SAND2023-02003C

Streaming Generalized Canonical Polyadic Tensor Decompositions

Eric T. Phipps
Center for Computing Research,
Sandia National Laboratories
Albuquerque, New Mexico, USA
etphipp@sandia.gov

Nicholas T. Johnson
Cerebras Systems Inc
Sunnyvale, California, USA
nick@cerebras.net

Tamara G. Kolda
MathSci.ai
Dublin, California, USA
tammy.kolda@mathsci.ai

ABSTRACT

In this paper, we develop a method which we call OnlineGCP for computing the Generalized Canonical Polyadic (GCP) tensor decomposition of streaming data. GCP differs from traditional canonical polyadic (CP) tensor decompositions as it allows for arbitrary objective functions which the CP model attempts to minimize. This approach can provide better fits and more interpretable models when the observed tensor data is strongly non-Gaussian. In the streaming case, tensor data is gradually observed over time and the algorithm must incrementally update a GCP factorization with limited access to prior data. In this work, we extend the GCP formalism to the streaming context by deriving a GCP optimization problem to be solved as new tensor data is observed, formulate a tunable history term to balance reconstruction of recently observed data with data observed in the past, develop a scalable solution strategy based on segregated solves using stochastic gradient descent methods, describe a software implementation that provides performance and portability to contemporary CPU and GPU architectures and demonstrate the utility and performance of the approach and software on several synthetic and real tensor data sets.

CCS CONCEPTS

• **Computing methodologies → Factorization methods**; **Online learning settings**.

PASC '23, June 26–28, 2023, Davos, Switzerland
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0190-0/23/06…$15.00
https://doi.org/10.1145/3592979.3593405

KEYWORDS

tensor decomposition, canonical polyadic, streaming

ACM Reference Format:
Eric T. Phipps, Nicholas T. Johnson, and Tamara G. Kolda. 2023. Streaming Generalized Canonical Polyadic Tensor Decompositions. In *Platform for Advanced Scientific Computing Conference (PASC '23), June 26–28, 2023, Davos, Switzerland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3592979.3593405

1 INTRODUCTION

We consider the problem of computing a generalized canonical polyadic (GCP) tensor decomposition [14, 19] in the situation where data is *streaming*. Generally speaking, the streaming paradigm assumes algorithms must update with a limited amount of data and limited passes on that data. The data may be streaming because the volume of data is too large to fit in memory all at once; however, the more general case is that the data is streaming because it is temporal and so arrives incrementally. For example, consider a tensor that captures crime statistics in the city of Chicago [6] so that entry (i, j, k) is the number of crimes of type i, in neighborhood j, at hour k. In the streaming scenario, we receive a new 3-way tensor of crime statistics every day, and we need to incorporate that information into the model.

The arrival of new data can be thought of in two different ways. We could view the Chicago Crime data as a 3-way tensor (type × neighborhood × hour) with new observations each day that can be considered a *statistical sample*. Alternatively, we can have an explicit time mode. For Chicago Crime, the tensor is then a 4-way tensor (type × neighborhood × hour × day) with a new hyperslice *appended* daily. In this latter case, the fourth mode corresponding to the day is growing and referred to as the *temporal* mode. The GCP tensor decomposition computes a *factor matrix* for each mode, so in the Chicago Crime example, we have a crime-type factor matrix for mode one, a neighborhood factor matrix for mode two, and an hour factor matrix for mode three, whether we think of it as a 3-way or 4-way tensor. In the 4-way interpretation, we additionally have a day factor matrix for mode four, and a new row is added to that factor matrix with each new day of data. In the 3-way interpretation, we can think instead of adjusting the weights of the factors each day. Ultimately, these two viewpoints are not very different since the new row in the temporal factor matrix in the 4-way interpretation is roughly equivalent to the updated weights in the 3-way interpretation.

A more interesting assumption is whether or not the underlying generative processes are changing with time. There is always some balancing of new information and old. If these processes are unchanging, then we may expect that our estimates of the non-temporal factor matrices will converge after a suitable number of observations. Such a formulation assumes observed data comes from some consistent, but unknown, distribution. This aligns better with incremental algorithms that progressively converge to a single value, and the ordering of observations aligns with sampling assumptions. In this situation, it is often useful to give a heavy weight to older information, slowing the amount of allowed change in the model as more observations accumulate. In fact, in these cases, the order that the information arrives is irrelevant.

In most cases of interest, however, the generative processes are changing as well, and we are interested in understanding these shifts. This is sometimes referred to as *concept drift*, where it is assumed the distributions of observed data are evolving in time [11]. Algorithms must be designed to adapt with drifting data, and the ordering of observations is critically important to track the evolution of the data distribution. In this case, we need to balance between adapting to changing generative processes without confusing them for statistical fluctuations.

Much research has been done in the case of the canonical polyadic (CP), also known as CANDECOMP/PARAFAC, decomposition. In this work, we extend existing methods to GCP, which differs from CP in that GCP allows for arbitrary objective functions. In particular, our contributions are as follows:

- As there is no single streaming problem formulation in the literature, we provide a concise overview of the streaming literation emphasizing the various assumptions made in different works (Section 3).
- We extend the GCP formalism to the streaming context by deriving a GCP optimization problem to be solved as each tensor slice is observed, enabling CP factorization of tensors using arbitrary objective functions (Subsection 4.1).
- This formulation incorporates a tunable history term into the optimization problem to balance reconstruction of recently observed data with data observed in the past.
- We develop a solution strategy for the GCP streaming problem based on segregated solves of the temporal weights and factor matrices using stochastic gradient descent solution methods (Subsection 4.2).
- We provide a highly performant software implementation of the algorithm in our GenTen software package leveraging low-level math kernels implemented on top of the Kokkos framework providing scalable thread parallelism and portability to contemporary CPU and GPU architectures (Section 5).
- We demonstrate the utility and performance of the approach on a variety of synthetic and realistic data sets using several GCP objective functions (Section 6).

2 BACKGROUND AND NOTATION

In this work, we assume the reader to be generally familiar with tensors and tensor decomposition methods. For a thorough overview, we refer the reader to Kolda and Bader [18]. Following standard practice, we denote tensors by bold calligraphic letters (e.g., $\boldsymbol{\mathcal{X}}$), matrices by bold capital letters ($\mathbf{A}$), vectors by bold lowercase letters ($\mathbf{a}$) and scalars by lowercase letters (a). We use multi-index notation to indicate tensor elements, i.e., $x_i \equiv x_{i_1 \ldots i_d}$ denotes the entry $i = (i_1, \ldots, i_d) \in \mathcal{I} \equiv \{1, \ldots, I_1\} \otimes \cdots \otimes \{1, \ldots, I_d\}$ of the d-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$.

2.1 Canonical Polyadic (CP) Tensor Decompositions

For a given d-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$, the Canonical Polyadic (CP) decomposition, also known as the CANDACOMP/PARAFAC decomposition, attempts to find a good approximating low-rank model tensor $\boldsymbol{\mathcal{M}}$ of the form

$$\boldsymbol{\mathcal{X}} \approx \boldsymbol{\mathcal{M}} = \sum_{j=1}^{R} s_j \; \mathbf{a}_j^{(1)} \circ \mathbf{a}_j^{(2)} \circ \cdots \circ \mathbf{a}_j^{(d)} \tag{1}$$

where s_j is a scalar weight, $\mathbf{a}_j^{(k)}$ is a column vector of size I_k, $\circ$ represents the tensor outer product, and R is the approximate rank. The column vectors for each mode k are often collected into a matrix $\mathbf{A}^{(k)} = [\mathbf{a}_1^{(k)} \cdots \mathbf{a}_R^{(k)}]$ of size $I_k \times R$ called a factor matrix. Given a weight vector $\mathbf{s} = [s_1 \cdots s_R]^T$ and factor matrices $\{\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}\}$, we refer to the resulting low-rank model $\boldsymbol{\mathcal{M}}$ as a Kruskal tensor (or K-tensor for short) and use the short-hand notation $\boldsymbol{\mathcal{M}} = [\![\mathbf{s}; \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}]\!]$ [1]. For traditional CP decompositions, $\boldsymbol{\mathcal{M}}$ is computed by solving a nonlinear least-squares problem

$$\min_{\boldsymbol{\mathcal{M}}} \; \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{M}}\|_F^2 \quad \text{s.t.} \quad \boldsymbol{\mathcal{M}} = [\![\mathbf{s}; \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}]\!] \tag{2}$$

where $\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{M}}\|_F^2 = \sum_{i \in \mathcal{I}} (x_i - m_i)^2$, with $\mathcal{I}$ defined as above, denotes the tensor Frobenius (sum-of-squares) norm. Note that in eq. (2), the minimization is with respect to both the weights $\mathbf{s}$ and factor matrices $\{\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}\}$. Many approaches have been developed for efficiently solving eq. (2) that are scalable to large, sparse tensors. However, a very common, successful approach that is also relevant to the streaming problem is alternating least-squares (called CP-ALS [3, 13]) which is an iterative method, that for each iteration, cycles over modes $k = 1, \ldots, d$, holds all of the modes other than mode k fixed, and solves the resulting linear least squares problem for $\mathbf{A}^{(k)}$.

2.2 Generalized CP Decompositions

As described in [14], the CP problem (2) is equivalent to a maximum likelihood estimation procedure where the entries x_i of the tensor of $\boldsymbol{\mathcal{X}}$ are i.i.d. Gaussian with with mean m_i and some variance σ^2 which is constant across the tensor, i.e., $x_i \sim \mathcal{N}(m_i, \sigma)$. Such a statistical assumption may not be appropriate for many types of data (e.g., count or binary), motivating the development of the Generalized Canonical Polyadic (GCP) method [14]. In this method, it is assumed the tensor entries follow some known, parameterized probability distribution $x_i \sim p(x_i|\eta_i)$ determining the likelihood of each entry x_i, where η_i is the (unknown) parameter of the distribution. In this case, the CP model is computed to maximize the likelihood $p(x_i|\eta_i)$ of the tensor entry observation x_i through an invertible link function $\ell(\eta_i) = m_i$ connecting the CP model parameter m_i to the distributional parameter η_i. This results in the more general optimization problem

$$\min_{\boldsymbol{\mathcal{M}}} \; F(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{M}}) = \sum_{i \in \mathcal{I}} f(x_i, m_i) \quad \text{s.t.} \quad \boldsymbol{\mathcal{M}} = [\![\mathbf{s}; \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}]\!], \tag{3}$$

where as before $\mathcal{I} = \{1, \ldots, I_1\} \otimes \cdots \otimes \{1, \ldots, I_d\}$ is the set of all tensor multi-indices (including both zeros[1] and nonzeros). Here $f(x, m) = -\log p(x|\ell^{-1}(m))$ is the negative log-likelihood and is called the loss function. For example, one may have $f(x, m) = m - x \log m$ with $\ell(\eta) = \eta$ for a tensor containing count data, assuming a Poisson distribution, or $f(x, m) = \log(m+1) - x \log m$ with $\ell(\eta) = \eta/(1-\eta)$ for a binary tensor assuming a Bernoulli distribution.[2] See [14] for a detailed derivation of these loss functions for different statistical distributions. In the Gaussian case, $f(x, m) = (x - m)^2$, so eq. (2) becomes a special case. It is important to note that in the general case however, the CP model no longer approximates the tensor itself, but rather the natural parameter of the distribution underlying the assumed statistical model of the tensor data, which will be crucial for the streaming method described later.

The challenge in the GCP method is solving eq. (3) for general loss functions f which loses the least-squares structure, making ALS-type approaches impossible. In [14], the authors instead pursue gradient-based optimization approaches and derive the corresponding gradient formulas:

$$\frac{\partial F}{\partial \mathbf{A}^{(k)}} = \mathbf{Y}_{(k)} \mathbf{Z}_k \operatorname{diag}(\mathbf{s}), \quad k = 1, \ldots, d, \tag{4}$$

$$\frac{\partial F}{\partial \mathbf{s}} = \mathbf{Z}^\intercal \mathbf{y} \tag{5}$$

where $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ is a gradient tensor defined by

$$y_i = \frac{\partial f}{\partial m}(x_i, m_i), \quad i \in \mathcal{I}. \tag{6}$$

Here $\mathbf{Y}_{(k)}$ denotes the mode-k matricization/unfolding of $\mathcal{Y}$, $\mathbf{y} = \operatorname{vec}(\mathcal{Y})$ is the vectorization of $\mathcal{Y}$,

$$\mathbf{Z}_k = \mathbf{A}^{(d)} \odot \cdots \odot \mathbf{A}^{(k+1)} \odot \mathbf{A}^{(k-1)} \odot \cdots \odot \mathbf{A}^{(1)}, \quad k = 1, \ldots, d, \tag{7}$$

$$\mathbf{Z} = \mathbf{A}^{(d)} \odot \mathbf{A}^{(d-1)} \odot \cdots \odot \mathbf{A}^{(1)}, \tag{8}$$

and $\odot$ denotes the Khatri-Rao product. Thus the factor matrix gradients (4) are given by the Matricized Tensor Times Khatri-Rao Product (MTTKRP) involving the gradient tensor $\mathcal{Y}$. Note that $\mathcal{Y}$ is in general dense, even if $\mathcal{X}$ is sparse, making traditional gradient-based methods impractical for large, sparse $\mathcal{X}$. Instead, the authors in [19] leverage stochastic gradient descent (SGD) in this case, using randomly sampled gradients of the form

$$\frac{\partial F}{\partial \mathbf{A}^{(k)}} \approx \tilde{\mathbf{Y}}_{(k)} \mathbf{Z}_k \operatorname{diag}(\mathbf{s}), \quad k = 1, \ldots, d, \tag{9}$$

$$\frac{\partial F}{\partial \mathbf{s}} \approx \mathbf{Z}^\intercal \tilde{\mathbf{y}} \tag{10}$$

where $\tilde{\mathcal{Y}}$ is a sparse, randomly sampled approximation of $\mathcal{Y}$. In the sequel, we will leverage these formulas for developing the streaming GCP algorithm for sparse tensors, employing the sparse, stratified sampling methodology of [19].

[1]In this work, zero values represent observed values that are zero, and must be accurately reflected in the CP model, as opposed to unobserved values that can be ignored as in tensor completion.

[2]Log-likelihood terms that are constant with respect to the model are dropped in the loss function. Furthermore, $\log m$ is in practice replaced by $\log(m + \epsilon)$ where ϵ is a small constant to allow $m = 0$. Finally, depending on the choice of loss function, the minimization problem eq. (3) may include additional constraints such as $m_i \geq 0$.

3 RELATED WORK

We review the work in the domain of streaming or online CP tensor decomposition. There is no single well-defined problem in this context, so we try to explain the different formulations and assumptions. We make a few assumptions throughout.

- Updates are processed in *discrete* batches indexed by time $t = 1, 2, \ldots$.
- At each time t, a complete d-way tensor is observed, unless otherwise stated.
- Dimensions are fixed throughout all time, unless otherwise stated.
- The CP rank is known and fixed, unless otherwise stated.

3.1 Problem Setup for Two-way Temporal Slices

In the case that $d = 2$, we receive a matrix $\mathbf{X}_t \in \mathbb{R}^{I \times J}$ for each time $t = 1, 2, \ldots$.

If the temporal mode is finite so that $t = 1, \ldots, T$, we can consider that $\mathcal{X}$ is the $I \times J \times T$ tensor formed by stacking all time slices. For a given rank R, the standard goal is to find factor matrices $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, and $\mathbf{S} \in \mathbb{R}^{T \times R}$ that minimize

$$\min_{\mathbf{A},\mathbf{B},\mathbf{S}} \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{t=1}^{T} (x_{ijt} - m_{ijt})^2 \quad \text{s.t.} \quad m_{ijt} = \sum_{\ell=1}^{R} a_{i\ell} b_{j\ell} s_{t\ell}.$$

There are a few different streaming and online formulations of this problem.

In one formulation, the challenge is that we can only see one (or a few) temporal slices $\mathbf{X}_t$ at any given time. This may be due to memory constraints. However, we assume that the factor matrices $\mathbf{A}$ and $\mathbf{B}$ are fixed for all time.

In other versions, the factor matrices $\mathbf{A}$ and $\mathbf{B}$ can change over time. Then the problem becomes more interesting. Let $\mathbf{s}_t \in \mathbb{R}^R$ denote row t (transposed) of $\mathbf{S}$, i.e.,

$$\mathbf{S} = \begin{bmatrix} \mathbf{s}_1 & \cdots & \mathbf{s}_T \end{bmatrix}^\intercal. \tag{11}$$

Then, at time step t, the goal is to find $\mathbf{s}_t \in \mathbb{R}^R, \mathbf{A} \in \mathbb{R}^{I \times R}, \mathbf{B} \in \mathbb{R}^{J \times R}$ that minimize

$$\min_{\mathbf{A},\mathbf{B},\mathbf{s}_t} \|\mathbf{X}_t - \mathbf{M}_t\|^2 \equiv \sum_{i=1}^{I} \sum_{j=1}^{J} (x_{ijt} - m_{ijt})^2 \quad \text{s.t.} \quad \mathbf{M}_t = \mathbf{A} \operatorname{diag}(\mathbf{s}_t) \mathbf{B}^\intercal$$

If we only fit $\mathbf{X}_t$, however, the problem is not well defined, i.e., it does not produce essentially unique minimizers $\mathbf{A}$ and $\mathbf{B}$. Instead, at time t, it is common to include some *historical* information in the objective function, the exact details of which depend on the formulation.

It can be argued that a truly streaming problem is not finite, so $t = 1, 2, \ldots$. In that case, we cannot save all the historical information. Additionally, such problems are generally more interesting if the factor matrices change slowly in time; otherwise, we can assume that the factor matrices would be learned within finite time and the only thing changing at each time step are the weights $\mathbf{s}_t$.

3.2 Problem Setup for Higher-order Temporal Slices

If $d > 2$, the updates are tensors. For each time $t = 1, 2, \ldots$, we receive a tensor $\mathcal{X}_t \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$. At time t, the goal is to find factor matrices $\mathbf{A}^{(k)} \in \mathbb{R}^{I_k \times R}$ for $k = 1, \ldots, d$ and weights $\mathbf{s}_t \in \mathbb{R}^R$ that minimize

$$\min_{\mathbf{s}_t, \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}} \|\mathcal{X}_t - \mathcal{M}_t\|^2 \equiv \sum_{i \in \mathcal{I}} (x_{it} - m_{it})^2 \quad \text{s.t.} \quad \mathcal{M}_t = [\![\mathbf{s}_t; \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}]\!], \tag{12}$$

generally with some methodology for incorporating historical information. Here we use the shorthand $x_{it} \equiv \mathcal{X}_t(i_1, \ldots, i_d)$ and $m_{it} \equiv \mathcal{M}_t(i_1, \ldots, i_d)$.

3.3 Earliest Work

To the best of our knowledge, the earliest work in this area is Nion and Sidiropoulos [22]. They consider the case where each observation is a two-way matrix, as in Subsection 3.1. The rank and sizes are fixed across time, and the factors are assumed to be *slowly* varying, though there are no experiments with factors that varied in time in that work. Their primary focus was on demonstrating an alternative method for fitting CP decompositions that traded a small amount of accuracy for increased speed. Their general formulation of the problem is as follows. At time t, solve

$$\min_{\mathbf{A}, \mathbf{B}, \mathbf{s}_t} \sum_{h=1}^{t} \theta^{t-h} \|\mathbf{X}_h - \mathbf{M}_h\|^2 \quad \text{s.t.} \quad \mathbf{M}_h = \mathbf{A}^{\intercal} \operatorname{diag}(\mathbf{s}_h) \mathbf{B} \text{ for all } h \in \{1, \ldots, t\}. \tag{13}$$

The parameter $\theta \in (0,1)$ downweights older hyperslices. The weights $\mathbf{s}_h$ are fixed for all $h < t$; however, the $\mathbf{A}$ and $\mathbf{B}$ matrices are updated at each time step. This means that $\mathbf{A}$ and $\mathbf{B}$ should still somewhat fit the older data. The summation over time incorporates *historical* information. We omit the details of the method since it is relatively complicated and has been subsequently bested by other methods. Technically, this method requires all historical information. However, because θ^{t-h} is exponentially decreasing, older information can effectively be discarded after a small number of time steps.

3.4 Online SGD

Mardani, Mateos, and Giannakis [21] consider both matrix and 3-way tensor streaming; we discuss only the tensor streaming part of their work. In contrast to eq. (13), Mardani et al. [21] account for missing data, add regularization, and have an entirely different computational approach.

To account for missing data, define the matrix

$$\mathbf{W}_t(i,j) = \begin{cases} 1 & \text{if entry } (i,j) \text{ is known at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

Additionally, Mardani et al. add regularization with parameter λ. At time t, the formulation is

$$\min_{\mathbf{A}, \mathbf{B}, \mathbf{s}_t} \sum_{h=1}^{t} \theta^{t-h} \|\mathbf{W}_h * (\mathbf{X}_h - \mathbf{M}_h)\|^2 + \bar{\lambda}_t (\|\mathbf{A}\|_F^2 + \|\mathbf{B}\|_F^2) + \lambda \|\mathbf{s}_t\|_2^2 \quad \text{s.t.} \quad \mathbf{M}_h = \mathbf{A}^{\intercal} \operatorname{diag}(\mathbf{s}_h) \mathbf{B} \text{ for all } h \in \{1, \ldots, t\}. \tag{14}$$

As with eq. (13), the parameter $\theta \in (0,1)$ downweights older hyperslices. For writing efficiency, we pulled the term for time t into the summation, but the weights $\mathbf{s}_h$ are fixed for all $h < t$. The asterisk ($*$) denotes *elementwise* multiplication, and the effect is that only observed entries are included in the summation. The definition of $\bar{\lambda}_t$ is somewhat unclear in the paper. At one point, it seems to propose that $\bar{\lambda}_t \equiv \lambda / \sum_{h=1}^{t} \theta^{t-h}$ to ensure a degree of consistent weighting as compared to the regular 3-way problem in the finite case, but the pseudo-code seems to indicate that either $\bar{\lambda}_t \equiv \lambda / t$ or $\bar{\lambda}_t \equiv \lambda / (t \sum_{h=1}^{t} \theta^{t-h})$. In the experiments, the regularization parameter is set according to a standard in matrix completion: $\lambda = \sqrt{2IJ\pi}\sigma$ where π is the proportion of sampled data at each time step and σ is the noise level.

Problem (14) is solved iteratively with two basic steps at each iteration. First, $\mathbf{s}_t$ is solved for via a closed form expression holding $\mathbf{A}$ and $\mathbf{B}$ fixed. Second, the method takes *one step* of gradient descent (GD) for updating $\mathbf{A}$ and $\mathbf{B}$. They call this *stochastic* gradient descent (SGD) because the updates are based only on the observed entries of $\mathbf{X}_t$, which may vary randomly from step to step.

There are some implicit assumptions that are not clearly stated in the paper. There is no proof that the stochastic gradient is correct in expectation. To do so requires some assumptions about how the data is sampled and also appropriate weighting. The experiments (on cardiac dynamic MRI and Internet traffic) are constructed so that it all works correctly enough—the data is sampled uniformly and the same number of samples are taken at each time step.

3.5 CP-Stream

CP-Stream [27] is similar to Online SGD [21]. The primary difference is that it avoids saving the older data and instead uses an approximation. It also works for $d > 2$. At time step t, CP-Stream executes two phases. Let $\bar{\mathbf{A}}^{(k)}$ denote the *old* factor matrices, i.e., from time $t-1$. The first phase computes $\mathbf{s}_t$ with all the old factor matrices by solving

$$\min_{\mathbf{s}_t} \|\mathcal{X}_t - \bar{\mathcal{M}}_t\|^2 + \lambda \|\mathbf{s}_t\|^2 \quad \text{s.t.} \quad \bar{\mathcal{M}}_t \equiv [\![\mathbf{s}_t; \bar{\mathbf{A}}^{(1)}, \ldots, \bar{\mathbf{A}}^{(d)}]\!]. \tag{15}$$

The second phases computes $\{\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}\}$, estimating the observations from prior time steps via $\mathcal{X}_h \approx \bar{\mathcal{M}}_h \equiv [\![\mathbf{s}_h; \bar{\mathbf{A}}^{(1)}, \ldots, \bar{\mathbf{A}}^{(d)}]\!]$:

$$\min_{\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}} \|\mathcal{X}_t - \mathcal{M}_t\|^2 + \sum_{h=1}^{t-1} \theta^{t-h} \|\bar{\mathcal{M}}_h - \mathcal{M}_h\|^2 \quad \text{s.t.} \quad \mathcal{M}_h = [\![\mathbf{s}_h; \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}]\!] \text{ for all } h = 1, \ldots, t. \tag{16}$$

3.6 Other Works

Numerous other streaming methods have been studied in the literature that are less relevant to the method proposed here. Several prominent examples summarized below.

OnlineCP. The OnlineCP method [32] works for arbitrary d-way tensors and solves exactly the standard least squares subproblems, regularizing by taking only a single step of ALS. The innovation is the clever reuse of expensive calculations when folding in each time slice. It effectively assumes that the factor matrices are fixed.

OLSTEC. The Online Low-Rank Subspace Tracking by Tensor CP Decomposition (OLSTEC) method [16] is similar to one of the methods proposed in [22], but it can handle missing data and is the first paper to consider changes in the factor matrices in its experimental results. The experiment results show that they do better in this regime than Online SGD [21].

MAST. Multi-aspect Streaming Tensor (MAST) [28] is notable because it allows for the non-temporal modes to grow in time.

SamBaTen. Sampling-Based Incremental Tensor Decomposition (SamBaTen) [12] samples multiple subtensors, factors those independently, and then merges the results. It depends heavily on the results being essentially unique and consistent across the subtensors, which necessarily assumes that the factors are not changing in time. The temporal aspect is not clear since the entire tensor (across all time) seems to be saved.

SeekAndDestroy. SeekAndDestory [23] handles concept drift by allowing the addition of new factors as time progresses. It is not specifically a streaming algorithm because it is not updating the factorization so much as augmenting it. It receives data in batches, computes the CP decomposition from scratch, and then it merges this with the information from prior batches. We do not consider this to be a streaming method because the existing decomposition is not updated directly. Instead, SeekAndDestroy finds those factors that are overlapping and then identifies older factors that do not appear in the new batch as well as any new factors in the new batch. For rank determination on each batch, it uses a heuristic called AutoTen. This method depends heavily on each new batch having sufficient information to compute a full and essentially unique decomposition as this is the only way to ensure that overlap with past factors can be identified.

The ENSIGN software [20] implements a method similar to SeekAndDestroy. In addition to CP-ALS, they include CP-APR [5] and CP-ALS-NN [4].

Bayesian Methods. Probabilistic Streaming Tensors (POST) [8] and Variational Bayesian Inference (VBI) [31] are two papers that propose priors for the tensor model. POST considers models for both continuous and binary data. VBI models each time step as a CP model plus sparse noise ($\mathcal{S}_t$) and Gaussian noise ($\mathcal{E}_t$):

$$\mathcal{X}_t = [\![\mathbf{s}_t; \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(d)}]\!] + \mathcal{S}_t + \mathcal{E}_t,$$

$$\begin{aligned}
\mathbf{A}^{(k)}(i_k, j) &\sim \mathcal{N}(0, \lambda_j) & & \\
\mathbf{s}_t(j) &\sim \mathcal{N}(0, \lambda_j) & \lambda_j &\sim \text{InvGamma}(\alpha_\lambda, \beta_\lambda) \\
\mathcal{S}_t(i_1, i_2, \dots, i_d) &\sim \mathcal{N}(0, \gamma_{i_1 i_2 \cdots i_d}) & \gamma_{i_1 i_2 \cdots i_d} &\sim \text{InvGamma}(\alpha_\gamma, \beta_\gamma) \\
\mathcal{E}_t(i_1, i_2, \dots, i_d) &\sim \mathcal{N}(0, \tau) & \tau &\sim \text{InvGamma}(\alpha_\tau, \beta_\tau)
\end{aligned}$$

The prior on the λ's encourages low-rank, and prior on the γ's allows for sparse outliers. Both methods consider decomposition of tensors with missing data for tensor completion.

4 STREAMING GCP

We now consider GCP factorization in the streaming context. We first motivate and describe the minimization problem for the streaming GCP problem, then describe the solution strategy, and then conclude with a summary of the solution algorithm which we call OnlineGCP.

4.1 Streaming GCP Problem Formulation

We are primarily interested in the infinite streaming problem where at each time step t, a new d-dimensional tensor $\mathcal{X}_t \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ is observed. We assume the dimensions $I_1, \dots, I_d$ of each tensor do not change, and at each time step, a complete d-way tensor is observed. Our work focuses on sparse tensors, but the method is equally applicable to dense. Since both the non-streaming GCP and Online SGD solution algorithms rely on gradient descent, our approach for streaming GCP is inspired by Online SGD.

For simplicity, we begin with the problem for a single temporal slice, $\mathcal{X}_t$, and add on history in the discussion that follows. For just time slice t, we pose the optimization problem for given rank R as

$$\begin{aligned}
\min_{\mathcal{M}_t} \quad & \sum_{i \in \mathcal{I}} f(x_{it}, m_{it}) + \frac{\lambda}{2} \sum_{k=1}^{d} \|\mathbf{A}^{(k)}\|_F^2 + \frac{\mu}{2} \|\mathbf{s}_t\|_2^2 \\
\text{s.t.} \quad & \mathcal{M}_t = [\![\mathbf{s}_t; \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(d)}]\!], \;\; \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(d)}, \mathbf{s}_t \geq l.
\end{aligned} \tag{17}$$

As above, we use the shorthand $x_{it} \equiv \mathcal{X}_t(i_1, \dots, i_d)$ and $m_{it} \equiv \mathcal{M}_t(i_1, \dots, i_d)$, $\mathbf{A}^{(k)} \in \mathbb{R}^{I_k \times R}$ for $k = 1, \dots, d$ are the factor matrices, and $\mathbf{s}_t \in \mathbb{R}^R$ is the weight for time t. We incorporate the option for a lower bound l on the factor matrix/weight entries (with the understanding that $l = 0$ for nonnegativity constraints and $l = -\infty$ for problems where there is no lower bound). As in Online SGD, we include regularization terms for the factor matrices $\mathbf{A}^{(k)}$ and weights $\mathbf{s}_t$ with multipliers λ and μ, respectively, to encourage low-rank solutions [2]. We do not explicitly include the dependency of the factor matrices on t since they are, ideally, less sensitive to time.

In general, we want to incorporate historical information to keep the factor matrices from changing too much at each time step. There are many ways such historical information could be included, and several of these have been used in the previous work discussed in Section 3. One possible approach is to add *historical regularization* to eq. (17) via the second term in the following where we define the historical model to be the old weight with the current factor matrices, i.e., $\mathcal{M}_h \equiv [\![\mathbf{s}_h; \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(d)}]\!]$:

$$\begin{aligned}
\min_{\mathcal{M}_t} \quad & \sum_{i \in \mathcal{I}} f(x_{it}, m_{it}) + \sum_{h \in \mathcal{H}_t} w_h \sum_{i \in \mathcal{I}} f(x_{ih}, m_{ih}) \\
& + \frac{\lambda}{2} \sum_{k=1}^{d} \|\mathbf{A}^{(k)}\|_F^2 + \frac{\mu}{2} \|\mathbf{s}_t\|_2^2 \\
\text{s.t.} \quad & \mathcal{M}_t = [\![\mathbf{s}_t; \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(d)}]\!], \;\; \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(d)}, \mathbf{s}_t \geq l.
\end{aligned} \tag{18}$$

Here, $\mathcal{H}_t \subseteq \{1, \dots, t-1\}$. The terms $x_{ih} = \mathcal{X}_h(i_1, \dots, i_d)$ and $m_{ih} = \mathcal{M}_h(i_1, \dots, i_d)$ index the "historical" tensors. The weights w_h control the importance of historical terms.

The time index t could be infinite, so we limit ourselves to a history window of fixed size such that $|\mathcal{H}_t| = \min\{\, t-1, H \,\}$ for some fixed constant H. Moreover, we impose $\mathcal{H}_t \subseteq \mathcal{H}_{t-1} \cup \{\, t-1 \,\}$ so that no older information is ever added to the history. *This means that we can discard all older information except what's in the history.* There are many ways in which the history window and weighting could be chosen. In our work, we use reservoir sampling [30] which works as follows. For $t \leq H+1$ we set $\mathcal{H}_t = \{1, \dots, t-1\}$. Then for $t > H+1$, We set $\mathcal{H}_t = \mathcal{H}_{t-1}$ with probability $1 - H/(t-1)$; otherwise, we set $\mathcal{H}_t$ to be $\mathcal{H}_{t-1}$ where we have ejected one existing element and replaced it with $t-1$. This ensures that $\mathcal{H}_t$ is a uniform random sample of $\{1,\ldots,t-1\}$. We use exponential weighting of the form $w_h = w\theta^{t-h}$ where $0 < \theta \le 1$ and w is a multiplier allowing the entire history term to be scaled by a constant.

The approach outlined so far stores at most H temporal slices, which could require significant memory storage if a large window size H is desired. Following the approach of CP-Stream (see Subsection 3.5), we would like to reduce storage costs further in the case where the factor matrices are assumed to change slowly in time by approximating these slices using the factor matrices $\bar{\mathbf{A}}^{(1)},\ldots,\bar{\mathbf{A}}^{(d)}$ from the previous time step. It is tempting to replace x_{ih} in eq. (18) with $\bar{m}_{ih}$ where $\bar{\mathcal{M}}_h \equiv [\![\mathbf{s}_h; \bar{\mathbf{A}}^{(1)},\ldots,\bar{\mathbf{A}}^{(d)}]\!]$ is the CP model derived from the prior factor matrices and the historical weights, and $\bar{m}_{ih} = \bar{\mathcal{M}}_h(i_1,\ldots,i_d)$. However, this is not a valid approximation in general because, as described in Subsection 2.2, the CP model generated in the GCP method does not directly approximate the data tensor, but instead the parameter of the assumed probability distribution (in fact, the support of x_{ih} may not even coincide with $\bar{m}_{ih}$, e.g., for a binary tensor, $x_{ih} \in \{0,1\}$ whereas $\bar{m}_{ih} \in (0,\infty)$). Instead, we propose adding a historical regularization term that penalizes changes in the CP model using the Frobenius norm:

$$\min_{\mathcal{M}_t} \sum_{i\in\mathcal{I}} f(x_{it}, m_{it}) + \frac{1}{2}\sum_{h\in\mathcal{H}_t} w_h \|\bar{\mathcal{M}}_h - \mathcal{M}_h\|_F^2 + \frac{\lambda}{2}\sum_{k=1}^{d}\|\mathbf{A}^{(k)}\|_F^2 + \frac{\mu}{2}\|\mathbf{s}_t\|_2^2$$
$$\text{s.t.} \quad \mathcal{M}_t = [\![\mathbf{s}_t; \mathbf{A}^{(1)},\ldots,\mathbf{A}^{(d)}]\!], \quad \mathbf{A}^{(1)},\ldots,\mathbf{A}^{(d)}, \mathbf{s}_t \ge l \tag{19}$$

We reiterate that in eq. (19), the optimization is over the factor matrices $\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(d)}$ and temporal weights $\mathbf{s}_t$ for each time step t, with $\bar{\mathbf{A}}^{(1)},\ldots,\bar{\mathbf{A}}^{(d)}$ and $\mathbf{s}_h$ for $h \in \mathcal{H}_t$ held fixed.[3]

4.2 Streaming GCP Solution Strategy

We now describe the proposed solution strategy for eq. (19), which we call OnlineGCP. Assume eq. (19) has already been solved for $h = 1,2,\ldots,t-1$ resulting in the current approximations $\bar{\mathbf{A}}^{(1)},\ldots,\bar{\mathbf{A}}^{(d)}$, $\mathbf{s}_{t-1}$, with $\mathbf{s}_h$ for $h \in \mathcal{H}_{t-1}$ known from prior iterations. First choose the new history window $\mathcal{H}_t \subseteq \mathcal{H}_{t-1} \cup \{t\}$. Given the new tensor slice $\mathcal{X}_t$, define

$$F(\mathcal{X}_t, \mathbf{A}^{(1)},\ldots,\mathbf{A}^{(d)}, \mathbf{s}_t) = \sum_{i\in\mathcal{I}} f(x_{it}, m_{it}) + \frac{1}{2}\sum_{h\in\mathcal{H}_t} w_h\|\bar{\mathcal{M}}_h - \mathcal{M}_h\|_F^2 + \frac{\lambda}{2}\sum_{k=1}^{d}\|\mathbf{A}^{(k)}\|_F^2 + \frac{\mu}{2}\|\mathbf{s}_t\|_2^2 \tag{20}$$

to be the streaming GCP objective function. We then solve eq. (19) using a two-step minimization procedure inspired by Online SGD. In particular, we first solve eq. (19) for $\mathbf{s}_t$ with $\mathbf{A}^{(k)} = \bar{\mathbf{A}}^{(k)}$ held fixed, namely

$$\min_{\mathbf{s}_t} F(\mathcal{X}_t, \mathcal{M}_t) = \sum_{i\in\mathcal{I}} f(x_{it}, m_{it}) + \frac{\mu}{2}\|\mathbf{s}_t\|_2^2 \quad \text{s.t.} \quad \mathbf{s}_t \ge l. \tag{21}$$

The history term and factor matrix regularization terms are dropped because they have no dependence on $\mathbf{s}_t$. To solve eq. (21), we use the SGD solver described in Subsection 2.2 for the static GCP problem, modified to only solve for $\mathbf{s}_t$ with the factor matrices fixed. Leveraging eq. (5), the gradient for this subproblem is

$$\frac{\partial F}{\partial \mathbf{s}_t} = \mathbf{Z}^\intercal \mathbf{y}_t + \lambda \mathbf{s}_t, \tag{22}$$

where as before $\mathbf{y}_t = \operatorname{vec}(\mathcal{Y}_t)$ and $\mathcal{Y}_t \in \mathbb{R}^{I_1\times\cdots\times I_d}$ is the gradient tensor for slice $\mathcal{X}_t$ defined by $y_{it} = \frac{\partial f}{\partial m}(x_{it}, m_{it})$. As in the static case, this tensor is sampled each SGD iteration resulting in each stochastic gradient.

Once $\mathbf{s}_t$ is computed, the factor matrices $\mathbf{A}^{(k)}$ are computed by applying a fixed number of ADAM SGD iterations to eq. (20), holding $\mathbf{s}_t$ fixed. Using eq. (4), the corresponding gradients can be shown to be

$$\frac{\partial F}{\partial \mathbf{A}^{(k)}} = (\mathcal{Y}_t)_{(k)}\mathbf{Z}_k \operatorname{diag}(\mathbf{s}_t) + \lambda\mathbf{A}^{(k)} + \sum_{h\in\mathcal{H}_t} w_h\Big(\mathbf{A}^{(k)}\operatorname{diag}(\mathbf{s}_h)\mathbf{Z}_k^T\mathbf{Z}_k\operatorname{diag}(\mathbf{s}_h) - \bar{\mathbf{A}}^{(k)}\operatorname{diag}(\mathbf{s}_h)\bar{\mathbf{Z}}_k^T\mathbf{Z}_k\operatorname{diag}(\mathbf{s}_h)\Big) \tag{23}$$

for $k = 1,\ldots,d$, where

$$\mathbf{Z}_k = \mathbf{A}^{(d)} \odot\cdots\odot \mathbf{A}^{(k+1)} \odot \mathbf{A}^{(k-1)} \odot\cdots\odot \mathbf{A}^{(1)}, \tag{24}$$
$$\bar{\mathbf{Z}}_k = \bar{\mathbf{A}}^{(d)} \odot\cdots\odot \bar{\mathbf{A}}^{(k+1)} \odot \bar{\mathbf{A}}^{(k-1)} \odot\cdots\odot \bar{\mathbf{A}}^{(1)}. \tag{25}$$

For both the temporal weight and factor matrix solves we employ the ADAM SGD update strategy [17] to make the SGD solves less sensitive to the choice of learning rate. Since the weights may change substantially from step to step, we restart each ADAM SGD solve anew with no tracking of ADAM moments across steps. However, for the factor matrix solves the ADAM moments are tracked across time steps as the changes in factor matrices from step to step are expected to be small.

4.3 Sampling for Streaming Stochastic GCP Approximations

Since each $\mathcal{Y}_t$ is in general dense, we must compute sampled approximations (denoted by $\tilde{\mathcal{Y}}_t$) for each SGD iteration. In principle, any sampling method can be used, but in this work we employ the stratified sampling approach of [19] where for each time step t, the set of sampled coordinates $\mathcal{I}_t$ for computing $\tilde{\mathcal{Y}}_t$, is partitioned into two disjoint sets consisting of indices corresponding to nonzeros and zeros in $\mathcal{X}_t$. As in [19], we assume these sets are formed by sampling uniformly, *with replacement*, p and q times from the sets of nonzeros and zeros, respectively (zeros are sampled by searching the tensor after each candidate is computed to verify the candidate is not a nonzero, and this continues until q samples have been generated). For each $i \in \mathcal{I}$, let $\tilde{p}_{it}$ be the number of times i is selected as a nonzero and $\tilde{q}_{it}$ the number of times it is selected as a zero.

[3] We note that a third possible approach is to directly penalize changes in the factor matrices by replacing the history regularization term in eq. (19) with a term of the form $\sum_{k=1}^{d}\tilde{w}_k\|\bar{\mathbf{A}}^{(k)} - \mathbf{A}^{(k)}\|_F^2$. However such an approach requires careful tuning of the regularization parameters $\tilde{w}_k$ since it does not incorporate the historical weights $\mathbf{s}_h$.

Then the entries of the sampled gradient tensor $\tilde{\mathcal{Y}}_t$ are given by

$$\tilde{y}_{it} = \left(\tilde{p}_{it}\frac{\eta_t}{p} + \tilde{q}_{it}\frac{\omega - \eta_t}{q}\right)\frac{\partial f}{\partial m}(x_{it}, m_{it}) \qquad (26)$$

where $\eta_t = \text{nnz}(\mathcal{X}_t)$ is the number of nonzeros in $\mathcal{X}_t$ and $\omega = \prod_{k=1}^{d} I_k$ is the total number of elements of $\mathcal{X}_t$ (which is independent of t). Since $\mathbb{E}[\tilde{p}_{it}] = p/\eta_t$ and $\mathbb{E}[\tilde{q}_{it}] = q/(\omega - \eta_t)$, it is easy to see that $\mathbb{E}[\tilde{y}_{it}] = y_{it}$. The history and regularization terms in eq. (23) could also be sampled, but since their true values can be computed efficiently, there is no reason to do so and their true gradient contributions are included in each stochastic gradient.

Similar sampling calculations are required for efficiently approximating the objective function F in eq. (20); however, as in [19], there are a few changes. First, we use a much larger number of samples when approximating F to ensure accuracy. Second, we use the same set of samples across all epochs within the temporal and factor matrix solvers for consistent estimations of convergence (but compute a different set of samples for the temporal and factor matrix solvers, and also for each slice $\mathcal{X}_t$). As before, we sample uniformly with replacement p' and q' times from the sets of indices corresponding to nonzeros and zeros, respectively. As in the gradient, the true value of the history and regularization objective terms can be efficiently computed, so the sampled approximation $\tilde{F}$ to F is given by

$$\begin{aligned}&\tilde{F}(\mathcal{X}_t, \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}, \mathbf{s}_t) = \\ &\sum_{i \in \mathcal{I}_t}\left(\tilde{p}'_{it}\frac{\eta_t}{p'} + \tilde{q}'_{it}\frac{\omega - \eta_t}{q'}\right) f(x_{it}, m_{it}) + \sum_{h \in \mathcal{H}_t} w_h \|\tilde{\mathcal{M}}_h - \mathcal{M}_h\|_F^2 \\ &\qquad + \frac{\lambda}{2}\sum_{k=1}^{d}\|\mathbf{A}^{(k)}\|_F^2 + \frac{\mu}{2}\|\mathbf{s}_t\|_2^2, \qquad (27)\end{aligned}$$

with similar definitions of $\tilde{p}'_{it}$ and $\tilde{q}'_{it}$. It is straightforward to see that $\mathbb{E}[\tilde{F}] = F$.

5 SOFTWARE IMPLEMENTATION

An open-source software implementation of the OnlineGCP approach described above is provided in the GenTen package for performance-portable tensor decompositions [24, 25]. This software package implements the required sampling procedures, MTTKRP, and SGD step procedures using the Kokkos C++ performance portability API [9, 10] allowing a single C++ implementation of each kernel to be executed with high performance on a variety of contemporary architectures, including multicore CPUs and manycore GPUs. For brevity, the details of this implementation are not provided here.

6 NUMERICAL EXPERIMENTS

We now present several numerical experiments that compare the accuracy of the OnlineGCP method with several static and streaming alternatives for Gaussian, Poisson, and Bernoulli loss functions. The static methods are applied to the entire $d+1$-way tensor formed by stacking the streamed slices across the temporal mode, whereas the streaming methods update their decomposition one slice at a time. For the streaming methods, we generate an initial CP model by applying an appropriate static decomposition method (i.e., CP-ALS for Gaussian, CP-APR for Poisson, or GCP for Bernoulli) to a small portion of the streaming data, which we call a warm-start. Throughout these experiments we will measure the effectiveness of the OnlineGCP approach by comparing *local* and *global* normalized reconstruction losses. In the context of streaming we define the local normalized reconstruction loss as the total loss for a given decomposition, divided by the norm of the data, for every observed time slice:

$$F_{local}(\mathcal{X}_t, \mathcal{M}_t) = \frac{1}{\|\mathcal{X}_t\|_F^2}\sum_{i \in \mathcal{I}} f(x_{it}, m_{it}) \qquad (28)$$

where $\mathcal{M}_t = [\![\mathbf{s}_t; \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}]\!]$ and $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}$ indicate the values of factor matrices computed at that point in time. This evaluates how well our current model fits the most recently observed data. For OnlineGCP, we compute a sampled approximation to F_{local} by subsampling $\mathcal{I}$ but not including history, factor matrix, or temporal weight regularization terms.

For the global normalized reconstruction loss we "back-test" the model at our final time step against all previously observed data. In particular, the functional form is the same as in eq. (28) however we use the factor matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}$ from the final time step. This evaluates how well the final model approximates all observed data. For OnlineGCP, we compute the true value of the global loss instead of a sampled approximation. Since the static methods use all available data, the local and global normalized reconstruction losses are identical.

The OnlineGCP method has numerous hyperparameters. The values used in each experiment are summarized in table 1. These values were chosen empirically to produce good results through hand-tuning, but are not necessarily optimal. Furthermore, OnlineGCP requires choosing the number of samples for the objective function and gradient evaluation for each iteration of SGD, and these values were chosen empirically to yield similar results to the other static and streaming methods. In comparing to static GCP, we used the same total number of samples as OnlineGCP by multiplying the number of samples used by OnlineGCP for each time slice by the number of slices.

Table 1: OnlineGCP hyperparameters for the numerical experiments. Here R is the rank of the CP model, α_w, α_f are the ADAM learning rates of the temporal weight and factor matrix solvers, respectively, κ_w, κ_f are the number of epochs for each solver, w is the multiplicative weight of the history term, H is the size of the history window, and H_{init} is the size of the warm-start. All experiments used $\theta = 1$ (no exponential down-weighting of historical slices), $\lambda = \mu = 0$ (no rank regularization penalty), $\tau_w = \tau_f = 100$ iterations per epoch, and the default ADAM update parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$).

Experiment	R	α_w	κ_w	α_f	κ_f	w	H	H_{init}
Synth. Gaussian	20	10.0	20	10^{-4}	5	1	50	10
Synth. Poisson	20	1.0	20	10^{-4}	10	10	50	10
Chicago Binary	50	0.1	5	10^{-3}	5	10	500	20
ArXiv Poisson	50	10.0	5	$5 \cdot 10^{-4}$	10	1	100	20

6.1 Synthetic Data Experiments

We first describe experiments with two synthetic data sets derived from randomly generated K-tensors which the computed decomposition methods should recover. For these experiments we also measure the congruence [29] between it and the K-tensor computed by each method (called the KTensor Score in the plots). A perfect recovery corresponds to a score of 1.0.

Gaussian. To construct a Gaussian-distributed synthetic data set, we first constructed a random 3-way, rank-20 K-tensor with factor matrices of size 300×20, 300×20, and 200×20, respectively. Each factor matrix entry was drawn uniformly at random from $(0, 1)$. This K-tensor provides the ground truth for the model. A dense tensor was then generated by multiplying out the K-tensor and perturbing each entry by draws from a zero-mean Gaussian distribution with a standard deviation of 0.2. This tensor is then streamed slice-by-slice, where each slice is a dense 300×300 matrix. The warm-start was generated by applying CP-ALS to the first 10 slices. We then compared our method using 10,000 nonzero samples for each objective/gradient evaluation, no zero samples (since the tensor is dense), and the remaining hyperparameters as indicated in table 1 to static CP-ALS and GCP applied to the full $300 \times 300 \times 200$ tensor, OnlineCP, and Online SGD. In fig. 1a we demonstrate comparable results for the local and global reconstruction loss, and K-tensor score with respect to the ground truth for the considered methods (although CP-ALS is able to achieve a somewhat better K-tensor score than all of the other methods).

Poisson. To generate a Poisson-distributed synthetic data set, we used the procedure described in [5] to generate a sparse 3-way tensor of size $300 \times 300 \times 200$, $R = 20$ factors, and roughly 3.2% nonzero sparsity. We then stream this tensor slice-by-slice in OnlineGCP as before, but this time comparing to static CP-APR and GCP with Poisson loss computed from the full tensor dataset. OnlineGCP used a warm-start constructed from applying CP-APR to the first 10 slices, all tensor nonzeros in the objective/gradient evaluations, and 50,000 and 10,000 zero samples for the objective and gradient, respectively. In fig. 1b we see fairly comparable results in losses and scores among all of the methods.

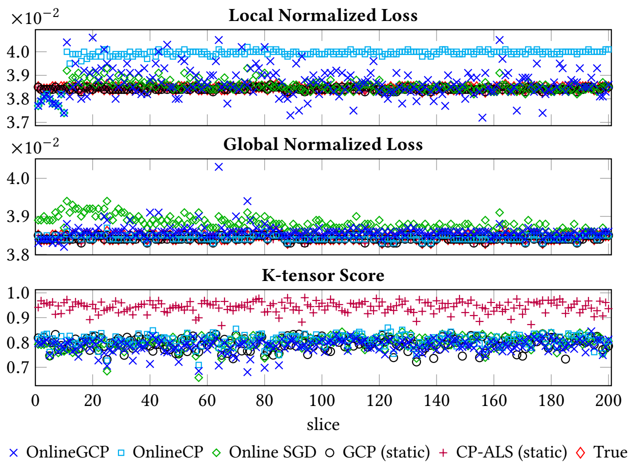
6.2 Realistic Data Experiments

We now present several experiments using real data tensors with non-Gaussian loss functions.

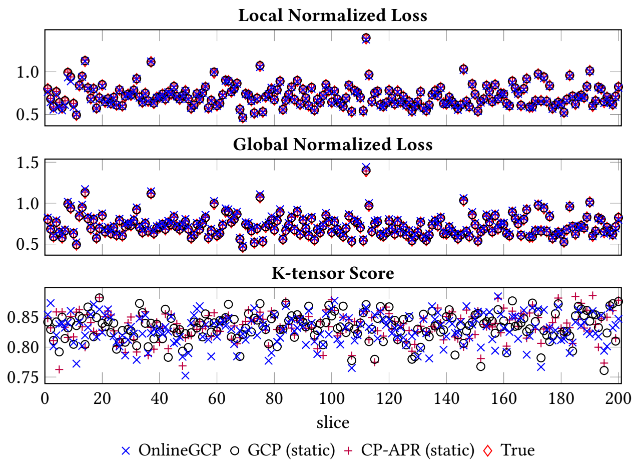
Chicago Crime. To demonstrate the approach for non-Gaussian loss, we used the Chicago Crime tensor provided by FROSTT [26] converted to a binary tensor where any nonzero value was replaced by one (this is reasonable since a majority of the entries are one anyway). Sticking with our streaming convention across the last mode, we oriented the tensor such that entry $\mathfrak{X}(i, j, k, l)$ denoted whether on hour i, crime j was committed in neighborhood k for day l from our first date. We used a starting data of $l = 500$ because there is significantly less data for days prior to this date. The resulting tensor is of size $24 \times 77 \times 32 \times 5687$ with roughly 1.6% sparsity. A warm-start for the first 20 days was generated via GCP with Bernoulli loss using 50,000 zero/nonzero samples for the objective function and 10,0000 zero/nonzero samples for the gradient. Given the relatively small tensor slices each time step, OnlineGCP used all nonzeros along with 10,000 and 1,000 zero samples for each objective function and gradient evaluation, respectively. In fig. 1c we again see comparable results in terms of the achieved local/global loss compared to the static GCP method. Note that OnlineGCP required a much larger history window ($H = 500$) than the other experiments to maintain consistent global loss over the entire streaming experiment. This, in conjunction with the observed slightly better local loss for the OnlineGCP method, indicates the method is somewhat over-solving each slice, suggesting the number of samples and/or temporal and factor matrix SGD iterations could be reduced.

ArXiv Abstracts. Kaggle provides a readily available collection of metadata from ArXiv papers that is updated regularly [7]. The metadata includes research categories (cs.LG:Machine Learning, etc.), date published, and full abstracts. We built a count tensor from these fields such that $\mathfrak{X}(i, j, k)$ corresponds to the number of abstracts with primary category i, that contain word j, and are k months from our start point. ArXiv was founded in 1991 but the earliest paper is back dated to 1986. We selected our start point as 120 months from this first paper once the volume of papers increased to a sufficient amount. We used Spacy's English core parser trained on over a million documents to preprocess all of the abstracts and extract unique words across the corpus [15]. Once the unique words were identified, we eliminated rarely occurring words by selecting words that occurred more than 100 times. The resulting tensor is of size $172 \times 24558 \times 300$ with approximately 2.4% sparsity. A warm-start for the first 20 months was generated via CP-APR. Decompositions were computed using OnlineGCP with Poisson loss (streamed one slice per month with 50,000 zero/nonzero samples for the objective and 10,000 zero/nonzero samples for the gradient), static GCP, and CP-APR. This data set demonstrates an example of concept drift where a large change in the factors is observed around month 160, causing a spike in local and global loss for the streaming method. The loss then slowly converges back to the loss obtained from the static methods as it gradually updates the factors. However, this then causes the global loss for the early time steps to degrade since they are not captured in the history window.

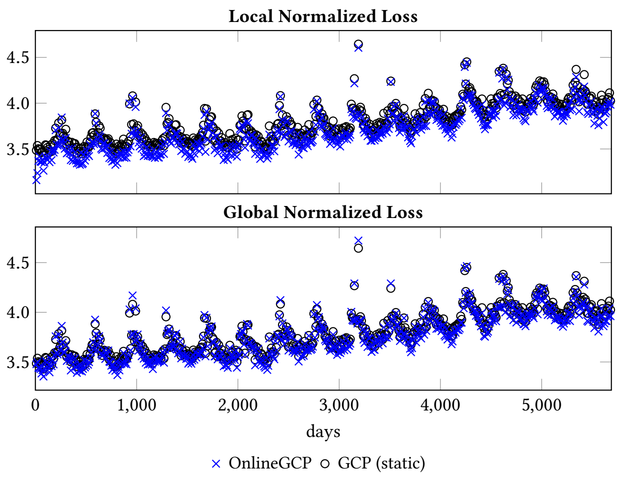
These experiments demonstrate the streaming GCP method is capable of producing decompositions with a similar level of accuracy as measured by the reconstruction loss as the static GCP method. Furthermore, in the synthetic experiments where the accuracy of the resulting K-tensor can be measured, it also achieves a similar score as the static GCP method (and most of the other methods considered). The approach uses significantly less memory, as it requires only storing a single tensor slice, and is applicable to the infinite streaming case where the number of time steps is unbounded. Furthermore, it is substantially more efficient when using the same total number of samples between streaming and static GCP, which is due to the significantly small number of SGD iterations required by the streaming method: the static GCP method uses 1000 iterations per epoch, with the number of epochs determined adaptively, whereas the streaming method uses in the range of 1–20 epochs with 100 iterations per epoch for each temporal and factor matrix solve, as shown in table 1. In fig. 2 we show the run time for each of the above synthetic and realistic data experiments, executed on a dual-socket Intel Xeon Platinum 8260 CPU with 24 cores per socket and an NVIDIA Volta V100 GPU. For the CPU experiments, we use



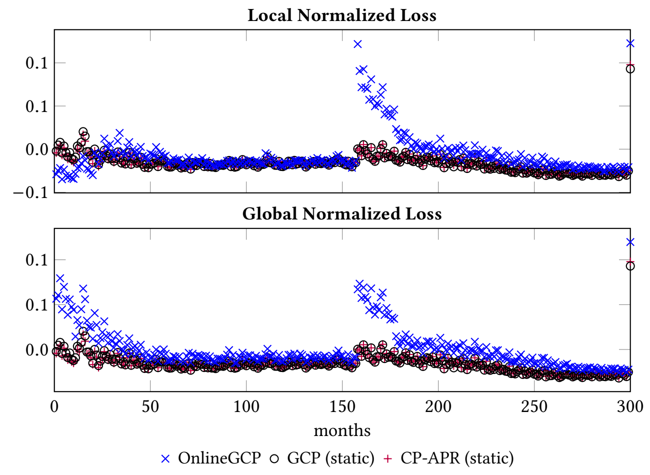
(a) Synthetic Gaussian



(b) Synthetic Poisson



(c) Chicago Crime (Bernoulli)



(d) ArXiv Abstracts (Poisson)

Figure 1: Results of the synthetic Gaussian and Poisson experiments as well as Chicago Crime with Bernoulli loss and ArXiv Abstracts with Poisson loss experiments showing comparable performance between OnlineGCP and appropriate methods including static GCP, Online SGD, OnlineCP, static CP-ALS and static CP-APR. Note, to make the Chicago Crime plot more legible, the maximum loss over 10 consecutive days is displayed.

the Kokkos OpenMP backend running with 48 threads, and for the GPU experiments we use the Kokkos CUDA backend. We see the OnlineGCP method provides comparable performance, and often substantially better performance, as the static GCP method while providing similar accuracy.

7 CONCLUSIONS

In this work, we developed a method called OnlineGCP for efficiently computing GCP decompositions of streaming tensor data. The method extends prior work in the literature on streaming CP decompositions to the GCP case allowing for arbitrary objective/loss functions defining the CP optimization problem. Similar to other streaming CP methods, the approach incrementally updates the temporal weights and CP model factors as each new tensor slice is observed without revisiting prior data. It includes a tunable history term to balance reconstruction of new and old tensor data, and employs stochastic gradient descent solvers enabling scalability to large, sparse tensors. The effectiveness of the approach was demonstrated on several synthetic and real datasets incorporating Gaussian, Poisson, and Bernoulli loss functions, where comparable losses were observed compared to other streaming and static methods appropriate for the chosen form of loss.

While the approach was shown to be effective and scalable, it relies on expert choice of numerous hyperparameters that can dramatically affect accuracy and computational cost. Unfortunately, our experience has shown these parameters must be empirically chosen on a case-by-case basis. The sensitivity of the method to these hyperparameters primarily derives from its use of stochastic

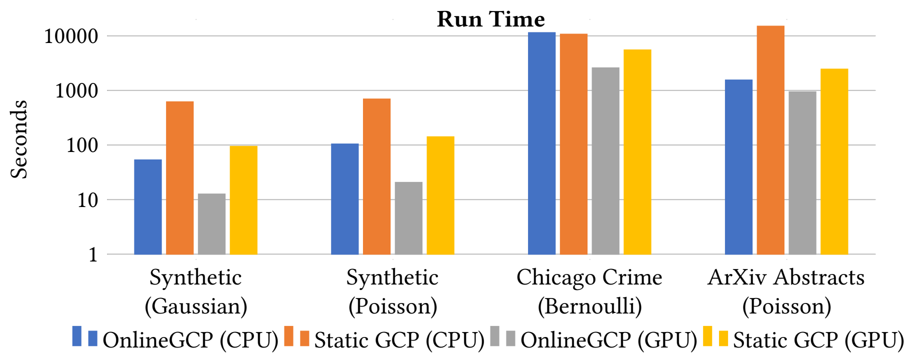


Figure 2: Running times in seconds of the streaming and static GCP methods showing comparable, and often substantially reduced, times on both CPU and GPU architectures for OnlineGCP compared to static GCP.

gradient descent as an optimization strategy, so future work will involve investigation of alternative solution strategies that rely on fewer hyperparameters and are more robust to their values.

REFERENCES

[1] Brett W. Bader and Tamara G. Kolda. 2007. Efficient MATLAB Computations with Sparse and Factored Tensors. *SIAM Journal on Scientific Computing* 30, 1 (Dec. 2007), 205–231. https://doi.org/10.1137/060676489

[2] J. A. Bazerque, G. Mateos, and G. B. Giannakis. 2013. Rank Regularization and Bayesian Inference for Tensor Completion and Extrapolation. *IEEE Transactions on Signal Processing* 61, 22 (2013), 5689–5703. https://doi.org/10.1109/TSP.2013.2278516

[3] J. D. Carroll and J. J. Chang. 1970. Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition. *Psychometrika* 35 (1970), 283–319. https://doi.org/10.1007/BF02310791

[4] Donghui Chen and Robert J. Plemmons. 2007. *Nonnegativity constraints in numerical analysis.* 109–139. https://doi.org/10.1142/9789812836267_0008

[5] Eric C. Chi and Tamara G. Kolda. 2012. On Tensors, Sparsity, and Nonnegative Factorizations. *SIAM J. Matrix Anal. Appl.* 33, 4 (Dec. 2012), 1272–1299. https://doi.org/10.1137/110859063

[6] City of Chicago. 2023. Chicago Crime Dataset. https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2

[7] Cornell University. 2023. arXiv Dataset. https://www.kaggle.com/Cornell-University/arxiv Accessed:2020-11-10.

[8] Yishuai Du, Yimin Zheng, Kuang chih Lee, and Shandian Zhe. 2018. Probabilistic Streaming Tensor Decomposition. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 99–108. https://doi.org/10.1109/icdm.2018.00025

[9] H. Carter Edwards, Daniel Sunderland, Vicki Porter, Chris Amsler, and Sam Mish. 2012. Manycore performance-portability: Kokkos multidimensional array library. *Scientific Programming* 20, 2 (2012), 89–114. https://doi.org/10.3233/SPR-2012-0343

[10] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. 2014. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *J. Parallel and Distrib. Comput.* 74 (2014), 3202–3216. Issue 12. https://doi.org/10.1016/j.jpdc.2014.07.003

[11] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 1–37.

[12] Ekta Gujral, Ravdeep Pasricha, and Evangelos E. Papalexakis. 2018. SamBaTen: Sampling-based Batch Incremental Tensor Decomposition. In *Proceedings of the 2018 SIAM International Conference on Data Mining*. 387–395. https://doi.org/10.1137/1.9781611975321.44

[13] Richard A. Harshman. 1970. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA working papers in phonetics* 16 (1970), 1–84.

[14] David Hong, Tamara G. Kolda, and Jed A. Duersch. 2020. Generalized Canonical Polyadic Tensor Decomposition. *SIAM Rev.* 62, 1 (2020), 133–163. https://doi.org/10.1137/18M1203626

[15] Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. (2017). To appear.

[16] Hiroyuki Kasai. 2016. Online Low-rank Tensor Subspace Tracking from Incomplete Data by CP Decomposition Using Recursive Least Squares. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. https://doi.org/10.1109/ICASSP.2016.7472131

[17] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. arXiv:1412.6980v9 [cs.LG] Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[18] Tamara G. Kolda and Brett W. Bader. 2009. Tensor Decompositions and Applications. *SIAM Rev.* 51, 3 (Sept. 2009), 455–500. https://doi.org/10.1137/07070111X

[19] Tamara G. Kolda and David Hong. 2020. Stochastic Gradients for Large-Scale Tensor Decomposition. *SIAM Journal on Mathematics of Data Science* 2, 4 (Oct. 2020), 1066–1095. https://doi.org/10.1137/19m1266265 arXiv:1906.01687

[20] Pierre-David Letourneau, Muthu Baskaran, Tom Henretty, James Ezick, and Richard Lethin. 2018. Computationally Efficient CP Tensor Decomposition Update Framework for Emerging Component Discovery in Streaming Data. In *HPEC'18 Proceedings*.

[21] Morteza Mardani, Gonzalo Mateos, and Georgios B. Giannakis. 2015. Subspace Learning and Imputation for Streaming Big Data Matrices and Tensors. *IEEE Transactions on Signal Processing* 63, 10 (may 2015), 2663–2677. https://doi.org/10.1109/tsp.2015.2417491

[22] D. Nion and N. D. Sidiropoulos. 2009. Adaptive Algorithms to Track the PARAFAC Decomposition of a Third-Order Tensor. *IEEE Transactions on Signal Processing* 57, 6 (June 2009), 2299–2310. https://doi.org/10.1109/TSP.2009.2016885

[23] Ravdeep Pasricha, Ekta Gujral, and Evangelos E. Papalexakis. 2019. Identifying and Alleviating Concept Drift in Streaming Tensor Decomposition. In *Machine Learning and Knowledge Discovery in Databases*. Springer International Publishing, 327–343. https://doi.org/10.1007/978-3-030-10928-8_20

[24] Eric T. Phipps et al. 2020. GenTen: Software for Generalized Canonical Polyadic Tensor Decompositions. Available online. https://gitlab.com/tensors/genten

[25] Eric T. Phipps and Tamara G. Kolda. 2019. Software for Sparse Tensor Decomposition on Emerging Computing Architectures. *SIAM Journal on Scientific Computing* 41, 3 (2019), C269–C290. https://doi.org/10.1137/18M1210691

[26] Shaden Smith, Jee W. Choi, Jiajia Li, Richard Vuduc, Jongsoo Park, Xing Liu, and George Karypis. 2017. *FROSTT: The Formidable Repository of Open Sparse Tensors and Tools.* http://frostt.io/

[27] Shaden Smith, Kejun Huang, Nicholas D. Sidiropoulos, and George Karypis. 2018. Streaming Tensor Factorization for Infinite Data Sources. In *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 81–89. https://doi.org/10.1137/1.9781611975321.10

[28] Qingquan Song, Xiao Huang, Hancheng Ge, James Caverlee, and Xia Hu. 2017. Multi-Aspect Streaming Tensor Completion. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. https://doi.org/10.1145/3097983.3098007

[29] Giorgio Tomasi and Rasmus Bro. 2006. A comparison of algorithms for fitting the PARAFAC model. *Computational Statistics & Data Analysis* 50, 7 (2006), 1700–1734. https://doi.org/10.1016/j.csda.2004.11.013

[30] Jeffrey S. Vitter. 1985. Random Sampling with a Reservoir. *ACM Trans. Math. Softw.* 11, 1 (March 1985), 37–57. https://doi.org/10.1145/3147.3165

[31] Zheng Zhang and Cole Hawkins. 2018. Variational Bayesian Inference for Robust Streaming Tensor Factorization and Completion. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1446–1451. https://doi.org/10.1109/ICDM.2018.00200

[32] Shuo Zhou, Nguyen Xuan Vinh, James Bailey, Yunzhe Jia, and Ian Davidson. 2016. Accelerating Online CP Decompositions for Higher Order Tensors. In *KDD'16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. https://doi.org/10.1145/2939672.2939763