OCTEN: ONLINE COMPRESSION-BASED TENSOR DECOMPOSITION

Ekta Gujral, Ravdeep Pasricha, Tianxiong Yang, Evangelos E. Papalexakis

Department of Computer Science, UC Riverside, California, USA
Email: (egujr001, rpasr001, tyang022)@ucr.edu, epapalex@cs.ucr.edu

ABSTRACT

Tensor decompositions are powerful tools for large data analytics, as they jointly model multiple aspects of data into one framework and enable the discovery of the latent structures and higher-order correlations within the data. One of the most widely studied and used decompositions, especially in data mining and machine learning, is the Canonical Polyadic or PARAFAC decomposition. However, today's datasets are not static and often grow and change over time. To operate on such large dynamic data, we present OCTEN, the first ever compression-based online parallel implementation for the CP/PARAFAC decomposition. We conduct an extensive empirical analysis of the algorithms in terms of fitness, memory used and CPU time and in order to demonstrate the compression and scalability of the method, we apply OCTEN to big tensor data. Indicatively, OCTEN performs on-par or better than state-of-the-art online and offline methods in terms of decomposition accuracy and efficiency, while achieving memory savings ranging in *40-200%*.

1. INTRODUCTION

A Tensor is a multi-way array of elements that represents higher-order or multi-aspect data. In recent years, tensor decompositions have gained increasing popularity in big data analytics [11]. Tensor decompositions are capable of finding complex patterns and higher-order correlations within the data. In the era of information explosion, data is generated or modified in large volume. In such environments, data may be added or removed from any of the dimensions with high velocity. When using tensors to represent this dynamically changing data, an instance of the problem is that of a "streaming", "incremental", or "online" tensors.

As the volume and velocity of data grow, the need for time- and space-efficient online tensor decomposition is imperative. There already exists a modest amount of prior work in online tensor decomposition both for Tucker [1, 14] and CP [9, 18, 16]. However, most of the existing online methods [1, 18, 9] , model the data in the full space, which can become very memory taxing as the size of the data grows. There exist memory efficient tensor decompositions, indicatively MET for Tucker[8] and PARACOMP [13] for CP/PARAFAC, neither of which are able to handle online tensors. In this paper, we fill that gap. Our contributions are summarized as follows:

Research was supported by the Department of the Navy, Naval Engineering Education Consortium under award no. N00174-17-1-0005, National Science Foundation CDS&E Grant no. OAC-1808591 and by an Adobe Data Science Research Faculty Award.

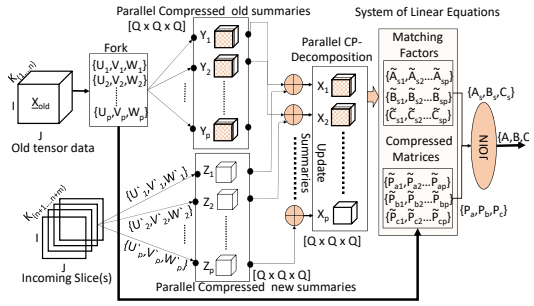

Fig. 1: Framework. Compressed tensor summaries $\underline{\mathbf{Y}}_\mathbf{p}$ and $\underline{\mathbf{Z}}_\mathbf{p}$ are obtained by applying randomly generated compression matrices $(\mathbf{U}_p, \mathbf{V}_p, \mathbf{W}_p)$ and $(\mathbf{U}_p^{'}, \mathbf{V}_p^{'}, \mathbf{W}_p^{'})$ to $\underline{\mathbf{X}}_{old}$ and $\underline{\mathbf{X}}_{new}$ respectively. The updated summaries are computed by $\underline{\mathbf{X}}_p$ = $\underline{\mathbf{Y}}_p + \underline{\mathbf{Z}}_p$. Each $\underline{\mathbf{X}}_p$ is independently decomposed in parallel. The update step anchors all compression and factor matrices to a single reference i.e. $(\mathbf{P}_a, \mathbf{P}_b, \mathbf{P}_c)$ and $(\mathbf{A}_s, \mathbf{B}_s, \mathbf{C}_s)$, and solves a linear equation for the overall A, B, and C.

- **Novel Parallel Algorithm** We introduce OCTEN, a novel compression-based algorithm for online tensor decomposotion that admits an efficient parallel implementation. We do not limit to 3-mode tensors, our algorithm can easily handle higher-order tensor decompositions.
- **Correctness guarantees** By virtue of using random compression, OCTEN provides the *identifiability* of the underlying CP/PARAFAC decomposition in the presence of streaming updates.
- **Extensive Evaluation** Through experimental evaluation on various datasets, we show that OCTEN provides stable decompositions (with quality on par with state-of-the-art), while offering up to *40-250 %* memory space savings.

2. PROBLEM FORMULATION

Given (a) an existing set of *summaries* $\{\underline{\mathbf{Y}}_1, \underline{\mathbf{Y}}_2 \dots \underline{\mathbf{Y}}_p\}$, which approximate tensor $\underline{\mathbf{X}}_{old}$ of size $\{ I^{(1)} \times I^{(2)} \times \dots I^{(N-1)} \times t_{old}\}$ at time t , (b) new incoming batch of slice(s) in form of tensor $\underline{\mathbf{X}}_{new}$ of size $\{I^{(1)} \times I^{(2)} \times \dots I^{(N-1)} \times t_{new}\}$, find updates of $(\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(N-1)}, \mathbf{A}^{(N)})$ **incrementally** to approximate tensor $\underline{\mathbf{X}}$ of dimension $\{I^{(1)} \times I^{(2)} \times \dots I^{(N-1)} \times I^{(N)}\}$ and rank R, where $I^{(N)} = (t_{old} + t_{new}) = I^{(N)}_{1\dots n} + I^{(N)}_{(n+1)\dots m}$ after appending new slice or tensor to N^{th} mode.

3. PROPOSED METHOD : OCTEN

In this section, we introduce OCTEN, a new method for parallel incremental decomposition designed with two main goals in mind: **G1**: Compression, speed, simplicity, and parallelization; and **G2**: correctness in recovering compressed partial results for incoming data, under suitable conditions. The algorithmic framework we propose is shown in Figure 1 and is described below:

We assume that we have a pre-existing set of *summaries* of the $\underline{\mathbf{X}}$ before the update. Summaries are in the form of compressed tensors of dimension $[Q \times Q \times Q]$. These are generated by multiplying random compression matrices $\{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$, that are independently obtain from an absolutely continuous uniform distribution with respect to the Lebesgue measure, with tensor's corresponding mode i.e. $\mathbf{U}$ is multiplied with I-mode and so on; see Figure 1 and Section 3.2 for its role in correct identification of factors.

In the following, $\underline{\mathbf{X}}_{old}$ is the tensor prior to the update and $\underline{\mathbf{X}}_{new}$ is the batch of incoming slice(s). Considering $S = \prod_{i=1}^{[N-1]} I^{(i)}$ and $T = \sum_{i=1}^{[N-1]} I^{(i)}$, we can write space and time complexity in terms of S and T. Given an incoming batch, OCTEN performs the following steps:

3.1. Parallel Compression and Decomposition

When handling large tensors $\underline{\mathbf{X}}$ that are unable to fit in main memory, we may compress the tensor $\underline{\mathbf{X}}$ to a smaller tensor that somehow apprehends most of the systematic variation in $\underline{\mathbf{X}}$. Keeping this in mind, for incoming slice(s) $\underline{\mathbf{X}}_{new}$, during the parallel compression step, we first need to create 'p' parallel triplets of random compression matrices (uniformly distributed) $\{\mathbf{U}_p, \mathbf{V}_p, \mathbf{W}_p\}$ of $\underline{\mathbf{X}}$. Thus, each worker (i.e. Matlab parpool) is responsible for creating and storing these triplets of size $\mathbf{U} \in \mathbb{R}^{I \times Q}$, $\mathbf{V} \in \mathbb{R}^{J \times Q}$ and $\mathbf{W} \in \mathbb{R}^{t_{new} \times Q}$. These matrices share at least 'shared' amount of column(s) among each other so that at final step we merge it correctly. Mathematically, we can describe it as follows:

$$\underline{\mathbf{X}} = \begin{bmatrix} \{\mathbf{U}_1, \mathbf{V}_1, \mathbf{W}_1\} \\ \{\mathbf{U}_2, \mathbf{V}_2, \mathbf{W}_2\} \\ \cdots \\ \{\mathbf{U}_p, \mathbf{V}_p, \mathbf{W}_p\} \end{bmatrix} = \begin{bmatrix} \{(\mathbf{u}\ \ \mathbf{U}_{1'}), (\mathbf{v}\ \ \mathbf{V}_{1'}), (\mathbf{w}\ \ \mathbf{W}_{1'})\} \\ \{(\mathbf{u}\ \ \mathbf{U}_{2'}), (\mathbf{v}\ \ \mathbf{V}_{2'}), (\mathbf{w}\ \ \mathbf{W}_{2'})\} \\ \cdots \\ \{(\mathbf{u}\ \ \mathbf{U}_{p'}), (\mathbf{v}\ \ \mathbf{V}_{p'}), (\mathbf{w}\ \ \mathbf{W}_{p'})\} \end{bmatrix} \tag{1}$$

where u, v and w are shared and have dimensions of $\mathbb{R}^{I \times m}$, $\mathbb{R}^{J \times m}$ and $\mathbb{R}^{t_{new} \times m}$, here $m = Q_{shared}$.

For compression matrices, we choose to assign each worker create a single row of each of the matrices to reduce the burden of creating an entire batch of $\{\mathbf{U}_p^{'}, \mathbf{V}_p^{'}, \mathbf{W}_p^{'}\}$ of $\underline{\mathbf{X}}_{new}$. We see that each worker is sufficient to hold these matrices in main memory. Now, we created compressed tensor replica or summaries $\{\underline{\mathbf{Z}}_1, \underline{\mathbf{Z}}_2 \ldots \underline{\mathbf{Z}}_p\}$ by multiplying each triplets of compression matrices and $\underline{\mathbf{X}}_{new}$;see Figure 1. $\underline{\mathbf{Z}}_p$ is 3-mode tensor of size $\mathbb{R}^{Q \times Q \times Q}$. Since Q is considerably smaller than [I ,J, K], we use $O(Q^3)$ of memory on each worker.

For $\underline{\mathbf{X}}_{old}$, we already have replicas $\{\underline{\mathbf{Y}}_1, \underline{\mathbf{Y}}_2 \ldots \underline{\mathbf{Y}}_p\}$ obtained from each triplets of compression matrices $\{\mathbf{U}_p, \mathbf{V}_p, \mathbf{W}_p\}$ and $\underline{\mathbf{X}}_{old}$; see Figure 1. In general, the compression comprises N-mode products which leads to overall complexity of $(Q_{(1)}St_{new} + Q_{(2)}St_{new} + Q_{(3)}St_{new} + \ldots Q_{(N)}St_{new})$ for dense tensor $\underline{\mathbf{X}}$, if the first mode is compressed first, followed by the second, and then the third mode and so on. We choose to keep $Q_1, Q_2, Q_3 \ldots Q_N$ of same order as well non-temporal dimensions are of same order in our algorithm, so time complexity of parallel compression step for N-mode data is $O(QSt_{new})$ for each worker. The *summaries* are always dense, because first mode product with tensor is dense, hence remaining mode products are unable to exploit sparsity. However, the size of summaries are extremely less than incoming data.

After appropriately computing *summaries* $\{\underline{\mathbf{Z}}_1, \underline{\mathbf{Z}}_2 \ldots \underline{\mathbf{Z}}_p\}$ for incoming slices, we need to update the old summaries $\{\underline{\mathbf{Y}}_1, \underline{\mathbf{Y}}_2 \ldots \underline{\mathbf{Y}}_p\}$ which were generated from previous data. We don't save entire $\underline{\mathbf{X}}_{old}$, and instead we only save the compressed summaries i.e. $\underline{\mathbf{Y}}$. Each worker reads its segment and process update in parallel as given below.

$$\begin{bmatrix} \underline{\mathbf{X}}_1 \\ \underline{\mathbf{X}}_2 \\ \vdots \\ \underline{\mathbf{X}}_p \end{bmatrix} = \begin{bmatrix} \underline{\mathbf{Y}}_1 \\ \underline{\mathbf{Y}}_2 \\ \vdots \\ \underline{\mathbf{Y}}_p \end{bmatrix} \oplus \begin{bmatrix} \underline{\mathbf{Z}}_1 \\ \underline{\mathbf{Z}}_2 \\ \vdots \\ \underline{\mathbf{Z}}_p \end{bmatrix} \cdot * \begin{bmatrix} \mathbf{W}_1^{'}(k,q) \\ \mathbf{W}_2(k,q) \\ \vdots \\ \mathbf{W}_p^{'}(k,q) \end{bmatrix} = \begin{bmatrix} (\mathbf{A}_{s(1)}, \mathbf{B}_{s(1)}, \mathbf{C}_{s(1)}) \\ (\mathbf{A}_{s(2)}, \mathbf{B}_{s(2)}, \mathbf{C}_{s(2)}) \\ \vdots \\ (\mathbf{A}_{s(p)}, \mathbf{B}_{s(p)}, \mathbf{C}_{s(p)}) \end{bmatrix} \tag{2}$$

where k is the number of slices of incoming tensor and q is the slice number for the compressed tensor. Further, for the decomposition step, we processed 'p' *summaries* on different workers, each one fitting the decomposition to the respective compressed tensor $\{\underline{\mathbf{X}}_1, \underline{\mathbf{X}}_2 \ldots \underline{\mathbf{X}}_p\}$ created by the compression step. We assume that the updated compressed tensor $\{\underline{\mathbf{X}}_1, \underline{\mathbf{X}}_2 \ldots \underline{\mathbf{X}}_p\}$ fits in the main memory, and performs in-memory computation. We denote p^{th} compressed tensor decompositions as $(\mathbf{A}_{s(p)}, \mathbf{B}_{s(p)}, \mathbf{C}_{s(p)})$ as discussed above. The data for each parallel worker $\underline{\mathbf{X}}_p$ can be uniquely decomposed, i.e. $(\mathbf{A}_p, \mathbf{B}_p, \mathbf{C}_p)$ is unique up to scaling and column permutation. Furthermore, parallel compression and decomposition is able to achieve Goal **G1**.

3.2. Factor match for identifiability

According to Kruskal [6], the CP decomposition is unique (under mild conditions) up to permutation and scaling of the components i.e. $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ factor matrices. Consider an 3-mode tensor $\underline{\mathbf{X}}$ of dimension I, J and K of rank R. If rank

$$r_c = F \implies K \geq R \,\&\, I(I-1)(J-1) \geq 2R(R-1), \tag{3}$$

then rank 1 factors of tensor $\underline{\mathbf{X}}$ can be uniquely computable[6, 7]. Kronecker product[2] property is described as $(\mathbf{U}^T \otimes \mathbf{V}^T \otimes \mathbf{W}^T)(\mathbf{A} \odot \mathbf{B} \odot \mathbf{C}) = ((\mathbf{U}^T\mathbf{A}) \odot (\mathbf{V}^T\mathbf{B}) \odot (\mathbf{W}^T\mathbf{C})) \approx (\widetilde{\mathbf{A}}, \widetilde{\mathbf{B}}, \widetilde{\mathbf{C}})$. Now combining Kruskal's uniqueness and Kronecker product property, we can obtain correct identifiable factors from *summaries* if

$$\min(Q, r_A) + \min(Q, r_B) + \min(Q, r_C) \geq 2R + 2 \tag{4}$$

where Kruskal-rank of $\mathbf{A}$, denoted as r_A, is the maximum r such that any r columns of $\mathbf{A}$ are linearly independent; see [13]. Hence, upon factorization of 3-mode $\underline{\mathbf{X}}_p$ into R components, we obtain $\mathbf{A} = a_p^T A \Pi_p \lambda_p^{(1/N)}$, where

a_p is shared among summaries decompositions, Π_p is a permutation matrix, and λ_p is a diagonal scaling matrix obtained from CP decomposition. To match factor matrices after decomposition step, we first normalize the shared columns of factor matrices $(\mathbf{A}_{s(i)}, \mathbf{B}_{s(i)}, \mathbf{C}_{s(i)})$ and $(\mathbf{A}_{s(i+1)}, \mathbf{B}_{s(i+1)}, \mathbf{C}_{s(i+1)})$ to unit norm $||.||_1$. Next, for each column of $(\mathbf{A}_{s(i+1)}, \mathbf{B}_{s(i+1)}, \mathbf{C}_{s(i+1)})$, we find the most similar column of $(\mathbf{A}_{s(i)}, \mathbf{B}_{s(i)}, \mathbf{C}_{s(i)})$, and store the correspondence. Finally, we can describe factor matrices as

$$\widetilde{\mathbf{A}}_\mathbf{s} = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_p^T \end{bmatrix} * \widetilde{\mathbf{A}}\Pi\lambda^{(1/N)}, \ \widetilde{\mathbf{B}}_\mathbf{s} = \begin{bmatrix} b_1^T \\ b_2^T \\ \vdots \\ b_p^T \end{bmatrix} * \widetilde{\mathbf{B}}\Pi\lambda^{(1/N)}, \ \widetilde{\mathbf{C}}_\mathbf{s} = \begin{bmatrix} c_1^T \\ c_2^T \\ \vdots \\ c_p^T \end{bmatrix} * \widetilde{\mathbf{C}}\Pi\lambda^{(1/N)} \tag{5}$$

where $\widetilde{\mathbf{A}}_s, \widetilde{\mathbf{B}}_s$, and $\widetilde{\mathbf{C}}_s$ are matrices of dimension $\widetilde{\mathbf{A}}_s \in \mathbb{R}^{pQ\times R}$, $\widetilde{\mathbf{B}}_s \in \mathbb{R}^{pQ\times R}$ and $\widetilde{\mathbf{C}}_s \in \mathbb{R}^{pQ\times R}$ respectively and N is number of dimensions of tensor. For 3-mode tensor, $N = 3$ and for 4-mode tensor, $N = 4$ and so on. Even though for 3-mode tensor, $\mathbf{A}$ and $\mathbf{B}$ do not increase their number of rows over time, the incoming slices may contribute valuable new estimates to the already estimated factors.Thus, we update all factor matrices in the same way. This is able to partially achieve Goal **G2**.

3.3. Update results

Final step is to remove all the singleton dimensions from the sets of compression matrices $\{\mathbf{U}_p, \mathbf{V}_p, \mathbf{W}_p\}$ and stack them together. Now, we can write compression matrices as

$$\widetilde{\mathbf{P}}_a = \begin{bmatrix} \mathbf{U}(:,:,\mathbf{1})^\mathbf{T} \\ \mathbf{U}(:,:,\mathbf{2})^\mathbf{T} \\ \vdots \\ \mathbf{U}(:,:,\mathbf{p})^\mathbf{T} \end{bmatrix}, \ \widetilde{\mathbf{P}}_b = \begin{bmatrix} \mathbf{V}(:,:,\mathbf{1})^\mathbf{T} \\ \mathbf{V}(:,:,\mathbf{2})^\mathbf{T} \\ \vdots \\ \mathbf{V}(:,:,\mathbf{p})^\mathbf{T} \end{bmatrix}, \ \widetilde{\mathbf{P}}_c = \begin{bmatrix} \mathbf{W}(:,:,\mathbf{1})^\mathbf{T} \\ \mathbf{W}(:,:,\mathbf{2})^\mathbf{T} \\ \vdots \\ \mathbf{W}(:,:,\mathbf{p})^\mathbf{T} \end{bmatrix} \tag{6}$$

where $\widetilde{\mathbf{P}}_a, \widetilde{\mathbf{P}}_b$, and $\widetilde{\mathbf{P}}_c$ are matrices of dimension $\widetilde{\mathbf{P}}_a \in \mathbb{R}^{pQ\times I}$, $\widetilde{\mathbf{P}}_b \in \mathbb{R}^{pQ\times J}$ and $\widetilde{\mathbf{P}}_c \in \mathbb{R}^{pQ\times K}$ respectively. The updated factor matrices ($\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$) for 3-mode tensor $\underline{\mathbf{X}}$ (i.e. $\underline{\mathbf{X}}_{old} + \underline{\mathbf{X}}_{new}$) can be obtained by :

$$\mathbf{A} = \widetilde{\mathbf{P}}_a^{-1} * \widetilde{\mathbf{A}}_s, \ \ \mathbf{B} = \widetilde{\mathbf{P}}_b^{-1} * \widetilde{\mathbf{B}}_s, \ \ \mathbf{C} = \left[\mathbf{C}_{old}; \widetilde{\mathbf{P}}_c^{-1} * \widetilde{\mathbf{C}}_s\right] \tag{7}$$

where $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ are matrices of dimension $\mathbb{R}^{I\times R}$, $\mathbb{R}^{J\times R}$ and $\mathbb{R}^{K_{1\ldots n,(n+1)\ldots m}\times R}$ respectively. Hence, we achieve Goal **G2**.

The time and space complexity of OCTENis $O(p^2QI + p^2QIR + QSt_{new})$ and $pQ(pT + t_{new} + R) + (T + t_{old})R + Q^3$ respectively.

4. EMPIRICAL ANALYSIS

4.1. Experimental Setup

Baselines: In this experiment, four baselines namely **OnlineCP**[18], **SambaTen**[5], **RLST**[9] and **ParaComp**[13] have been selected as the competitors to evaluate the performance.

Evaluation Measures: We evaluate OCTEN and the baselines using three quantitative criteria: Fitness(%), Processor Memory used (MBytes), and CPU-Time (in seconds).

The specifications of each synthetic dataset are given in Table 1. For real datasets, we use AUTOTEN [10] to find rank of tensor.

Table 1: Table of Datasets analyzed.

I=J=K	NNZ	BATCH SIZE	P	Q	SHARED	NOISE (μ, σ)
50	$125K$	5	20	30	5	(0.1, 0.2)
100	$1M$	10	30	35	10	(0.2, 0.2)
500	$125M$	50	40	30	6	(0.5, 0.6)
1000	$1B$	20	50	40	10	(0.4, 0.2)
5000	$7B$	10	90	70	25	(0.5, 0.6)
10000	$1T$	10	110	100	20	(0.2, 0.7)
50000	$6.25T$	4	140	150	30	(0.6, 0.9)

4.2. Results

4.2.1. *Memory efficient, Fast and Accurate*

For all datasets we compute Fitness(%), CPU time (seconds) and Memory(MB) space required. For OCTEN, OnlineCP, ParaComp, Sambaten and RLST we use 10% of the time-stamp data in each dataset as existing old tensor data. The results for qualitative measure for data is shown in Figure 2. All state-of-art methods address the issue very well. Compared with OnlineCP, ParaComp, Sambaten and RLST, OCTEN give comparable fitness and reduce the mean CPU running time by up to 2x times for big tensor data. For all datasets, PARACOMP's accuracy (fitness) is better than all methods. But it is able to handle upto $\underline{\mathbf{X}} \in \mathbb{R}^{10^4\times 10^4\times 250}$ size only. For small size datasets, OnlineCP's efficiency is better than all methods. For large size dataset, OCTEN outperforms the baseline methods w.r.t fitness, average running time (improved 2x-4x) and memory required to process the updates. It significantly saved 40-200% of memory as compared to Online CP, Sambaten and RLST as shown in Figure 2. It saved 40-80% memory space compared to Paracomp. Hence, OCTEN is comparable to state-of-art methods for small dataset and outperformed them for large dataset.

4.2.2. *Scalability Evaluation*

To evaluate the scalability of our method, firstly, a tensor $\underline{\mathbf{X}}$ of small slice size ($I \in [20, 50, 100]$) but longer time dimension ($K \in [10^2 - 10^6]$) is created. Its first $\leq$10% timestamps of data is used for $\underline{\mathbf{X}}_{old}$ and each method's running time for processing batch of $\leq$10 data slices at each time stamp measured. As can be seen from Figure 2(d, e), increasing length of temporal mode increases time consumption quasi-linearly. However the slope is different for various non-temporal mode data sizes. In terms of memory consumption, OCTEN also behaves linearly. This is expected behaviour because with increase in temporal mode, the parameters i.e. p and Q also grows. Once again, our proposed method illustrates that it can efficiently process large sized temporal data.

4.2.3. *Parameter Sensitivity*

We extensively evaluate sensitivity of number of compressed cubes required, size of compressed cubes and number of shared columns required for OCTEN. we fixed batch size to $\approx 0.1 * K$ for all datasets, where K is time dimension of tensor. The results are shown in figure 3. We observe thatfor identifiability 'p' must satisfy the condition, $p \geq \max([\frac{(I-shared)}{(Q-shared)} \ \ \frac{J}{Q} \ \ \frac{K}{Q}])$, to achieve better fitness, lower CPU Time (seconds) and low memory space (MB).

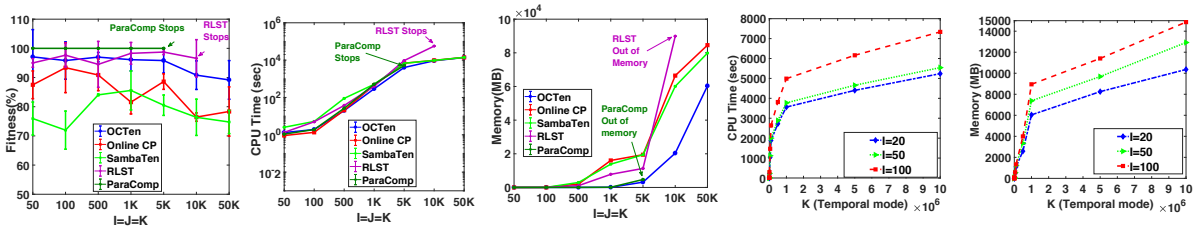

Fig. 2: (a,b) Speed and accuracy of approximation of incoming slices. OCTEN gives comparable accuracy and speed to baseline.(c) Memory required to process the incoming slices. The OCTEN remarkably save the memory as compared to baseline methods. (d,e) CPU time (in seconds) and Memory (MB) used for processing slices to tensor $\underline{\mathrm{X}}$ incrementing in its time mode.

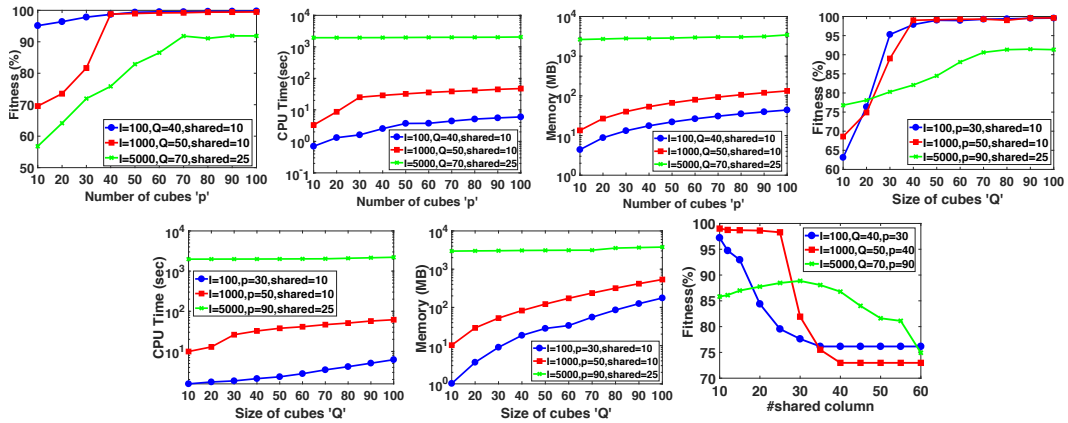

Fig. 3: (a, b, c) OCTEN average Fitness, CPU time and Memory used vs. Number of compressed tensors 'p' on different datasets. With large 'p', high fitness is achieved. (d, e, f) OCTEN average Fitness, CPU Time and memory used vs. compressed tensors size 'Q'. (g) OCTEN average fitness vs. shared columns of compressed tensors '$shared$' on different datasets.

4.2.4. *Effectiveness on real world dataset*

In order to truly evaluate the effectiveness of OCTEN, we test its performance against five sparse (density $\approx 10^{-5}$) real datasets present in Table 2. American College Football Network (ACFN) [115 x 115 x 10k] dataset includes interaction between players of Division IA colleges games during Fall 2000; Foursquare-NYC [1k x 40k x 310] dataset contains 10 months check-in data in New York city collected from Foursquare; Facebook Links [63k x 63k x 650] dataset contains a list of all of the user-to-user links from the Facebook New Orleans sub-network; NELL [12k x 9k x 28k] dataset is an entity-relation-entity tuple snapshot of the Never Ending Language Learner knowledge base and NIPS Publications [2.5k x 2.8k x 14k] dataset consists of papers published from 1987 to 2003 in NIPS. The OCTEN's performance in terms of timing and memory utilization on datasets are summarized in Table 2 and 3. OCTEN outperforms other baseline methods in most of the real dataset. In the case of Foursquare-NYC, Facebook-links, NELL and NIPS dataset, OCTEN gives better results compared to the baselines, specifically in terms of memory used (*better up to average 50-70 times*) and CPU time (*better up to average 5-8%*). OCTEN outperforms for ACFN dataset in terms of memory usage and CPU time is comparable to other methods. Due to high dimensions of dataset, RSLT and PARACOMP are unable to execute within 12 hours for Facebook-links and NELL datasets. OCTEN took advantage of parallel compression and decomposition of incoming slices and save huge amount of memory requirements along with giving comparable run time.

Dataset	OnlineCP	SambaTen	RLST	ParComp	OCTEN
ACFN [12]	**12.91**	16.45	43.52	20.23	14.48
Foursquare-NYC [17]	712.21	746.20	761.42	1.2k	**642.37**
Facebook Links [15]	n/a	4.7k	n/a	n/a	**3.9k**
NELL [3]	42k	37k	n/a	n/a	**35k**
NIPS Publications [4]	372.03	343.98	1.6k	448.63	**315.47**

Table 2: Average CPU Time (sec) over all the batches. The lower the better. The best performance is shown in bold.

Dataset	OnlineCP	SambaTen	RLST	ParComp	OCTEN
ACFN	1.47	1.21	1.11	2.06	**0.02**
Foursquare-NYC	2.36	2.18	2.13	4.34	**0.34**
Facebook Links	n/a	45.49	n/a	n/a	**12.78**
NELL	17.11	16.61	n/a	n/a	**9.43**
NIPS Publications	3.18	2.08	3.86	6.7	**0.45**

Table 3: Average Memory (GB) usage over all the batches. The lower the better. The best saving performance is shown in bold.

5. CONCLUSIONS

We proposed OCTEN a novel compression-based framework for online CP/PARAFAC decomposition for general high-order tensors, with identifiability guarantees. OCTEN approximately preserves order of component of the underlying CP/PARAFAC decomposition in the presence of streaming updates. We experimentally validate OCTEN's effectiveness and accuracy, and we demonstrate its memory efficiency, outperforming state-of-the-art approaches (by 40-200 %). Future work will focus on investigating different tensor decomposition methods and incorporating various such tensor mining methods into our framework.

6. REFERENCES

[1] Woody Austin, Grey Ballard, and Tamara G Kolda. Parallel tensor compression for large-scale scientific

data. In *PDPS, 2016 IEEE Int.*, pages 912–922. IEEE, 2016.

[2] John Brewer. Kronecker products and matrix calculus in system theory. *IEEE Transactions on circuits and systems*, 25(9):772–781, 1978.

[3] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3, 2010.

[4] 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.

[5] Ekta Gujral, Ravdeep Pasricha, and Evangelos E Papalexakis. Sambaten: Sampling-based batch incremental tensor decomposition. pages 387–395, 2018.

[6] Richard A Harshman. Determination and proof of minimum uniqueness conditions for parafac1. *UCLA Working Papers in phonetics*, 22(111-117):3, 1972.

[7] Tao Jiang and Nikos D Sidiropoulos. Kruskal's permutation lemma and the identification of candecomp/parafac and bilinear models with constant modulus constraints. *IEEE Transactions on Signal Processing*, 52(9):2625–2636, 2004.

[8] Tamara G Kolda and Jimeng Sun. Scalable tensor decompositions for multi-aspect data mining. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 363–372. IEEE, 2008.

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

[10] Evangelos E Papalexakis. Automatic unsupervised tensor mining with quality assessment. In *SDM*. SIAM, 2016.

[11] Evangelos E Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. Tensors for data mining and data fusion: Models, applications, and scalable algorithms. *TIST*, 2016.

[12] Fatemeh Sheikholeslami, Brian Baingana, Georgios B. Giannakis, and Nikolaos D. Sidiropoulos. Egonet tensor decomposition for community identification. In *GlobalSIP*, 2016.

[13] N Sidiropoulos, Evangelos E. Papalexakis, and C Faloutsos. Parallel randomly compressed cubes. *IEEE Signal Processing Magazine*, 2014.

[14] Jimeng Sun, Dacheng Tao, Spiros Papadimitriou, Philip S. Yu, and Christos Faloutsos. Incremental tensor analysis:theory and applications. *ACM Trans. Knowl. Discov. Data*, October 2008.

[15] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna 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.

[16] Bo Yang, Ahmed Zamzam, and Nicholas D Sidiropoulos. Parasketch: Parallel tensor factorization via sketching. In *SIAM Int. Conf. on Data Mining*, pages 396–404. SIAM, 2018.

[17] Dingqi Yang, Daqing Zhang, Vincent. W. Zheng, and Zhiyong Yu. Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(1):129–142, 2015.

[18] Shuo Zhou, Nguyen Vinh, James Bailey, Yunzhe Jia, and Ian Davidson. Accelerating online cp decompositions for higher order tensors. In *22nd ACM SIGKDD*. ACM, 2016.