SamBaTen: Sampling-based Batch Incremental Tensor Decomposition

Ekta Gujral
UC Riverside
egujr001@ucr.edu

Ravdeep Pasricha
UC Riverside
rpasr001@ucr.edu

Evangelos E. Papalexakis
UC Riverside
epapalex@cs.ucr.edu

Abstract

Tensor decompositions are invaluable tools in analyzing multimodal datasets. In many real-world scenarios, such datasets are far from being static, to the contrary they tend to grow over time. For instance, in an online social network setting, as we observe new interactions over time, our dataset gets updated in its "time" mode. How can we maintain a valid and accurate tensor decomposition of such a dynamically evolving multimodal dataset, without having to re-compute the entire decomposition after every single update? In this paper we introduce SamBaTen, a *Sam*pling-based *Ba*tch Incremental *Ten*sor Decomposition algorithm, which incrementally maintains the decomposition given new updates to the tensor dataset. SamBaTen is able to scale to datasets that the state-of-the-art in incremental tensor decomposition is unable to operate on, due to its ability to effectively summarize the existing tensor and the incoming updates, and perform all computations in the reduced summary space. We extensively evaluate SamBaTen using synthetic and real datasets. Indicatively, SamBaTen achieves comparable accuracy to state-of-the-art incremental and non-incremental techniques, while being up to *25-30 times faster*. Furthermore, SamBaTen scales to very large sparse and dense dynamically evolving tensors of dimensions up to $100K \times 100K \times 100K$ where state-of-the-art incremental approaches were not able to operate.

1 Introduction

Tensor decomposition is a very powerful tool for many problems in data mining [11, 16]. The success of tensor decomposition lies in its capability of finding complex patterns in multi-way settings, by leveraging higher-order structure and correlations within the data. The dominant tensor decompositions are CP/PARAFAC (henceforth referred to as CP), which extracts interpretable latent factors from the data, and Tucker, which estimates the joint subspaces of the tensor. In this work we focus on the CP decomposition, which has been shown to be extremely effective in exploratory data mining time and time again [16].

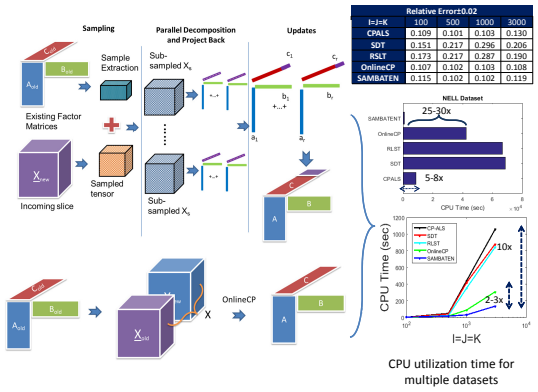

Figure 1: SamBaTen outperforms state-of-the-art baselines while maintaining competitive accuracy.

In a wide array of modern real-world applications, data are far from being static. To the contrary, data get updated dynamically. For instance, in an online social network, new interactions occur every second and new friendships are formed at a similar pace. In the tensor realm, we may view a large proportion of these dynamic updates as an introduction of new "slices" in the tensor: in the social network example, new interactions that happen as time evolves imply the introduction of new snapshots of the network, which grow the tensor in the "time" mode. A tensor decomposition in that tensor can discover *communities* and their evolution over time. How can we handle such updates in the data without having to re-compute the decomposition whenever an update arrives, but incrementally update the existing results given the new data? In the community detection example, how can we track the evolution of the existing communities, and discover new ones, for the new updates that continuously arrive?

Computing the decomposition for a dynamically updated tensor is challenging, with the challenges lying, primarily, on two of the three V's in the traditional definition of Big Data: *Volume* and *Velocity*. As a tensor dataset is updated dynamically, its volume increases to the point that techniques which are not equipped to handle those updates *incrementally*, inevitably fail to execute due to the sheer size of the data. Furthermore,

even though the applications that tensors have been successful so far do not require real-time execution per se, the decomposition algorithm must, nevertheless, be able to ingest the updates to the data at a rate that will not result in the computation being "drowned" by the incoming updates.

The majority of prior work has focused on the Tucker Decomposition of incrementing tensors [15, 19, 7], however very limited amount of work has been done on the CP. Nion and Sidiropoulos [14] proposed two methods namely Simultaneous Diagonalization Tracking (SDT) and Recursive Least Squares Tracking (RLST) and most recently, [21] introduced the OnlineCP decomposition for higher order online tensors. Even though prior work in incremental CP decomposition, by virtue of allowing for incremental updates to the already computed model, is able to deal with *Velocity*, when compared to the naive approach of re-executing the entire decomposition on the updated data, every time a new update arrives, it falls short when the *Volume* of the data grows.

In this paper we propose a novel large scale incremental CP tensor decomposition that leverages (potential) sparsity of the data, and achieves faster and more scalable performance than state-of-the-art baselines, while maintaining comparable accuracy.

We show a snapshot of our results in Figure 1: SamBaTen is faster than all state-of-the-art methods on data that the baselines were able to operate on. Furthermore, SamBaTen was able to scale to, both dense and sparse, dynamically updated tensors, where none of the baselines was able to run. Finally, SamBaTen achieves comparable accuracy to existing incremental and non-incremental methods. Our contributions are summarized as follows:

- **Novel scalable algorithm**: The advantage of SamBaTen stems from the fact that it only operates on small summaries of the data at all times, thereby being able to maintain its efficiency regardless of the size of the full data. To the best of our knowledge, this is the first incremental tensor decomposition which effectively leverages sparsity in the data.
- **Extensive experimental evaluation**: Through experimental evaluation on six real-world datasets with sizes that range up to 70GB, and synthetic tensors that range up to $100K \times 100K \times 100K$, we show thatSamBaTen can incrementally maintain very accurate decompositions, faster and in a more efficient and scalable manner than state-of-the-art methods.

Reproducibility: We make our Matlab implementation publicly available at link [1]. Furthermore, all the datasets we use for evaluation are publicly available.

2 Problem Formulation

2.1 Preliminary Definitions Tensor : A tensor is a higher order generalization of a matrix. In order to avoid overloading the term "dimension", we call an $I \times J \times K$ tensor a three "mode" tensor, where "modes" are the numbers of indices used to index the tensor. The number of modes is also called "order". Table 1 contains the symbols used throughout the paper. We refer the interested reader to several surveys that provide more details and a wide variety of tensor applications [10, 16]. In the interest of space, we also refer the reader to [16] for the definitions of Kronecker and Khatri-Rao products which are not essential for following the basic derivation of our approach.

Slice : A slice is a (m-1)-dimension partition of tensor where an index is varied in one mode and the indices fixed in the other modes. There are three categories of slices : horizontal ($\underline{\mathbf{X}}$(i,:,:)) , lateral ($\underline{\mathbf{X}}$(:,j,:)), and frontal ($\underline{\mathbf{X}}$(:,:,k)) for third-order tensor X as shown in Figure 2.

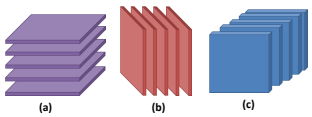

Figure 2: Slices of 3-order tensor (a) horizontal $\underline{\mathbf{X}}(i,:,:)$ (b) lateral $\underline{\mathbf{X}}(:,j,:)$, (c) frontal $\underline{\mathbf{X}}(:,j,:)$.

Symbols	Definition
$\underline{\mathbf{X}}, \mathbf{X}, \mathbf{x}, x$	Tensor, Matrix, Column vector, Scalar
$\mathbb{R}$	Set of Real Numbers
$\circ$	Outer product
$\|\mathbf{A}\|_F, \|\mathbf{a}\|_2$	Frobenius norm, ℓ_2 norm
$\mathbf{x}(I)$	Spanning the elements of $\mathbf{x}$ in indices $\in I$
$\mathbf{x}(:)$	Spanning all elements of $\mathbf{x}$
$\mathbf{X}(:,r)$	r^{th} column of $\mathbf{X}$
$\mathbf{X}(r,:)$	r^{th} row of $\mathbf{X}$
$\otimes$	Kronecker product
$\odot$	Khatri-Rao product (column-wise Kronecker product [16])

Table 1: Table of symbols and their description

Canonical Polyadic Decomposition : One of the most popular and widely used tensor decompositions is the Canonical Polyadic (CP) or CANDECOMP/PARAFAC decomposition [3, 9]. We henceforth refer to this decomposition as CP. In CP., the tensor is decomposed into a sum of rank-one tensors, i.e., a sum of outer products of three vectors (for three-mode tensors): $\underline{\mathbf{X}} \approx \sum_{r=1}^{R} \mathbf{A}(:,r) \circ \mathbf{B}(:,r) \circ \mathbf{C}(:,r)$ where $\mathbf{A} \in \mathbb{R}^{I \times R}, \mathbf{B} \in \mathbb{R}^{J \times R}, \mathbf{C} \in \mathbb{R}^{K \times R}$, and the outer product is given by $(\mathbf{A}(:,r) \circ \mathbf{B}(:,r) \circ \mathbf{C}(:,r))(i,j,k) =$

[1] http://www.cs.ucr.edu/~egujr001/ucr/madlab/src/SAMBATEN.zip

$\mathbf{A}(i,r)\mathbf{B}(j,r)\mathbf{C}(k,r)$ for all i,j,k. In order to compute the decomposition we typically need to minimize the squared differences (i.e., Frobenius norm) between the original tensor and the model There exist other modeling approaches in the literature [4] which minimize the KL-Divergence, however, Frobenius norm-based approaches are still to this day the most well studied. We reserve investigation of other loss functions as future work.

2.2 Problem Definition In many real-world applications, data grow dynamically . In a time-evolving social network, we observe user interactions every few seconds, which can be translated to new tensor slices, after fixing the temporal granularity (a problem which, on its own merit, is very hard to solve optimally, and we do not address in this paper). This incremental property of data gives rise to the need for an on-the-fly update of the existing decomposition, which we name incremental tensor decomposition. Notice that the literature (and thereby this paper) uses the terms "incremental", "dynamic", and "online" interchangeably. In such scenarios, data updates happen very fast which make traditional (non-incremental) methods to collapse because they need to recompute the decomposition for the entire dataset. We focus on a 3-mode tensor one of whose dimensions are growing with time. However, the problem definition (and our proposed method) extends to any number of modes. Let us consider $\underline{\mathbf{X}}(t)=\mathbb{R}^{I\times J\times K_1(t)}$ at time t. The CP decomposition of $\underline{\mathbf{X}}(t)$ is given as :

$$\underline{\mathbf{X}}^{(1)}(t) \approx (\mathbf{A}(t) \odot \mathbf{B}(t))\mathbf{C}^T(t) \approx \mathbf{L}(t)\mathbf{C}^T(t)$$

where $\mathbf{L}(t) = (\mathbf{A}(t) \odot \mathbf{B}(t))$ of dimension $IJ \times R$ and $\mathbf{C}^T(t)$ is of dimension $K_1 \times R$. When new incoming slice $\underline{\mathbf{X}}(t^{'})=\mathbb{R}^{I\times J\times K_2(t^{'})}$ is added in mode 3, required decomposition at time $t^{'}$ is :

$$\underline{\mathbf{X}}^{(1)}(t+t^{'}) \approx \mathbf{L}(t+t^{'})\mathbf{C}^T(t+t^{'})$$

where $\mathbf{L}(t+t^{'}) = (\mathbf{A}(t+t^{'}) \odot \mathbf{B}(t+t^{'}))$ of dimension $IJ\times R$ and $\mathbf{C}^T(t+t^{'})$ is of dimension $(K_1+K_2)\times R$.

The problem that we solve is the following:

Problem Definition. Given (a) an existing set of decomposition results $\mathbf{A}(t), \mathbf{B}(t)$ and $\mathbf{C}(t)$ of R components, which approximate tensor $\underline{\mathbf{X}}_{old}$ of size $I \times J \times K_1$ at time t , (b) new incoming batch of slices in form of tensor $\underline{\mathbf{X}}_{new}$ of size $I \times J \times K_2$ at any time $t^{'}$, find updates of $\mathbf{A}(t^{'}), \mathbf{B}(t^{'})$ and $\mathbf{C}(t^{'})$ **incrementally** to approximate tensor $\underline{\mathbf{X}}$ of dimension $I \times J \times K$, where K $=K_1 + K_2$ after appending new slice or tensor to 3^{rd} mode while maintaining a comparable accuracy with running the full CP decompositon on the entire updated tensor $\underline{\mathbf{X}}$.

To simplify notation, we will interchangeably refer to $\mathbf{A}(t)$ as $\mathbf{A}_{old}$ (when we need to refer to specific indices of that matrix), and similarly for $\mathbf{A}(t')$ we shall refer to it as $\mathbf{A}^{'}$.

3 Proposed Method: SamBaTen

As we mention in the introduction, there exists a body of work in the literature that is able to efficiently and incrementally update the CP decomposition in the presence of incoming tensor slices [14, 21]. However, those methods fall short when the size of of the dynamically growing tensor increases, and eventually are not able to scale to very large dynamic tensors. The reason why this happens is because these methods operate on the *full data*, and thus, even though they incrementally update the decomposition (avoiding to re-compute it from scratch), inevitably, as the size of the full data grows, it takes a toll on the run-time and scalability.

In this paper we propose SamBaTen, which takes a different view of the solution, where instead of operating on the full data, it operates on a summary of the data. Suppose that the "complete" tensor (i.e., the one that we will eventually get when we finish receiving updates) is denoted by $\underline{\mathbf{X}}$. Any given incoming slice (or even a batch of slice updates) can be, thus, seen as a sample of that tensor, $\underline{\mathbf{X}}$ where the sampled indices in the third mode (which we assume is the one receiving the updates) are the indices of the incoming slice(s). Suppose, further, that given a set of sample tensors (which are drawn by randomly selecting indices from all the modes of the tensor) we can approximate the original tensor with high-accuracy (which, in fact, the literature has shown that it is possible [6, 5]). Therefore, when we receive a new batch of slices as an update, if we update those samples with the new indices, then we should be able to compute a decomposition very efficiently which incorporates the slice updates, and approximates the updated tensor well. A visual summary of SamBaTen is shown in Figure 3.

3.1 The heart of SamBaTen The algorithmic framework we propose is shown in Figure 3 and is described below: We assume that we have an existing set of decomposition results, as well as a set of *summaries* of the tensor, before the update. Summaries are in the form of sampled sub-tensors, as described in the text below. For simplicity of description, we assume that we are receiving updated slices on the third mode, which in turn have to add new rows to the $\mathbf{C}$ matrix (that

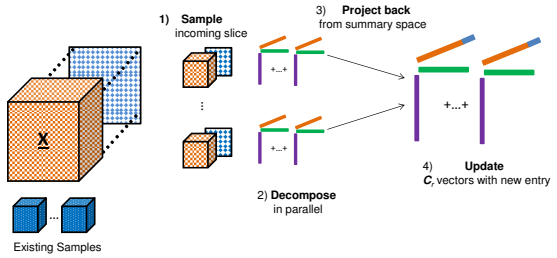

Figure 3: SamBaTen: Sampling-based Batch Incremental Tensor Decomposition: 1) Sample incoming tensor into sub-tensors, 2) run parallel decompositions on the samples, 3) project back the results into the original space, and, finally, 4) update the incrementally growing factor matrix C.

corresponds to the third mode). We, further, assume that the updates come in batches of new slices, which, in turn, ensures that we see a mature-enough update to the tensor, which contains useful structure. Trivially, however, SamBaTen can operate on singleton batches. In the following lines, $\underline{\mathbf{X}}$ is the tensor prior to the update and $\underline{\mathbf{X}}_{new}$ is the batch of incoming slices. Given an update, SamBaTen performs the following steps:

Sample: The rationale behind SamBaTen is that each batch $\underline{\mathbf{X}}_{new}$ can be seen as a sample of third-mode indices of (what is going to be) the full tensor. In this step, we are going to merge these incoming indices with an already existing set of sampled tensors. In order to obtain those pre-existing samples, we follow a similar approach to [6]. Namely, we sample indices from the tensor $\underline{\mathbf{X}}$ based on a measure of importance. To determine the importance for each mode m and then sample the indices using this measure as a sampling weight divided by its probability. An appropriate measure of importance (MoI) is the **sum-of-squares** of the tensor for each mode. For the first mode , MoI is defined as: $x_a(i) = \sum_{j=1}^{J}\sum_{k=1}^{K}\underline{\mathbf{X}}(i,j,k)^2$ for $i \in$ (1,I). Similarly, we can define the MoI for modes 2 and 3.

We sample each mode of $\underline{\mathbf{X}}$ without replacement, using the above MoI to bias the sampling probabilities. With s as sampling factor, i.e. if $\underline{\mathbf{X}}$ has size $I \times J \times K$, then $\underline{\mathbf{X}}_s$ will be of size $\frac{I}{s}, \frac{J}{s}, \frac{K}{s}$. Sampling rate for each mode is independent from each other, and in fact, different rates can be used for imbalanced modes. In the case of sparse tensors, the sample will focus on the dense regions of the tensor which contains most of the useful structure. In the case of dense tensors, the sample will give priority to the regions of the tensor with the highest variation.

After forming the sample summary $\underline{\mathbf{X}}_s$ for $\underline{\mathbf{X}}$, we merge it with the samples obtained from the intersection of the third-mode indices of $\underline{\mathbf{X}}_{new}$ and the already sampled indices in the remaining modes, so that the final sample is equal to $\underline{\mathbf{X}}_s$=$\underline{\mathbf{X}}(I_s, J_s, K_s \cup [K+1 \cdots K_{new}])$, where $K+1 \cdots K_{new}$ are the third-mode indices of $\underline{\mathbf{X}}_{new}$.

Due to the randomized nature of this summarization, we need to draw multiple samples, in order to obtain a reliable set of summaries. Each such independent sample is denoted as $\underline{\mathbf{X}}_s^{(r)}$. In the case of dense tensors, obtaining multiple, independent random samples helps summarize as much of the useful variation as possible. In fact, we will see in the experimental evaluation that increasing the number of samples, especially for dense tensors, improves accuracy.

In [6] the authors note that in order for their method to work, a set of anchor indices must be common between all samples, so that, later on, we can establish the correct correspondence of latent factors. However, in SamBaTen we do not have to actively fix a set of indices across sampling repetitions. When we sample I_s, J_s, K_s each time, those indices correspond to a portion of the decomposition that is already computed. Therefore, the entire set of indices I_s, J_s, K_s can serve as the set of anchors. This is a major advantage compared to [6], since SamBaTen 1) does not need to commit to a set of fixed indices for all samples a-priori, which, due to randomization may happen to represent a badly structured portion of the tensor, 2) does not need to be restricted in a "small" set of fixed common indices (which is required in [6] in order to ensure that sufficiently enough new indices are sampled across repetitions), but to the contrary, is able to use a larger number of anchor indices to establish correspondence more reliably, and 3) does not require any synchronization between different sampling repetitions, which results in higher parallelism potential.

Decompose: Having obtained $\underline{\mathbf{X}}_s$, from the previous step, SamBaTen decomposes the summary using any state-of-the-art algorithm, obtaining factor matrices $[\mathbf{A}_i^{'}, \mathbf{B}_i^{'}, \mathbf{C}_i^{'}]$. For the purposes of this paper, we use the Alternating Least Squares (ALS) algorithm for the CP decomposition, which is probably the most well studied algorithm for CP.

Project back: The CP decomposition is unique (under mild conditions) up to permutation and scaling of the components [16]. This means that, even though the existing decomposition $[\mathbf{A}_{old}, \mathbf{B}_{old}, \mathbf{C}_{old}]$ may have established an order of the components, the decomposition $[\mathbf{A}_i^{'}, \mathbf{B}_i^{'}, \mathbf{C}_i^{'}]$ we obtained in the previous step is very likely to introduce a different ordering and scaling, as a result of the aforementioned permutation and scaling ambiguity. Formally, the sampled portion of the existing factors and the currently computed factors are

Algorithm 1: SAMBATEN

Input: Tensor $\underline{\mathbf{X}}_{new}$ of size $I \times J \times K_{new}$, Factor matrices $\mathbf{A}_{old}, \mathbf{B}_{old}, \mathbf{C}_{old}$ of size $I \times R$, $J \times R$ and $K_{old} \times R$ respectively, sampling factor s and number of repetitions r.
Output: Factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ of size $I \times R$, $J \times R$ and $(K_{new} + K_{old}) \times R$, λ.

1: **for** $i = 1$ **to** r **do**
2: Compute $\mathbf{x}_a, \mathbf{x}_b$ and $\mathbf{x}_c$.
3: Sample a set of indices I_s, J_s, K_s from $\underline{\mathbf{X}}$ without replacement using $\mathbf{x}_a(i) / \sum_{i=1}^{I} x_a(i)$ as probability (accordingly for the rest).
4: $\underline{\mathbf{X}}_s = \underline{\mathbf{X}}(I_s, J_s, K_s \cup [K+1 \cdots K_{new}])$
5: $[\mathbf{A}_i^{'}, \mathbf{B}_i^{'}, \mathbf{C}_i^{'}] = \text{CP}(\underline{\mathbf{X}}_s, R)$.
6: Normalize $\mathbf{A}_i^{'}, \mathbf{B}_i^{'}, \mathbf{C}_i^{'}$ (as in the text) and absorb scaling in $\boldsymbol{\lambda}$.
7: Compute optimal matching between the columns of $\mathbf{A}_{old}, \mathbf{B}_{old}, \mathbf{C}_{old}$ and $\mathbf{A}_i^{'}, \mathbf{B}_i^{'}, \mathbf{C}_i^{'}$ as in the text
8: Update only zero entries of $\mathbf{A}, \mathbf{B}, \mathbf{C}$ that correspond to sampled entries of $\mathbf{A}_i^{'}, \mathbf{B}_i^{'}, \mathbf{C}_i^{'}$
9: Obtain $\mathbf{C}_{new}$ of size $K_{new} \times R$ by taking last K_{new} rows of $\mathbf{C}^{'}$
10: Use a shared copy of $\mathbf{C}_{new}$ and average out its entries in a column-wise fashion across different sampling repetitions.
11: **end for**
12: Update $\mathbf{C}$ of size $(K_{new} + K_{old}) \times R$ as : $C = \begin{bmatrix} \mathbf{C}_{old} \\ \mathbf{C}_{new} \end{bmatrix}$
13: Update scaling $\boldsymbol{\lambda}$ as the average of the previous and new value.
14: **return** $\mathbf{A}, \mathbf{B}, \mathbf{C}, \boldsymbol{\lambda}$

connected through the following relation:

$$[\mathbf{A}_{old}(I_s,:), \mathbf{B}_{old}(J_s,:), \mathbf{C}_{old}(K_s,:)] =$$
$$[\mathbf{A}_i^{'}(I_s,:)\boldsymbol{\Lambda}\boldsymbol{\Pi}, \mathbf{B}_i^{'}(J_s,:)\boldsymbol{\Pi}, \mathbf{C}(K_s,:)_i^{'}\boldsymbol{\Pi}]$$

where $\boldsymbol{\Lambda}$ is a diagonal scaling matrix (which without loss of generality we absorb on the first factor), and $\boldsymbol{\Pi}$ is a permutation matrix that permutes the order of the components (columns of the factors).

In order to tackle the scaling ambiguity, we need to normalize the results in a consistent manner. In particular, we normalize such that each column of the newly computed factors which spans the indices that are shared with $[\mathbf{A}_{old}, \mathbf{B}_{old}, \mathbf{C}_{old}]$ has unit norm: $\mathbf{A}_i^{'}(:, f) = \frac{\mathbf{A}_i^{'}}{||\mathbf{A}_i^{'}(I_s, f)||_2}$, and accordingly for the remaining factors. Note that for $\mathbf{A}_i^{'}$, trivially holds that $\mathbf{A}_i^{'}(I_s, f) = \mathbf{A}_i^{'}(:, f)$ and similarly for $\mathbf{B}_i^{'}$. After normalizing, the relation between the existing factors and the currently computed is $\mathbf{A}_{old}(I_s,:) = \mathbf{A}_i^{'}\boldsymbol{\Pi}$ (and similarly for the rest). Each iteration retains a copy of $[\mathbf{A}_{old}(I_s,:), \mathbf{B}_{old}(J_s,:), \mathbf{C}_{old}(K_s,:)]$ which will serve as the anchor for disambiguating the permutation of components. We normalize $[\mathbf{A}_{old}(I_s,:), \mathbf{B}_{old}(J_s,:), \mathbf{C}_{old}(K_s,:)]$ to unit norm as well, and the reason behind that lies in Lemma 3.1:

LEMMA 3.1. *Consider* $\mathbf{a} = \mathbf{A}_i^{'}(:, f_1)$ *and* $\mathbf{b} = \mathbf{A}_{old}(:, f_2)$. *If* f_1 *and* f_2 *correspond to the same latent CP factor, in the noiseless case, then* $\mathbf{a}^T\mathbf{b} = 1$ *otherwise* $\mathbf{a}^T\mathbf{b} < 1$.

Proof. From Cauchy-Schwartz inequality $\mathbf{a}^T\mathbf{b} \leq \|\mathbf{a}\|_2\|\mathbf{b}\|_2$. The above inequality is maximized when $\mathbf{a} = \mathbf{b}$ and for unit norm $\mathbf{a}, \mathbf{b}$, $\mathbf{a}^T\mathbf{b} \leq 1$. Therefore, if $\mathbf{a} = \mathbf{b}$, which happens when f_1 and f_2 correspond to the same latent CP factor, $\mathbf{a}^T\mathbf{b} = 1$.

Lemma 3.1 is a guide for identifying the permutation matrix $\boldsymbol{\Pi}$: For every column of $\mathbf{A}_i^{'}$ we compute the inner product with every column of $\mathbf{A}_{old}(I_s,:)$ and compute a matching when the inner product is equal (or close) to 1. Given a large-enough number of rows for $\mathbf{A}_i^{'}$ (which is usually the case, since we require a large-enough sample of the tensor in order to augment it with the update and compute the factor updates accurately), this matching can be computed reliably even in noisy real-world data, as we show in the experimental evaluation.

Update results: After appropriately permuting the columns of $\mathbf{A}_i^{'}, \mathbf{B}_i^{'}, \mathbf{C}_i^{'}$, we have all the information needed to update our model. Returning to the problem definition of Section 2, $\mathbf{A}_i^{'}$ contains the updates to the rows within I_s for $\mathbf{A}(t)$ (and similarly for $\mathbf{B}$ and $\mathbf{C}$). Even though $\mathbf{A}, \mathbf{B}$ do not increase their number of rows over time, the incoming slices may contribute valuable new estimates to the already estimated factors. Thus, for the already existing portions of $\mathbf{A}, \mathbf{B}, \mathbf{C}$ we only update the zero entries that fall within the range of I_s, J_s, and K_s respectively. Finally, $\mathbf{C}_i^{'}([K+1 \cdots K_{new}],:)$ contains the factors for the newly arrived slices, which need to be merged to the already existing columns of $\mathbf{C}$. After properly permuting the columns of $\mathbf{C}_i^{'}$, we accumulate the lower portion of the $\mathbf{C}_i^{'}$ (corresponding to the new slices) into $\mathbf{C}_{new}$ and we take the column-wise average of the rows to-be-appended to $\mathbf{C}$, across repetitions. Finally, we update $\mathbf{C}(t') = \begin{bmatrix} \mathbf{C}_{old} \\ \mathbf{C}_{new} \end{bmatrix}$.

Guarantees for Correctness Lemma 3.1 is essentially providing a guarantee for the correctness of SAMBATEN. In particular, Lemma 3.1 ensures that, under mild conditions that make the CP decomposition unique and identifiable, SAMBATEN will discover the correct latent factors, disambiguate the permutation and scaling ambiguity, and update the correct columns of the factor matrix that is to be augmented. As we will also empirically demonstrate in the experimental evaluation, indeed, SAMBATEN is able to produce correct results that are on par with state-of-the-art methods.

4 Experimental Evaluation

In this section we extensively evaluate the performance of SAMBATEN on multiple synthetic and real datasets, and compare its performance with state-of-the-art approaches. We experiment on the different parameters of SAMBATEN and the baselines, and how that affects performance. We implemented SAMBATEN in Matlab

using the functionality of the Tensor Toolbox for Matlab [1] which supports very efficient computations for sparse tensors. Our implementation is available at link [2]. We used Intel(R) Xeon(R), CPU E5-2680 v3 @ 2.50GHz machine with 48 CPU cores and 378GB RAM.

4.1 Data-set description Below, we describe the process for generating synthetic data and the real datasets we used.

4.1.1 Synthetic data generation We generate random tensors of dimension $I = J = K$ with increasing I. Those tensors are created from a known set of randomly generated factors, so that we have full control over the ground truth of the full decomposition. We dynamically calculate the size of batch or slice for our all experiments to fit the data into memory. The machine used in this experiments has ≈380GB of memory. For example, in case of $I = J = K = 50000$, batch size is up to 5, the dense incoming slice requires memory space equivalent to ≈80 GB and the rest of memory space is used for the computations. The specifications of each synthetic dataset are given in Table 2.

Dimension ($I = J = K$)	Density-dense	Density-sparse	Batch size	Sampling factor
100	100%	65%	50	2
500	100%	65%	150	2
1000	100%	55%	150	2
3000	100%	55%	100	5
5000	100%	55%	100	5
10000	100%	55%	10	2
50000	100%	35%	5	2
100000	100%	35%	5	2

Table 2: Table of Datasets analyzed

4.1.2 Real Data Description In order to truly evaluate the effectiveness of SamBaTen, we test its performance against six real datasets that have been used in the literature. Those datasets are summarized in Table 3 and are publicly available at [17] .

4.2 Evaluation Measures We evaluate SamBaTen and the baselines using three criteria: Relative Error, Wall-Clock time and Fitness. These measures provide a quantitative way to compare the performance of our method. More Specifically, **Relative Error** is effectiveness measurement and defined as :

$$RelativeError = \frac{||\underline{\mathbf{X}}_{original} - \underline{\mathbf{X}}_{predicted}||}{||\underline{\mathbf{X}}_{original}||}$$

where, the lower the value, the better.

[2] http://www.cs.ucr.edu/~egujr001/ucr/madlab/src/SAMBATEN.zip

CPU time (sec): The average running time denoted by T_{tot} for processing all slices for given tensor, measured in seconds, and is used to validate the time efficiency of an algorithm.

Relative Fitness: Relative Fitness is defined as:

$$RelativeFitness = \frac{||\underline{\mathbf{X}}_{original} - \underline{\mathbf{X}}_{\text{SamBaTen}}||}{||\underline{\mathbf{X}}_{original} - \underline{\mathbf{X}}_{BaseLine}||}$$

where, again, lower is better.

4.3 Baselines for Comparison Here we briefly present the state-of-the-art baselines we used for comparison. Note that for each baseline we use the *reported parameters* that yielded the best performance in the respective publications. For fairness, we compare against the parameter configuration for SamBaTen that yielded the best performance in terms of low wall-clock timing, low relative error and fitness. Note that all comparisons were carried out over 10 iterations each, and each number reported is an average with a standard deviation attached to it.

CP_ALS [1]: is considered the most standard and well optimized algorithm for CP. We use the implementation of the Tensor Toolbox for Matlab [1]. Here, we simply re-compute CP using CP_ALS after every update.

SDT [14]: Simultaneous Diagonalization Tracking (SDT) is based on incrementally tracking the Singular Value Decomposition (SVD) of the unfolded tensor $\underline{\mathbf{X}}_{(3)} = U \sum V^T$. **RLST [14]**: Recursive Least Squares Tracking (RLST) is another online approach in which recursive updates are computed to minimize the Mean Squared Error (MSE) on incoming slice. **OnlineCP [21]**: This is the most recent and state-of-the-art method in online computation of CP.

We conduct our experiments on multiple synthetic datasets and six real-world tensors datasets. We set the tolerance rate for convergence between consecutive iterations to 10^{-5} and the maximum number of iteration to 1000 for all the algorithms. The batch size and sampling factor is selected based on dimensions of first mode i.e. I, provided in Table 2 and 3 for synthetic and real dataset respectively. We use the publicly available implementations for the baselines, as provided by the authors. We only modified the interface of the baselines, so that it is consistent across all methods with respect to the way that they receive the incoming slices. No other functionality has been changed.

4.4 Experimental Results The following major three aspects are analyzed.

Q1. Effectiveness and Accuracy: How effective is SamBaTen as compared to the baselines on different synthetic and real world datasets?

Name	Description	Dimensions	NNZ	Batch size	Sampling factor	Dataset File Size
NIPS [8]	(Paper,Author,Word)	2,482 x 2862 x 14036	3,101,609	500	10	57MB
NELL [2]	(Entity,Relation,Entity)	12092 x 9184 x 28818	76,879,419	500	10	1.4GB
Facebook-wall [20]	(Wall owner, Poster, day)	62,891 x 62,891 x 1,070	78,067,090	100	5	2.1GB
Facebook-links [20]	(User, Links, Day)	62,891 x 62,891 x 650	263,544,295	50	2	3.8GB
Patents[17]	(Term ,Term, Year)	239,172 x 239,172 x 46	3,596,640,708	10	2	73GB
Amazon[13]	(User, Product ,Word)	4,821,207 x 1,774,269 x 1,805,187	1,741,809,018	50000	20	43GB

Table 3: Real datasets analyzed

Q2. Speed & Scalability: How fast is SAMBATEN when compared to the state-of-the-art methods on very large sized datasets?
Q3. Parameter Sensitivity: What is the influence of sampling factor s and sampling repetitions r?

4.4.1 Baselines for Comparison For all datasets we compute Relative Error,CPU time (sec) and Fitness. For SAMBATEN, CP_ALS , OnlineCP, RSLT and SDT we use 10% of the data in each dataset as existing dataset. We experimented for both dense as well as sparse tensor to check the performance of our method. The results for the dense and sparse synthetic data are shown in Table 4 - 5. For each of datasets , the best result is shown in bold. OnlineCP, SDT and RLST address the issue very well. Compared with CP_ALS, SDT and RLST reduce the mean running time by up to 2x times and OnlineCP reduce mean time by up to 3× for small dataset (I up to 3000). Performance of RLST was better than SDT algorithm on 8 out of 8 third-order synthetic tensor datasets. In fact, the efficiency (in terms of CPU time (sec)) of SDT is quite close to RLST. However, the main issue of SDT and RLST is their estimation of relative error and fitness. For some datasets, such as $I = 100$ and $I = 3000$, they perform well, while for some others, they exhibit poor fitness and relative error, achieving only nearly half of the fitness of other methods. For *small size* datasets, OnlineCP's efficiency and accuracy is better than all methods. As the dimension grows, however, the performance of OnlineCP method reduces,and particularly for datasets of dimension larger than $5000 \times 5000 \times 5000$. Same behavior is observed for sparse tensors. SAMBATEN is comparable to baselines for small dataset and outperformed the baselines for large dataset. CP_ALS is the only baseline able to run on datasets up to size $3000 \times 3000 \times 3000$. These results answer **Q1** as the SAMBATEN have comparable accuracy to other baseline methods.

SAMBATEN is efficiently able to compute $100K \times 100K \times 100K$ sized tensor with batch size of 5 and sampling factor 2. It took **58095.72s** and **47232.2s** to compute online decomposition for dense and sparse tensor, respectively. On other hand, state-of-art methods are unable to handle such large scaled incoming data.

Table 6 shows the comparison between methods. SAMBATEN outperforms other state-of-the-art approaches in most of the real dataset. In the case of NIPS datset, SAMBATEN gives better results compared to the baselines, specifically in terms of CPU Time (*faster up to 20 times*) and Fitness (*better up to 15-20%*). SAMBATEN outperforms for NELL, Facebook-Wall and Facebook-Links dataset in terms of efficiency comparable to CP_ALS. For the NIPS dataset, SAMBATEN is *25-30 times faster* than OnlineCP method. Due to high dimensions of dataset, RSLT and SDT are unable to execute further. Note that all the real datasets we use are highly sparse, however, no baselines except CP_ALS actually take advantage of that sparsity, therefore, repeated CP_ALS tends to be faster because the baselines have to deal with dense computations which tend to be slower, when the data contain a lot of zeros. Most importantly, however, SAMBATEN performed very well on Amazon and Patent datasets, arguably the hardest of the six real datasets we examined and have been analyzed in the literature, *where none of the baselines was able to run.* These result answer **Q1** and **Q2** and show that SAMBATEN is able to handle large dimensions and sparsity.

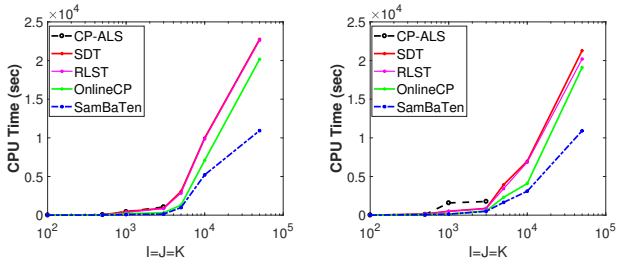

Figure 4: Experimental results for CPU time (sec) for (a) dense tensor (b) sparse tensor

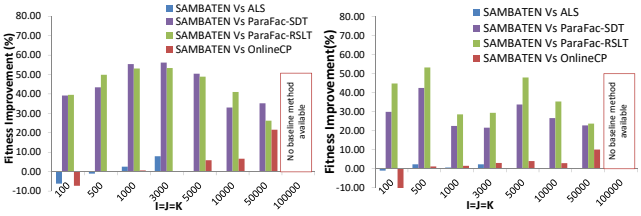

Figure 5: Experimental results for Relative Fitness Improvement for (a) dense tensor (b) sparse tensor

4.4.2 Sensitivity of Sampling Factor s The sampling factor plays an important role in SAMBATEN. We

I=J=K	CP_{ALS}	OnlineCP	SDT	RLST	SamBaTen
100	0.109 ± 0.01	**0.107± 0.02**	0.173± 0.02	0.151± 0.02	0.115± 0.02
500	**0.102 ± 0.09**	**0.102± 0.09**	0.217± 0.06	0.217± 0.06	**0.102± 0.09**
1000	0.103± 0.01	0.103± 0.01	0.287± 0.01	0.296± 0.01	**0.102± 0.01**
3000	0.119± 0.01	**0.108± 0.01**	0.189± 0.01	0.206± 0.01	0.109± 0.01
5000	N/A	0.122± 0.002	0.201± 0.002	0.196± 0.04	**0.115± 0.009**
10000	N/A	0.173± 0.04	0.225± 0.04	0.252± 0.06	**0.162± 0.01**
50000	N/A	0.215± 0.03	0.229± 0.03	0.26± 0.01	**0.169± 0.01**
100000	N/A	N/A	N/A	N/A	**0.275 ± 0.00**

Table 4: Experimental results for relative error for synthetic dense tensor. We see that SamBaTen gives comparable accuracy to baseline.

I=J=K	CP_{ALS}	OnlineCP	SDT	RSLT	SamBaTen
100	0.169± 0.01	**0.154± 0.02**	0.306± 0.01	0.313± 0.01	0.178± 0.01
500	**0.175± 0.01**	0.188± 0.01	0.43± 0.01	0.421± 0.01	0.184± 0.01
1000	0.179± 0.01	0.185± 0.01	0.613± 0.01	0.813± 0.01	**0.178± 0.01**
3000	0.177± 0.02	**0.171± 0.04**	0.446± 0.03	0.513± 0.02	0.176± 0.03
5000	N/A	0.192± 0.02	0.494± 0.09	0.535± 0.13	**0.187± 0.04**
10000	N/A	**0.173± 0.01**	0.212± 0.01	0.224± 0.11	0.198± 0.12
50000	N/A	0.222± 0.00	0.262± 0.00	0.259± 0.00	**0.200± 0.00**
100000	N/A	N/A	N/A	N/A	**0.283± 0.00**

Table 5: Experimental results for relative error for synthetic sparse tensor. We see that SamBaTen works better in very large scale dataset such as 50000 × 50000 × 50000.

Dataset	CPU Time (sec)					Fitness SamBaTen w.r.t			
	CP_{ALS}	OnlineCP	SDT	RSLT	SamBaTen	CP_{ALS}	OnlineCP	SDT	RSLT
NIPS	177.46± 2.9	372.03± 8.9	1608.23± 37.5	1596.07± 15.2	**43.98± 0.6**	0.96± 0.01	0.98± 0.01	**0.78± 0.02**	0.82± 0.01
NELL	8783.27± 11.2	42325.22± 70.0	65325.22± 25.2	63485.98± 10.6	**983.83± 30.7**	0.95±0.02	0.81± 0.01	**0.76± 0.02**	0.81± 0.01
Facebook-wall	3041.98±3.8	N/A	N/A	N/A	**736.07±4.1**	0.97±0.01	N/A	N/A	N/A
Facebook-links	2689.69±7.9	N/A	N/A	N/A	**343.32±6.3**	0.96±0.06	N/A	N/A	N/A
Amazon	N/A	N/A	N/A	N/A	**4892.07±61.8**	N/A	N/A	N/A	N/A
Patent	N/A	N/A	N/A	N/A	**8068.27±55.4**	N/A	N/A	N/A	N/A

Table 6: SamBaTen performance for real datasets. SamBaTen outperforms the baselines for all the large tensors.

performed experiments to evaluate the impact of changing sampling factor on SamBaTen. For these experiments , we fixed batch size to 50 for all datasets. We see in figure 6 that increasing sampling factor results in reduction of CPU time (as sparsity of sub sampled tensor increased) and it reduces the fitness of output up-to 2-3%. In sum, these observations demonstrate that: 1) a suitable sampling factor on sub-sampled tensor could improve the fitness and result in better tensor decomposition, and 2) the higher sampling factor is, the lower the CPU time. This result partially answers **Q3**.

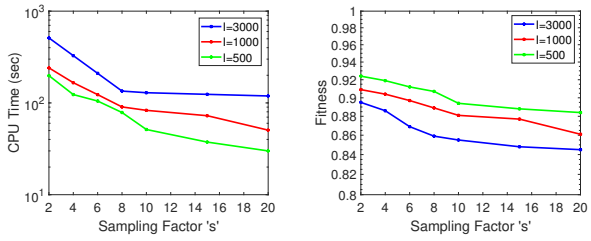

Figure 6: SamBaTen CPU Time (sec) and Relative Fitness vs. Sampling Factor s on different datasets (*lower is better*).

4.4.3 Sensitivity of Repetition Factor r

We evaluate the performance for parameter setting r i.e the number of parallel decompositions. For these experiments, we choose batch size and sampling rate for synthetic 500 × 500 × 500 dataset and NIPS real world dataset as provided in table 2 and 3, respectively. We can see that with higher values of the repetition factor r, Relative Fitness (SamBaTen vs CP_ALS) is improved as shown in Figure 7(a). We experiment on varying repetition factor r with Sampling factor s on NIPS real world dataset to check the performance of our method as shown in Figure 7(b). The lower the relative fitness score, the better the decomposition. This result completes the answer to **Q3**.

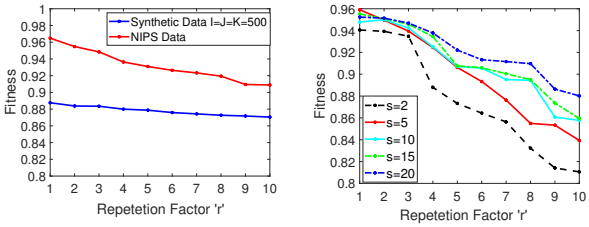

Figure 7: SamBaTen Relative Fitness vs. repetition factor r on synthetic and NIPS datasets (*lower is better*).

5 Related Work

Incremental tensor methods in the literature can be categorized into three main categories: 1) Tucker decomposition, 2) CP decomposition, 3) Tensor completion

Tucker Decomposition: Online tensor decomposition was first proposed by Sun *el at.*[19] as ITA (Incremental Tensor Analysis), describing the three variants of Incremental Tensor Analysis. Liu *el at.*[15] proposed an efficient method to diagonalize the core tensor to overcome this problem. Hadi *el at.* [7] proposed the multi-aspect-streaming tensor analysis (MASTA) method that and allows the tensor to simultaneously grow in all modes.

CP Decomposition: There is very limited study on online CP decomposition methods. Phan *el at.* Nion *el at.*[14], proposed two algorithms that focus on CP

decomposition namely SDT (Simultaneous Diagonalization Tracking) and RLST (Recursive Least Squares Tracking). The latest related work is OnlineCP, proposed by Zhou, *el at.* [21].

Tensor Completion: The main difference between completion and decomposition techniques is that in completion "zero" values are considered "missing" and are not part of the model, and the goal is to impute those missing values accurately, rather than extracting latent factors from the observed data. The earliest work on incremental tensor completion traces back to [12], and recently, Qingquan *el at.*[18], proposed streaming tensor completion based on block partitioning.

6 Conclusions

We introduce SamBaTen, a novel sample-based incremental CP tensor decomposition. We show its effectiveness with respect to approximation quality, with its performance being on par with state-of-the-art incremental and non-incremental algorithms, and we demonstrate its efficiency and scalability by outperforming state-of-the-art approaches (*25-30 times faster*) and being able to run very large incremental tensors where none of the baselines was able to produce results.

7 Acknowledgments

Research was supported by the Department of the Navy, Naval Engineering Education Consortium under Award no. N00174-17-1-0005, by the National Science Foundation Grant no. EAGER 1746031, and an Adobe Data Science Research Faculty Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding parties. We would also like to thank Xia Ben Hu for fruitful discussions on the problem.

References

[1] B. W. Bader, T. G. Kolda, et al. Matlab tensor toolbox version 2.6, available online, february 2015, 2015.

[2] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3, 2010.

[3] J. D. Carroll and J.-J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. *Psychometrika*, 35(3):283–319, 1970.

[4] E. C. Chi and T. G. Kolda. On tensors, sparsity, and nonnegative factorizations. *SIAM Journal on Matrix Analysis and Applications*, 33(4):1272–1299, 2012.

[5] D. Erdos and P. Miettinen. Walk'n'merge: A scalable algorithm for boolean tensor factorization. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 1037–1042. IEEE, 2013.

[6] Evangelos E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos. Parcube: Sparse parallelizable tensor decompositions. In *ECML-PKDD'12*.

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

[8] A. Globerson, G. Chechik, F. Pereira, and N. Tishby. Euclidean Embedding of Co-occurrence Data. *The Journal of Machine Learning Research*, 8:2265–2295, 2007.

[9] R. Harshman. Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis. 1970.

[10] T. Kolda and B. Bader. Tensor decompositions and applications. *SIAM review*, 51(3), 2009.

[11] T. G. Kolda, B. W. Bader, and J. P. Kenny. Higher-order web link analysis using multilinear algebra. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005.

[12] M. Mardani, G. Mateos, and G. B. Giannakis. Subspace learning and imputation for streaming big data matrices and tensors. *IEEE Transactions on Signal Processing*, 63(10):2663–2677, 2015.

[13] J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172. ACM, 2013.

[14] D. Nion and N. Sidiropoulos. Adaptive algorithms to track the parafac decomposition of a third-order tensor. *Signal Processing, IEEE Transactions on*, 57(6):2299–2310, 2009.

[15] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *Proceedings of the 31st international conference on Very large data bases*, pages 697–708. VLDB Endowment, 2005.

[16] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos. Tensors for data mining and data fusion: Models, applications, and scalable algorithms. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):16, 2016.

[17] S. Smith, J. W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, and G. Karypis. FROSTT: The formidable repository of open sparse tensors and tools, 2017.

[18] Q. Song, H. G. Xiao Huang, J. Caverlee, and X. Hu. Multi-aspect streaming tensor completion. In *Proceedings of the 23th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2017.

[19] J. Sun, D. Tao, S. Papadimitriou, P. S. Yu, and C. Faloutsos. Incremental tensor analysis: Theory and applications. *ACM Trans. Knowl. Discov. Data*, 2(3):11:1–11:37, Oct. 2008.

[20] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM workshop on Online social networks*, pages 37–42. ACM, 2009.

[21] S. Zhou, N. X. Vinh, J. Bailey, Y. Jia, and I. Davidson. Accelerating online cp decompositions for higher order tensors. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1375–1384. ACM, 2016.