method can be more suitable for applications such as computer vision.

Acknowledgments

This work is supported by the Australian Research Council via grant number DP140101969.

7. REFERENCES

[1] E. Acar, et al. Scalable tensor factorizations for incomplete data. *Chemometrics and Intelligent Laboratory Systems*, 106(1):41–56, March 2011.

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

[3] M. Bächlin, et al. Wearable assistant for parkinson's disease patients with the freezing of gait symptom. *IEEE Trans. Inf. Tech. Biomed.*, 14(2):436–446, 2010.

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

[5] Y. Cai, et al. Facets: Fast comprehensive mining of coevolving high-order time series. In *SIGKDD*, 2015.

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

[7] A. Cichocki, et al. Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *Signal Processing Magazine, IEEE*, 32(2):145–163, 2015.

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

[9] L. De Lathauwer, et al. On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors. *SJMAEL*, 21(4):1324–1342, 2000.

[10] D. M. Dunlavy, T. G. Kolda, and E. Acar. Temporal link prediction using matrix and tensor factorizations. *TKDD*, 5(2):10, 2011.

[11] H. Fanaee-T and J. Gama. Multi-aspect-streaming tensor analysis. *Knowledge-Based Systems*, 89:332–345, 2015.

[12] J. Fonollosa, et al. Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring. *Sens. Actuator B-Chem.*, 215:618–629, 2015.

[13] C. Hu, et al. Scalable bayesian non-negative tensor factorization for massive count data. In *ECML-PKDD*, 2015.

[14] W. Hu, et al. Incremental tensor subspace learning and its applications to foreground segmentation and tracking. *IJCV*, 91(3):303–327, 2011.

[15] T. G. Kolda. Multilinear operators for higher-order decompositions. Tech. Report SAND2006-2081, Sandia National Laboratories, April 2006.

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

[17] W. Liu, et al. Utilizing common substructures to speedup tensor factorization for mining dynamic graphs. In *CIKM*, 2012.

[18] X. Ma, et al. Dynamic updating and downdating matrix svd and tensor hosvd for adaptive indexing and retrieval of motion trajectories. In *ICASSP*, 2009.

[19] S. A. Nene, et al. Columbia Object Image Library (COIL-20). Tech. report, Feb 1996.

[20] D. Nion, et al. Adaptive algorithms to track the parafac decomposition of a third-order tensor. *IEEE Trans. Sig. Process.*, 57(6):2299–2310, 2009.

[21] S. Papadimitriou, et al. Streaming pattern discovery in multiple time-series. In *VLDB*, 2005.

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

[23] F. S. Samaria, et al. Parameterisation of a stochastic model for human face identification. In *WACV*, 1994.

[24] F. Schimbinschi, et al. Traffic forecasting in complex urban networks: Leveraging big data and machine learning. In *Big Data*, 2015.

[25] L. Shi, et al. Stensr: Spatio-temporal tensor streams for anomaly detection and pattern discovery. *KAIS*, 43(2):333–353, 2015.

[26] A. Sobral, et al. Incremental and multi-feature tensor subspace learning applied for background modeling and subtraction. In *ICIAR*, 2014.

[27] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *SIGKDD*, 2006.

[28] J. Sun, et al. Incremental tensor analysis: Theory and applications. *TKDD*, 2(3):11, 2008.

[29] M. A. O. Vasilescu, et al. Multilinear analysis of image ensembles: Tensorfaces. In *ECCV*, 2002.

[30] M. Zhang and A. A. Sawchuk. Usc-had: A daily activity dataset for ubiquitous activity recognition using wearable sensors. In *Ubicomp-SAGAware*, 2012.