Streaming Tensor Factorization for Infinite Data Sources

Shaden Smith[*] [†] shaden.smith@intel.com
Kejun Huang[*] [‡] huang663@umn.edu
Nicholas D. Sidiropoulos[§] nikos@virginia.edu
George Karypis[‡] karypis@cs.umn.edu

Abstract

Sparse tensor factorization is a popular tool in multi-way data analysis and is used in applications such as cybersecurity, recommender systems, and social network analysis. In many of these applications, the tensor is not known *a priori* and instead arrives in a streaming fashion for a potentially unbounded amount of time. Existing approaches for streaming sparse tensors are not practical for unbounded streaming because they rely on maintaining the full factorization of the data, which grows linearly with time. In this work, we present CP-stream, an algorithm for streaming factorization in the model of the canonical polyadic decomposition which does not grow linearly in time or space, and is thus practical for long-term streaming. Additionally, CP-stream incorporates user-specified constraints such as non-negativity which aid in the stability and interpretability of the factorization. An evaluation of CP-stream demonstrates that it converges faster than state-of-the-art streaming algorithms while achieving lower reconstruction error by an order of magnitude. We also evaluate it on real-world sparse datasets and demonstrate its usability in both network traffic analysis and discussion tracking. Our evaluation uses exclusively public datasets and our source code is released to the public as part of SPLATT, an open source high-performance tensor factorization toolkit.

1 Introduction

Tensors are the natural extension of matrices to multi-way data. Tensor *factorization* is a technique for analyzing multi-way data and recently has had major success in applications spanning signal processing and machine learning [19]. Within these fields, the data of interest frequently has a temporal component that can be exploited to gain additional insights. For example, discussion tracking [1], cybersecurity [6, 8] and social network analysis [14] have all benefited from incorporating the temporal dimension into the tensor factorization. The ability to effectively and efficiently process and analyze these temporal datasets is thus of great interest to practitioners and researchers alike.

These temporal applications bring an additional challenge to tensor factorization: the complete tensor is not available *a priori*, and instead batches of data arrive in a streaming fashion. Consider for example the cybersecurity setting in which network activity is modeled as a tensor with modes such as *time*, *IP addresses*, and *network ports*. In order to be useful as a security tool, tensor factorization should operate in an online manner and process incoming data in real time. Critically, each batch of data should not be treated as an independent tensor to analyze, but instead a snapshot of an unbounded stream of related data which may change over time. Thus, the streaming factorization should ensure that the underlying model is tracked from batch to batch, while still remaining flexible enough to allow new trends to emerge.

In order to be practical for real-world applications, a streaming factorization algorithm must at least satisfy the following requirements. First, a streaming algorithm must be less costly than simply recomputing a new factorization each time a new batch of data arrives. Therefore, the cost of incorporating a new batch of data should not depend on the number of previous batches. Second, the algorithm must be resilient to sparse and noisy incoming data. State-of-the-art solutions often satisfy either the efficiency *or* the quality constraints, but are not able to satisfy both.

To address these challenges, we present CP-stream, an algorithm that can effectively model a streaming tensor while having low-cost updates in terms of both space and time. This is accomplished by maintaining historical tensor data in a factored form that does not increase in size as time progresses. Moreso, it utilizes the recently-proposed combination of alternating updates with the alternating direction method of multipliers (AO-ADMM) [10] to incorporate constraints that improve stability and interpretability while maintaining a low computational cost. Our contributions include:

1. CP-stream, a streaming algorithm that updates a factorization with space and time complexities that are constant in terms of the number of previous timesteps, and is thus suitable for long-term execution.
2. Extensions to CP-stream including non-negativity and sparsity constraints, and optimizations for sparse data.
3. An experimental evaluation that demonstrates success on both dense and sparse data and a demonstration of event discoveries on sparse, real-world, and public datasets.

[*] Authors contributed equally to this work.
[†] Intel Parallel Computing Lab
[‡] University of Minnesota
[§] University of Virginia

4. An open source implementation of CP-stream that builds upon existing high-performance tensor and matrix kernels, resulting in an implementation that is parallel and suitable for large-scale data analysis.

The rest of the paper is organized as follows. Section 2 provides an overview of tensors and tensor factorization. Section 3 reviews related work on streaming tensor factorizations. Section 4 formulates CP-stream and provides an analysis of its complexity. Section 5 addresses extensions to CP-stream including non-negativity constraints and considerations for sparse data. Section 6 evaluates our method on a variety of synthetic and real-world datasets. Lastly, Section 7 offers concluding remarks.

2 Background and Notation

In this section, we define the notation used throughout the paper and also provide a brief overview of tensor factorization. For additional information on tensors and their factorizations, we direct the reader to the surveys by Kolda and Bader [13] and Sidiropoulos et al. [19].

2.1 Notation We denote vectors with bold lowercase letters ($\boldsymbol{s}$), matrices with bold capital letters ($\boldsymbol{A}$), and tensors with bold calligraphic letters ($\boldsymbol{\mathcal{X}}$). We say that a tensor has N modes of lengths $I_1, \ldots, I_N$. We denote the Frobenius norm of a vector, matrix, or tensor as $\|\cdot\|$. A tensor can be unfolded, or *matricized*, along any of its modes into a matrix. The tensor unfolding along the nth mode is denoted $\boldsymbol{X}_{(n)} \in \mathbb{R}^{I_n \times I_1, \ldots, I_{n-1}, I_{n+1}, \ldots, I_N}$. More simply, the nth mode of the tensor forms the rows of the matrix and the remaining modes form the columns.

There are two essential matrix products when discussing tensor factorization. The first is the Hadamard product, denoted $\boldsymbol{A} \circledast \boldsymbol{B}$, which is simply the elementwise product. The second operation is the Khatri-Rao product, denoted $\boldsymbol{A} \odot \boldsymbol{B}$, which is the columnwise Kronecker product:

$$\boldsymbol{A} \odot \boldsymbol{B} = \left[\boldsymbol{a}_1 \otimes \boldsymbol{b}_1, \ldots, \boldsymbol{a}_J \otimes \boldsymbol{b}_J\right],$$

where $\boldsymbol{A} \in \mathbb{R}^{P \times J}$, $\boldsymbol{B} \in \mathbb{R}^{Q \times J}$, and $(\boldsymbol{A} \odot \boldsymbol{B}) \in \mathbb{R}^{PQ \times J}$.

2.2 Canonical Polyadic Decomposition (CPD) The CPD is perhaps the most popular tensor decomposition, and the one that is the focus of this work. Shown in Figure 1, the CPD models a tensor as the summation of outer products. The number of outer products is the *rank* of the factorization, and is denoted K. Applications in signal processing and machine learning are almost always interested in *low-rank* factorizations in which K is chosen to be a small integer usually on the order of 10 or 100. The low-rank CPD can be seen as one higher-order interpretation of the truncated SVD.

The vectors that form the outer products in the CPD are collected into N factor matrices: $\boldsymbol{A}^{(1)} \in \mathbb{R}^{I_1 \times K}, \ldots, \boldsymbol{A}^{(N)} \in \mathbb{R}^{I_N \times K}$. The outer product formulation is thus written $\sum_{k=1}^{K} \boldsymbol{a}_k^{(1)} \circ \cdots \circ \boldsymbol{a}_k^{(N)}$, or abbreviated $[\![\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}]\!]$. Lastly, the CPD can be written in terms of an unfolding:

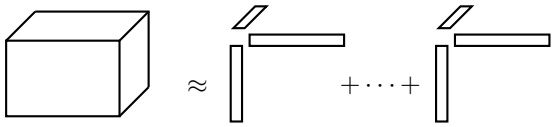

Figure 1: The canonical polyadic decomposition (CPD) models a tensor as the summation of outer products.

$$\underset{\{\boldsymbol{A}^{(n)}\}}{\text{minimize}} \quad \frac{1}{2} \left\| \boldsymbol{X}_{(n)} - \boldsymbol{A}^{(n)} \left(\bigodot_{\nu \neq n} \boldsymbol{A}^{(\nu)} \right)^{\top} \right\|^2 .$$

A remarkable trait of the CPD, and a major reason for its success, is that the CPD usually unique up to a permutation of the outer products and a scaling ambiguity in the vectors of each outer product [19]. When one mode of the tensor is much larger than the others, which suits well to the streaming tensor case we consider here, recent results have shown that the CPD is essentially unique almost surely even when K is as large as $(I_1 - 1)(I_2 - 1)$ for 3-way tensors [7].

3 Related Work

Streaming tensor factorization can be viewed as an extension to streaming matrix factorization, with applications in subspace tracking [27, 2] and online dictionary learning [15]. Computing the CPD in a streaming fashion was first studied in signal processing [17], to the best of our knowledge. Due to the size of the problems considered there, the algorithms were not designed for memory efficiency. A memory-efficient online algorithm called Online-SGD was proposed by Mardani et al. [16], where a stochastic gradient descent (SGD) update is performed for all the factors except the one corresponding to the ever-growing dimension. The drawback for Online-SGD is the need to tune the learning rate for the SGD step, which turn out to be a non-trivial task in practice. Several approaches have been developed which avoid the need to tune a learning rate [11, 28].

The Tucker model is another popular model for tensor factorization that has been studied in the streaming setting [24, 3]. However, for reasons including model simplicity and the uniqueness properties of the CPD, we are more in favor of the streaming CPD model.

Another conceptually related line of work is on stochastic algorithms for tensor decompositions [18, 26, 4], which deals with large-scale data as well, but the problem dimension is predetermined. The focus of this work is in the case when one dimension of the tensor is growing in a potentially unbounded manner. The two ideas of stochastic sampling and streaming CPD were recently integrated in SamBaTen [9].

Table 1: Summary of algorithms for streaming CPD.

	Memory	Higher-order?	Constraints?	Tuning-free?
PARAFAC-SDT [17]	$O(TI_1I_2)$			✓
PARAFAC-RLST [17]	$O(KI_1I_2)$			✓
Online-SGD [16]	$O(K(I_1+I_2))$			
OLSTEC [11]	$O(K(I_1+I_2))$			
Online-CP [28]	$O(K\sum I_n)$	✓		✓
CP-stream (proposed)	$O(K\sum I_n)$	✓	✓	✓

Memory is the required memory for computing a batch update, excluding the latest tensor itself. Memory overheads which are not feasible for unbounded streams or sparse datasets are colored red. **Higher-order** indicates that the algorithm has been extended to tensors with more than three modes. **Constraints** indicates that the algorithm supports constraints on the factorization, including non-negativity. **Tuning-free** indicates that the algorithm either does not require tuning of hyper-parameters, or is not overly sensitive to their values.

We compare the state-of-the-art streaming CPD algorithms to the proposed algorithm in Table 1.

4 Streaming Tensor Factorization

We now detail our streaming CPD algorithm, CP-stream. We begin with a formalization of our streaming setting in Section 4.1 and then detail our algorithm in Section 4.2.

4.1 Problem Setting We consider the problem of finding the rank-K CPD of an $N+1$-way tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times \cdots \times I_N \times T}$, in which N-way subtensors arrive over T batches (T is potentially unbounded). We arrive at the following optimization problem

(4.1)
$$\underset{\{\boldsymbol{A}^{(n)} \in \mathbb{R}^{I_n \times K}\}, \boldsymbol{S} \in \mathbb{R}^{T \times K}}{\text{minimize}} \quad \frac{1}{2} \left\| \boldsymbol{\mathcal{Y}} - [\![\boldsymbol{A}^{(1)}, ..., \boldsymbol{A}^{(N)}, \boldsymbol{S}]\!] \right\|^2,$$

where $\{\boldsymbol{A}^{(n)}\}$ are the factor matrices modeling the common N modes and $\boldsymbol{S}$ incorporates temporal information.

We can equivalently model $\boldsymbol{\mathcal{Y}}$ as a sequence of N-way tensors $\boldsymbol{\mathcal{X}}_1, \ldots, \boldsymbol{\mathcal{X}}_T \in \mathbb{R}^{I_1 \times \cdots \times I_N}$. Tensor $\boldsymbol{\mathcal{X}}_t$ is modeled with $[\![\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}; \boldsymbol{s}_t]\!]$, where $\boldsymbol{A}^{(n)} \in \mathbb{R}^{I_n \times K}$ and $\boldsymbol{s}_t \in \mathbb{R}^K$. Together, the sequence forms the following optimization problem

$$\underset{\{\boldsymbol{A}^{(n)} \in \mathbb{R}^{I_n \times K}\}, \{\boldsymbol{s}_t \in \mathbb{R}^K\}}{\text{minimize}} \quad \sum_{t=1}^{T} \frac{1}{2} \left\| \boldsymbol{\mathcal{X}}_t - [\![\{\boldsymbol{A}^{(n)}\}; \boldsymbol{s}_t]\!] \right\|^2.$$

Our goal is to design an algorithm to find a good approximate solution of (4.1) by considering only one tensor sample $\boldsymbol{\mathcal{X}}_t$ at a time.

When we are specifically interested in emphasizing recent data over long-ago historical data in the model (e.g., subspace tracking), an exponential decay can be applied to the various $\boldsymbol{\mathcal{X}}$ tensors. This is also useful for modeling real-world data in which the joint rank-K CPD model may change over time, and we wish to "forget" some of the CPD in order to better model the new data.

There is a scaling ambiguity inherent in the CPD problem (4.1): if each factor $\boldsymbol{A}^{(n)}$ is multiplied by a diagonal matrix $\boldsymbol{D}_n$ on the right, and factor $\boldsymbol{S}$ by diagonal matrix $\boldsymbol{D}_S$, as long as these diagonal matrices satisfies that $\boldsymbol{D}_1 ... \boldsymbol{D}_N \boldsymbol{D}_S = \boldsymbol{I}$, this change of variable does not affect the quality of the factorization. Therefore, without loss of generality, we propose to slightly modify problem (4.1) as follows

(4.2)
$$\underset{\{\boldsymbol{A}^{(n)} \in \mathcal{C}\}, \boldsymbol{S}}{\text{minimize}} \quad \frac{1}{2} \left\| \boldsymbol{\mathcal{Y}} - [\![\boldsymbol{A}^{(1)}, ..., \boldsymbol{A}^{(N)}, \boldsymbol{S}]\!] \right\|^2 + \frac{\lambda}{2} \|\boldsymbol{S}\|^2,$$

where $\boldsymbol{A}^{(n)} \in \mathcal{C}$ denotes the constraint that the norm of each column of $\boldsymbol{A}^{(n)}$ should be less than or equal to one. With the help of the regularization term in (4.2), the norms of all columns of $\boldsymbol{A}^{(n)}$ will be equal to one—otherwise we can scale up the unattained columns in $\boldsymbol{A}^{(n)}$ without violating the constraints, while scaling down the $(\lambda/2)\|\boldsymbol{S}\|^2$ term; thus obtaining a smaller loss value for (4.2). As long as we choose a relatively small λ, the solution of (4.2) will be very close to the original problem (4.1). As we will demonstrate in Section 6, this modified formulation helps significantly in the convergence of the algorithm, although mathematically it makes a minor difference.

4.2 Proposed Algorithm The high-level idea of CP-stream is as follows: at time t, we receive tensor $\boldsymbol{\mathcal{X}}_t$ and have the previous estimates $\{\boldsymbol{A}_{t-1}^{(n)}\}$ and a sufficient statistic for $\boldsymbol{s}_1, ..., \boldsymbol{s}_{t-1}$. First, we compute $\boldsymbol{s}_t$ as the solution of the following optimization problem

$$\underset{\boldsymbol{s}_t}{\text{minimize}} \quad \frac{1}{2} \left\| \boldsymbol{\mathcal{X}}_t - [\![\{\boldsymbol{A}_{t-1}^{(n)}\}; \boldsymbol{s}_t]\!] \right\|^2 + \frac{\lambda}{2} \|\boldsymbol{s}_t\|^2.$$

As we can see, $\boldsymbol{s}_t$ only depends on the current data sample $\boldsymbol{\mathcal{X}}_t$ and the previous estimates $\{\boldsymbol{A}_{t-1}^{(n)}\}$, and in this case we have a closed-form update

(4.3)
$$\boldsymbol{s}_t \leftarrow \left(\circledast_{n=1}^{N} \boldsymbol{A}_{t-1}^{(n)\top} \boldsymbol{A}_{t-1}^{(n)} + \lambda \boldsymbol{I} \right)^{-1} \left(\odot_{n=1}^{N} \boldsymbol{A}_{t-1}^{(n)} \right)^{\top} \text{vec}(\boldsymbol{\mathcal{X}}_t).$$

Note that the final matrix-vector multiplication can be efficiently carried out using existing high-performance software

for computing the *matricized tensor times Khatri-Rao product* (MTTKRP), e.g. SPLATT [22, 23].

After obtaining $\boldsymbol{s}_t$, we keep it fixed and update the $\{\boldsymbol{A}^{(n)}\}$ factors. Ideally, we would want to update them according to all the historical data $\boldsymbol{\mathcal{X}}_1, \ldots, \boldsymbol{\mathcal{X}}_t$; however, since in the streaming setting, the number of tensors can grow very large, this is not feasible. We therefore follow Vandecappelle et al. [25] and minimize an approximate loss by replacing $\boldsymbol{\mathcal{X}}_1, \ldots, \boldsymbol{\mathcal{X}}_{t-1}$ with the existing factorization $[\![\{\boldsymbol{A}^{(n)}_{t-1}\}; \boldsymbol{S}_{t-1}]\!]$, where $\boldsymbol{S}_{t-1} \in \mathbb{R}^{t-1 \times K}$ is the matrix with rows $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{t-1}$. We arrive at the following optimization problem:

$$\underset{\{\boldsymbol{A}^{(n)} \in \mathcal{C}\}}{\text{minimize}} \quad \frac{1}{2} \left\| \boldsymbol{\mathcal{X}}_t - [\![\{\boldsymbol{A}^{(n)}\}; \boldsymbol{s}_t]\!] \right\|^2 + \sum_{i=1}^{t-1} \frac{\mu^{t-i}}{2} \left\| [\![\{\boldsymbol{A}^{(n)}_{t-1}\}; \boldsymbol{s}_i]\!] - [\![\{\boldsymbol{A}^{(n)}\}; \boldsymbol{s}_t]\!] \right\|^2,$$

where $\mu \in [0, 1]$ is a forgetting factor to down-weight the importance of fitting the data that was observed long ago. However, the nonlinear least squares update of Vandecappelle et al. requires access to all of the prior $\{\boldsymbol{s}_i\}$, and is thus infeasible for the long-term streaming setting that we consider. Instead, we adopt an alternating update rule and see that the optimization with respect to $\boldsymbol{A}^{(n)}$ is equivalent to

$$\underset{\boldsymbol{A}^{(n)} \in \mathcal{C}}{\text{minimize}} \quad \frac{1}{2} \text{tr}\left(\boldsymbol{A}^{(n)} \boldsymbol{\Phi}^{(n)} \boldsymbol{A}^{(n)\top} \right) - \text{tr}\left(\boldsymbol{\Psi}^{(n)\top} \boldsymbol{A}^{(n)} \right), \tag{4.4}$$

where

$$\boldsymbol{\Phi}^{(n)} = \left(\circledast_{\nu \neq n} \boldsymbol{A}^{(\nu)\top} \boldsymbol{A}^{(\nu)} \right) \circledast \left(\mu \boldsymbol{G}_{t-1} + \boldsymbol{s}_t \boldsymbol{s}_t^\top \right), \tag{4.5}$$

$$\boldsymbol{\Psi}^{(n)} = \left(\bigodot_{\nu \neq n}^{N} \boldsymbol{A}^{(\nu)} \right)^\top \text{vec}(\boldsymbol{\mathcal{X}}_t) + \boldsymbol{A}^{(n)} \left(\left(\circledast_{\nu \neq n} \boldsymbol{A}^{(\nu)\top}_{t-1} \boldsymbol{A}^{(\nu)} \right) \circledast \mu \boldsymbol{G}_{t-1} \right), \tag{4.6}$$

and

$$\boldsymbol{G}_{t-1} = \sum_{i=1}^{t-1} \mu^{t-i} \boldsymbol{s}_i \boldsymbol{s}_i^\top.$$

This formulation closely resembles the alternating least squares formulation for computing the traditional (i.e., non-streamed) CPD [13, 19]. Intuitively, $\boldsymbol{\Phi}^{(n)} \in \mathbb{R}^{K \times K}$ is a matrix of normal equations that includes the updated factors and a down-weighted matrix of temporal information, $\boldsymbol{G}_{t-1} \in \mathbb{R}^{K \times K}$. Similarly, $\boldsymbol{\Psi}^{(n)} \in \mathbb{R}^{I_n \times K}$ is a matrix of I_n right-hand-sides and its two summed terms incorporate current and historical tensor data, respectively.

Problem (4.4) is a constrained least squares problem that does not have a closed-form solution. However, an approximate solution can be efficiently obtained via the alternating direction method of multipliers (ADMM) [5]. Specifically, for solving problem (4.4), we follow the matrix form updates derived in prior work [10, 20]

$$\begin{cases} \widetilde{\boldsymbol{A}} \leftarrow \left(\boldsymbol{\Psi}^{(n)} + \rho(\boldsymbol{A}^{(n)} + \boldsymbol{U}) \right) \left(\boldsymbol{\Phi}^{(n)} + \rho \boldsymbol{I} \right)^{-1}, \\ \boldsymbol{A}^{(n)} \leftarrow \text{Proj}_{\mathcal{C}} \left[\widetilde{\boldsymbol{A}} - \boldsymbol{U} \right], \\ \boldsymbol{U} \leftarrow \boldsymbol{U} + \widetilde{\boldsymbol{A}} - \boldsymbol{A}^{(n)}, \end{cases} \tag{4.7}$$

where $\widetilde{\boldsymbol{A}}$ and $\boldsymbol{U}$ are auxiliary variables, and the notation $\text{Proj}_{\mathcal{C}}[\cdot]$ denotes the operation of projecting the argument onto the set $\mathcal{C}$. Since (4.4) is a convex problem, in principle any choice of $\rho > 0$ guarantees convergence of (4.7) to an optimal solution of (4.4). It was found empirically that $\rho = \text{tr}(\boldsymbol{\Phi}^{(n)})/K$ results in a fast convergence rate, together with warm start using the previous update, which we adopt here as well [10]. Computationally, the most expensive step is the construction of $\boldsymbol{\Phi}^{(n)}$ and $\boldsymbol{\Psi}^{(n)}$—afterwards, if we cache the Cholesky decomposition of $\boldsymbol{\Phi}^{(n)} + \rho \boldsymbol{I}$, the per-iteration complexity of (4.7) is only $O(I_n K^2)$. For a few number of ADMM iterations, the computation is almost the same as that of a single (unconstrained) least-squares update.

Lastly, after updating all of the factor matrices we update $\boldsymbol{G}_t$ for the next timestep. This can be accomplished recursively:

$$\boldsymbol{G}_t = \mu \boldsymbol{G}_{t-1} + \boldsymbol{s}_t \boldsymbol{s}_t^\top.$$

Note that by adopting an alternating solution, the temporal vectors $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_t$ are compactly stored in $\boldsymbol{G}_t$.

Our proposed algorithm is fleshed out in Algorithm 1.

Algorithm 1 CP-stream

Require: $\boldsymbol{\mathcal{X}}_1, \boldsymbol{\mathcal{X}}_2, ..., \boldsymbol{\mathcal{X}}_T$; forgetting factor μ
initialize $\boldsymbol{A}^{(1)}_0, ..., \boldsymbol{A}^{(N)}_0$
$\boldsymbol{G}_0 \leftarrow \boldsymbol{0}$
for $t = 1, ..., T$ **do**
  $\boldsymbol{s}_t \leftarrow$ least-squares update (4.3)
  repeat
    for $n = 1, ..., N$ **do**
      construct $\boldsymbol{\Phi}^{(n)}$ and $\boldsymbol{\Psi}^{(n)}$ per (4.5) and (4.6)
      $\rho = \text{tr}(\boldsymbol{\Phi}^{(n)})/K$
      $\boldsymbol{A}^{(n)}_t \leftarrow$ ADMM iterates (4.7)
    end for
  until convergence
  $\boldsymbol{G}_t = \mu \boldsymbol{G}_{t-1} + \boldsymbol{s}_t \boldsymbol{s}_t^\top$
end for

5 Extensions of the algorithm

5.1 Imposing structures In many cases, we not only want to factor a stream of tensors into a joint CPD model, but also want to impose structural constraints onto the latent factors $\{\boldsymbol{A}^{(n)}\}$ and/or $\boldsymbol{s}_t$'s. We consider two of the most

popular structure people like to impose: non-negativity and sparsity, via adding an additional ≥ 0 constraint or an ℓ_1-norm regularization term. Since the update of $\boldsymbol{s}_t$ essentially solves an unconstrained least-squares problem, it will be easy to add non-negativity constraint and/or ℓ_1-norm regularization and solve by ADMM instead. The iterate will be similar to (4.7), and details of the derivation and pertinent projection/proximity operators can be found in prior work [10, 20].

Things are more complicated in the case of the $\boldsymbol{A}^{(n)}$ factors. For formulation purposes, we have already imposed the constraint that each column of $\boldsymbol{A}^{(n)}$ should have norm no greater than one. The question is whether non-negativity/ℓ_1-regularization, together with the norm constraint, still leads to a computationally efficient update as in the second line of (4.7)? The answer is yes. However, to the best of our knowledge, it has not been specifically derived in the literature. We therefore provide detailed derivation here.

PROPOSITION 5.1. *The optimal solution for*

$$\underset{\boldsymbol{x}}{\text{minimize}}\ \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{z}\|^2 \ \text{ subject to } \boldsymbol{x}\geq 0, \|\boldsymbol{x}\|^2\leq 1$$

is $\dfrac{(\boldsymbol{z})_+}{\max(1,\|(\boldsymbol{z})_+\|)}$.

Proof. Since it is a convex problem, it suffices to find the $\boldsymbol{x}$ that satisfies the KKT condition. Denote the dual variable for $\boldsymbol{x}\geq 0$ as $\boldsymbol{y}\geq 0$, and that of $\|\boldsymbol{x}\|^2\leq 1$ as $\beta\geq 0$. Taking the derivative of the Lagrangian and setting it equal to zero, we have

$$\boldsymbol{x}=\frac{\boldsymbol{z}+\boldsymbol{y}}{1+\beta}.$$

Since both $\boldsymbol{x}$ and $\boldsymbol{y}$ are non-negative, this implies that

$$\boldsymbol{y}=-(\boldsymbol{z})_-, \quad \text{and} \quad \boldsymbol{x}=\frac{(\boldsymbol{z})_+}{1+\beta}.$$

Furthermore, dual variable β should be chosen so that the constraint $\|\boldsymbol{x}\|^2\leq 1$ is satisfied, resulting in the solution given in Proposition 5.1. **Q.E.D.**

PROPOSITION 5.2. *The optimal solution for*

$$\underset{\boldsymbol{x}}{\text{minimize}}\ \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{z}\|^2+\gamma\|\boldsymbol{x}\|_1 \ \text{ subject to } \|\boldsymbol{x}\|^2\leq 1$$

is $\dfrac{\mathcal{S}_\gamma(\boldsymbol{z})}{\max(1,\|\mathcal{S}_\gamma(\boldsymbol{z})\|)}$, *where* $\mathcal{S}_\gamma(\boldsymbol{z})$ *denotes the soft-thresholding operator.*

Proof. Similar to Proposition 5.1, we prove it by finding the $\boldsymbol{x}$ that satisfies the KKT condition. Since $\gamma\|\boldsymbol{x}\|_1$ is non-differentiable, the optimality condition states that zeros is a sub-gradient of the Lagrangian

$$\boldsymbol{0}\in\boldsymbol{x}-\boldsymbol{z}+\partial\gamma\|\boldsymbol{x}\|_1+\beta\boldsymbol{x}.$$

If $\beta=0$, then it is well-known that $\mathcal{S}_\gamma(\boldsymbol{z})$ satisfy this condition, meaning if $\|\mathcal{S}_\gamma(\boldsymbol{z})\|^2\leq 1$, the solution is simply $\mathcal{S}_\gamma(\boldsymbol{z})$. Otherwise, $\beta>0$, but we can see that if $(1+\beta)\boldsymbol{x}=\mathcal{S}_\gamma(\boldsymbol{z})$, it again satisfies the condition. Again, β should be chosen so that $\|\boldsymbol{x}\|^2\leq 1$ is satisfied as equality, according to complimentary slackness, rendering the solution given in Proposition 5.2. **Q.E.D.**

As we can see, for both non-negativity constraint and ℓ_1-regularization, an additional bound on the norm simply results in a normalization, if the corresponding projection or soft-thresholding has norm greater than one. This is an interesting result because projection onto the intersection of two sets is in general not obtained by two consecutive single projections.

To summarize, if we want to impose non-negativity constraint or ℓ_1-regularization to the latent factor $\boldsymbol{A}^{(n)}$, we can simply replace the second step of (4.7) with Proj$[(\widetilde{\boldsymbol{A}}-\boldsymbol{U})_+]$ or Proj$[\mathcal{S}_\gamma(\widetilde{\boldsymbol{A}}-\boldsymbol{U})]$. The ADMM iterates (4.7) are still guaranteed to find a conditionally optimal update, and the per-iteration complexity remains low.

5.2 Considerations for sparse data Some additional challenges arise when the streamed data is sparse.

The discussion thus far assumes that the lengths of the tensor modes are known ahead of time in order to initialize the problem. However, in a sparse setting many indices will not appear in the data stream until later in the factorization. To address this challenge, we dynamically grow the factor matrices throughout the computation. Whenever a new index is observed in the incoming data, a new row is added to the corresponding factor matrix $\boldsymbol{A}_t^{(n)}$ and initialized randomly. We additionally add a row to the existing factorization $\boldsymbol{A}_{t-1}^{(n)}$, but initialize it with zero in order to signify it not appearing in the data stream previously.

Another challenge in the sparse setting is that the distribution of incoming non-zeros can be highly non-uniform, with some indices not receiving updates for many timesteps. This challenge is not present in the traditional factorization setting, as all indices are available to be updated each iteration of the factorization. As a result, rows which do not observe non-zeros will be updated and converge to zero, losing their significance in the factorization.

This challenge can be addressed in multiple ways, depending on the application and needs of the user. The most simple solution is to allow rows of the factors to receive no updates and ultimately converge to zero. This strategy is sensible in the subspace tracking scenario, when the goal of the streaming factorization is to primarily model recent data. If instead the quality of the factorization over all timesteps is important, then the strategy employed by SamBaTen [9] may instead by preferable: only factor rows which observe non-zeros in the current timestep are updated. This strategy

Table 2: Summary of real-world datasets.

Dataset	NNZ	Dimensions	T	Sparse?
AirportHall [11]	5M	144×176	200	
ChicagoCrime [21]	5M	$24 \times 77 \times 32$	6K	✓
Reddit2008 [21]	66M	$88\text{K} \times 2\text{K} \times 30\text{K}$	367	✓
CyberLANL [12]	165M	$26\text{K} \times 16\text{K} \times 16\text{K} \times 13\text{K}$	1.4K	✓

NNZ is the number of nonzero entries in the dataset. **K** and **M** stand for thousand and million, respectively. **Dimensions** are the dimensions of the streamed data, excluding the streamed mode. $\boldsymbol{T}$ is the length of the temporal mode (i.e., the number of batches).

will prevent rows from converging to zero and better model the data at a global scale. However, it is not applicable in applications when the *absence* of activity is important, such as a cybersecurity scenario. Given the subspace tracking motivation of this work, we elect to always update rows and allow them to converge to zero.

6 Experimental Methodology & Results

6.1 Experimental Setup

Configuration. Results on sparse datasets are acquired using the Cori supercomputer at NERSC. We use the Intel Xeon Phi 7250 ("Knights Landing") many-core processors, each with 68 cores, 16GB of MCDRAM configured as a last-level cache, and 96GB of DDR4 memory. CP-stream is implemented both in Matlab and C++. We use the Matlab version to evaluate on synthetic datasets and to compare against the state-of-the-art algorithms which are also implemented in Matlab. The C++ version of CP-stream is parallelized for multi- and many-core systems and is integrated into the upcoming SPLATT software release.

Datasets. We evaluate our factorization on a set of public, real-world tensor datasets. The tensors are chosen to reflect a diverse set of applications and are summarized in Table 2. AirportHall is surveillance footage distributed with OLSTEC [11], in which each batch is a frame of video. The remaining tensors are sparse and taken from the from the FROSTT collection [21]. ChicagoCrime is a *date-hourofday-neighborhood-crime* tensor representing crime reports in Chicago over a sixteen year time period. Tensor entries are counts. Reddit2008 is a *date-user-community-word* tensor representing all user comments from Reddit[1] from 2008. Tensor entries are word counts. CyberLANL is an *hour-user-machine-machine-domain* tensor of network logins at Los Alamos National Laboratory over a two month period. Tensor entries are a count of login attempts.

6.2 Synthetic Data Evaluations We first evaluate the correctness of CP-stream on a stream of synthetic two-way tensor data samples, which is equivalent to factoring a three-way tensor in the batch manner. Fixing $I_1{=}I_2{=}100$ and $K{=}10$, the ground truth factors $\boldsymbol{A}_\natural^{(1)}$ and $\boldsymbol{A}_\natural^{(2)}$ are generated by first randomly draw each entry from i.i.d. Gaussian distribution $\mathcal{N}(0,1)$. Next, each column of $\boldsymbol{A}_\natural^{(1)}$ and $\boldsymbol{A}_\natural^{(2)}$ are normalized to have norm equal to one, which can be done without loss of generality because of the scaling ambiguity inherent in the CPD model. At each time t, a vector $\boldsymbol{s}_t^\natural \in \mathbb{R}^K$ is generated from $\mathcal{N}(0, \boldsymbol{I})$, and a tensor sample $\boldsymbol{\mathcal{X}}_t$ is constructed as

$$\boldsymbol{\mathcal{X}}_t = [\![\boldsymbol{A}_\natural^{(1)}, \boldsymbol{A}_\natural^{(2)}; \boldsymbol{s}_t^\natural]\!] + \boldsymbol{\mathcal{W}}_t,$$

where $\boldsymbol{\mathcal{W}}$ is a random tensor with each entry generated from i.i.d. $\mathcal{N}(0, \sigma^2)$. Here we choose $\sigma = 10^{-3}$.

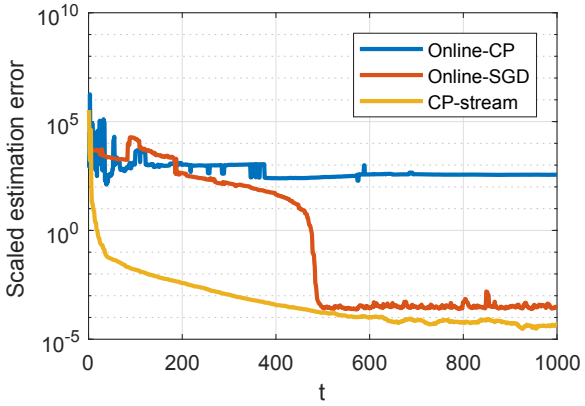

Figure 2: Normalized error in the recovery of the ground-truth latent factors over time. The estimation error is measured via Equation (6.8).

We compare the performance of CP-stream with Online-SGD [16] and Online-CP [28]. The criteria is to compare with an algorithm dealing with the same streaming setting and joint CPD model, *and* has the same property of memory efficiency. For this reason, the adaptive PARAFAC algorithm [17] is not compared, since it requires to keep track of at least a dense matrix of size $I_1 I_2 \times K$, with another version requiring to store all the historical data, which exceeds the memory requirement considered in this paper. Another baseline is OLSTEC [11], which is similar to Online-CP, and thus we omit for brevity. For our method, we choose λ to be a relatively small number $\lambda = 10^{-4}$ so as not to affect the quality of fitting the data, and a forgetting factor $\mu = 0.99$. For Online-SGD, there is a similar parameter λ (for all the factors, since there is no bound constraint on $\{\boldsymbol{A}^{(n)}\}$ as we did), and we choose it to be $\lambda = \sqrt{2MN}\sigma$ as suggested by the authors. All methods are initialized at the same random factors $\boldsymbol{A}_0^{(1)}$ and $\boldsymbol{A}_0^{(2)}$.

Figure 2 shows the progress of normalized estimation error of the latent factors after resolving the permutation ambiguity of the columns, defined as

$$\frac{\|\boldsymbol{A}_\natural^{(1)} - \boldsymbol{A}_t^{(1)}\|^2}{\|\boldsymbol{A}_\natural^{(1)}\|^2} + \frac{\|\boldsymbol{A}_\natural^{(2)} - \boldsymbol{A}_t^{(2)}\|^2}{\|\boldsymbol{A}_\natural^{(2)}\|^2}. \tag{6.8}$$

This is a meaningful evaluation criterion due to the generic

[1] https://reddit.com/

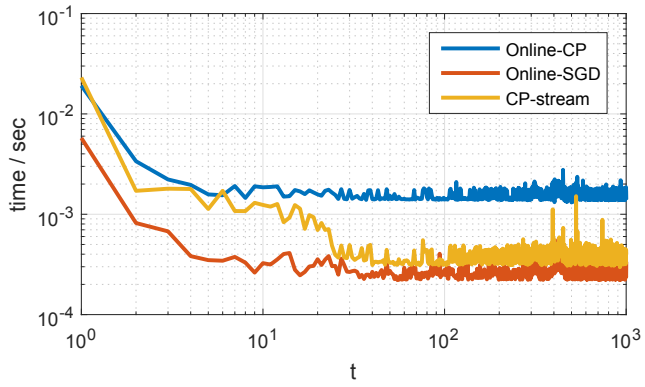

Figure 3: The amount of time in seconds to process each batch of the dense synthetic dataset.

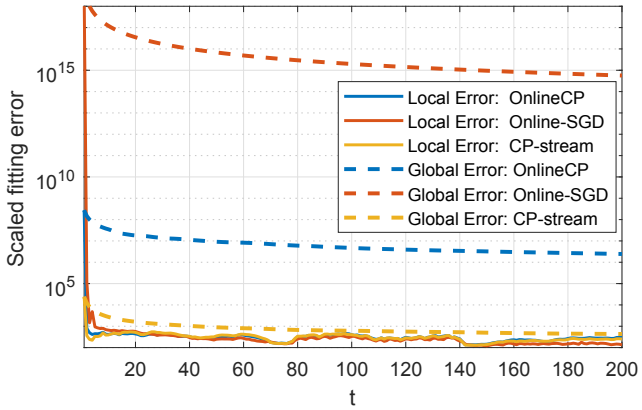

Figure 4: Scaled fitting error on the dense `AirportHall` tensor. **Local** and **Global** errors are defined by Equations (6.9) and (6.10), respectively.

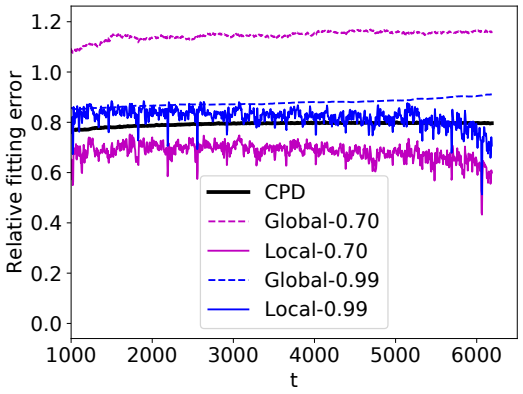

Figure 5: Relative fitting error on the `ChicagoCrime` tensor for $K=50$ and two forgetting factors (μ). **CPD** denotes a non-streaming CPD using all preceding batches. **Local** denotes the relative fitting error using only the previous ten batches. The first 1000 batches are executed, but not shown due to noise resulting from few data points (i.e., orders of magnitude fewer non-zeros than later times).

uniqueness property of the CPD model. We see that the estimation error rapidly converges to a small value, and the final estimation quality of CP-stream is more than 10 times better than that of Online-SGD. Online-CP fails to obtain a good estimate of the latent factors within the given 1000 data samples. Notice that we also need to tune the step size for Online-SGD, which is a non-trivial task. For this simulation we set the step size of Online-SGD to be 2, which works the best among a few dozens of ones that we tried. If we change the problem sizes and/or noise levels, this parameter needs to be tuned again for better performances. CP-stream, on the other hand, is completely free from tuning—we only need to choose an appropriate λ, and our general rule-of-thumb is simply a relatively small number—which is a good property for practical purposes.

Figure 3 shows the execution time for each iteration while factoring the synthetic dataset. Since Online-SGD only does a simple stochastic gradient step to update $\boldsymbol{A}^{(n)}$, it is much faster than that of Online-CP. An interesting observation of our proposed CP-stream is that the number of ADMM (4.7) iterations quickly drops down to 1, therefore after about $t>30$, the per-iteration execution time of CP-stream is comparable to Online-SGD.

6.3 Real-World Data Evaluations We evaluate the fitting ability of the methods on the `AirportHall` tensor in Figure 4. We measure two forms of fitting error. Solid lines denote the instantaneous fitting error, defined as

$$\frac{1}{I_1 I_2} \left\| \boldsymbol{\mathcal{X}}_t - [\![\boldsymbol{A}_t^{(1)}, \boldsymbol{A}_t^{(2)}; \boldsymbol{s}_t]\!] \right\|^2, \tag{6.9}$$

and dashed lines denote the overall fitting error, defined as

$$\frac{1}{I_1 I_2 t} \sum_{i=1}^{t} \left\| \boldsymbol{\mathcal{X}}_i - [\![\boldsymbol{A}_t^{(1)}, \boldsymbol{A}_t^{(2)}; \boldsymbol{s}_i]\!] \right\|^2. \tag{6.10}$$

All three methods perform well in terms of local error. However, only CP-stream is able to reduce the global error down to comparable scale as the local error. We attribute this to Online-CP and Online-SGD having large errors in the estimates of the $\boldsymbol{s}_t$ vectors over the first few batches, which dominate global error and are not revisited in a streaming setting. CP-stream, on the other hand, is able to obtain very accurate estimate of the $\boldsymbol{s}_t$ vectors at an early stage, giving rise to the strong performance in terms of global error.

Figure 5 shows the fitting error over time on the sparse `ChicagoCrime` tensor using two forgetting factors, μ. We now use *relative* error for the sparse dataset, as it lends a more interpretable result. Relative global error is defined as

$$\frac{1}{\|\boldsymbol{\mathcal{X}}\|^2} \left\| \boldsymbol{\mathcal{X}} - [\![\{\boldsymbol{A}^{(n)}\}; \boldsymbol{S}]\!] \right\|^2, \tag{6.11}$$

and local error is defined similarly. We compare against

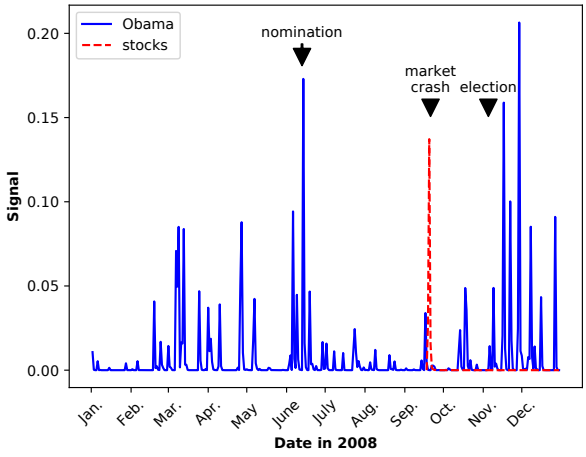

Figure 6: Signal of the word "Obama" and the community "stocks" over the year 2008 in the `Reddit2008` dataset. **Signal** is the inner product of the time vector $\boldsymbol{s}_t$ and the corresponding rows of the factor matrices. Signals are from the same rank-100 non-negative factorization.

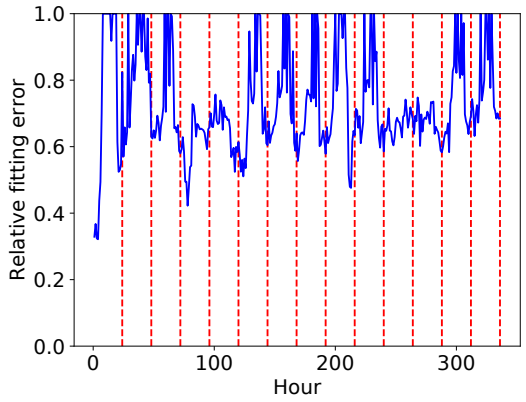

Figure 7: The relative reconstruction error of each slice over two weeks of the `CyberLANL` tensor. Dashed red lines are placed every 24 hours. We use K=100, $\mu = 0.2$ to closely track localized behavior, and λ=0.01, though we found all small values of λ to show similar results.

a batched CPD as baseline. At each time t, CPD simply recomputes a new factorization using all preceding batches. We do not evaluate against Online-CP because it has not been adapted to support sparsity, and we do not evaluate against Online-SGD due to its treatment of sparse data (i.e., they treat sparsity as *missing* values, which is useful in other applications that we do not consider).

When μ=0.99, both the global and local error closely follow the CPD baseline. This indicates that CP-stream can learn trends from a noisy dataset with a small accuracy loss compared to the significantly more expensive batched CPD computation. When μ=0.70, CP-stream instead trades off a loss of higher global error for a model which captures local information better than the CPD baseline. We can thus conclude that the "forgetting" parameter μ allows CP-stream to be useful in a variety of applications.

6.4 Case Studies Lastly, we evaluate the capability of CP-stream to perform two typical streaming tasks.

Discussion tracking. Figure 6 demonstrates the capability of CP-stream to discover events in a real-world streaming dataset. We compute the rank-100 non-negative CPD of the `Reddit2008` tensor and simultaneously track the activity of a word ("Obama") and a community ("stocks") over time. At each timestep t, we extract the corresponding latent vectors representing the tracked concepts and compute their inner product with $\boldsymbol{s}_t$. Since the factorization is non-negative, the resulting value is easily interpretable and can be viewed as a level of activity (denoted *signal*). Several major events from 2008 emerge as clear outliers, such as the USA presidential election and the stock market crash.

Network traffic analysis. We examine two weeks of network traffic in the `CyberLANL` dataset in Figure 7. Values are the relative reconstruction error at one hour increments. A clear day-night cycle emerges, as nighttime features less traffic and can be better modeled by a low-rank factorization. Moreso, we can observe spans of two days with low error, which correspond to weekends which also feature low amounts of traffic. Spans of high error during the daytime decrease over time, as the factorization is able to better learn the model for both day and night events.

7 Conclusions

Tensor factorization is a powerful tool for multi-way data analysis. In many applications, we are interested in factoring a tensor which includes temporal data that is not available ahead of time, and can potentially be of unbounded length (i.e., it is *streamed*). In this work, we present CP-stream, an algorithm for streaming tensor factorization that is suitable for both dense and sparse data. CP-stream is shown to converge faster than state-of-the-art approaches on synthetic and real-world datasets and improve factorization quality by 10×. It can additionally incorporate constraints such as non-negativity or factor sparsity. We demonstrate its applicability to large-scale sparse data in two application areas and identify events and patterns in real-world datasets.

Acknowledgments

Shaden Smith was located at the University of Minnesota during the majority of this work. This work was supported in part by NSF (IIS-0905220, OCI-1048018, CNS-1162405, IIS-1247632, IIP-1414153, IIS-1447788), Army Research Office (W911NF-14-1-0316), a University of Minnesota Doctoral Dissertation Fellowship, Intel Software and Services

Group, and the Digital Technology Center at the University of Minnesota. Access to research and computing facilities was provided by the Digital Technology Center and the Minnesota Supercomputing Institute. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] B. W. Bader, M. W. Berry, and M. Browne. Discussion tracking in enron email using parafac. *Survey of Text Mining II*, pages 147–163, 2008.

[2] L. Balzano, R. Nowak, and B. Recht. Online identification and tracking of subspaces from highly incomplete information. In *Communication, Control, and Computing (Allerton), Annual Allerton Conference on*, pages 704–711. IEEE, 2010.

[3] M. Baskaran, M. H. Langston, T. Ramananandro, D. Bruns-Smith, T. Henretty, J. Ezick, and R. Lethin. Accelerated low-rank updates to tensor decompositions. In *High Performance Extreme Computing Conference (HPEC)*. IEEE, 2016.

[4] A. Beutel, P. P. Talukdar, A. Kumar, C. Faloutsos, E. E. Papalexakis, and E. P. Xing. Flexifact: Scalable flexible factorization of coupled tensors on hadoop. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 109–117. SIAM, 2014.

[5] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[6] D. Bruns-Smith, M. M. Baskaran, J. Ezick, T. Henretty, and R. Lethin. Cyber security through multidimensional data decompositions. In *Cybersecurity Symposium (CYBERSEC), 2016*, pages 59–67. IEEE, 2016.

[7] L. Chiantini and G. Ottaviani. On generic identifiability of 3-tensors of small rank. *SIAM Journal on Matrix Analysis and Applications*, 33(3):1018–1037, 2012.

[8] H. Fanaee-T and J. Gama. Tensor-based anomaly detection: An interdisciplinary survey. *Knowledge-Based Systems*, 98:130–147, 2016.

[9] E. Gujral, R. Pasricha, and E. E. Papalexakis. SamBaTen: Sampling-based batch incremental tensor decomposition. *arXiv preprint arXiv:1709.00668*, 2017.

[10] K. Huang, N. D. Sidiropoulos, and A. P. Liavas. A flexible and efficient algorithmic framework for constrained matrix and tensor factorization. *IEEE Transactions on Signal Processing*, 64(19):5052–5065, 2016.

[11] H. Kasai. Online low-rank tensor subspace tracking from incomplete data by cp decomposition using recursive least squares. In *Acoustics, Speech and Signal Processing (ICASSP), International Conference on*. IEEE, 2016.

[12] A. D. Kent. Cybersecurity Data Sources for Dynamic Network Research. In *Dynamic Networks in Cybersecurity*. Imperial College Press, 2015.

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

[14] D. Koutra, E. E. Papalexakis, and C. Faloutsos. Tensorsplat: Spotting latent anomalies in time. In *Informatics (PCI), 2012 16th Panhellenic Conference on*, pages 144–149. IEEE, 2012.

[15] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online Learning for Matrix Factorization and Sparse Coding. *Journal of Machine Learning Research*, 11:19–60, 2010.

[16] 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.

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

[18] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos. ParCube: Sparse parallelizable tensor decompositions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2012.

[19] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017.

[20] S. Smith, A. Beri, and G. Karypis. Constrained tensor factorization with accelerated AO-ADMM. In *46th International Conference on Parallel Processing (ICPP '17)*. IEEE, 2017.

[21] 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.

[22] S. Smith and G. Karypis. Tensor-matrix products with a compressed sparse tensor. In *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*. ACM, 2015.

[23] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis. SPLATT: Efficient and parallel sparse tensor-matrix multiplication. In *Parallel and Distributed Processing Symposium (IPDPS), International*. IEEE, 2015.

[24] J. Sun, D. Tao, S. Papadimitriou, P. S. Yu, and C. Faloutsos. Incremental tensor analysis: Theory and applications. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2(3):11, 2008.

[25] M. Vandecappelle, N. Vervliet, and L. De Lathauwer. Nonlinear least squares updating of the canonical polyadic decomposition. Technical report, KU Leuven, 2017.

[26] N. Vervliet and L. De Lathauwer. A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors. *IEEE Journal of Selected Topics in Signal Processing*, 10(2):284–295, 2016.

[27] B. Yang. Projection approximation subspace tracking. *IEEE Transactions on Signal processing*, 43(1):95–107, 1995.

[28] 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*. ACM, 2016.