SAND2023-12776C

Sandia National Laboratories

Incremental Update Techniques for Online Analysis of Streaming Tensor Data

Eric Phipps, Saibal De, Hemanth Kolla (Sandia National Laboratories), Nick Johnson (Cerebras Systems), Tammy Kolda (MathSci.ai), Zitong Li (UC-Irvine)

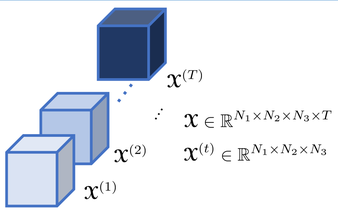
Tensors, or multidimensional arrays of data, are ubiquitous in data science applications. Analysis of tensor data is often facilitated through tensor decompositions, akin to matrix factorizations for two-dimensional data, and are higher dimensional analogs of techniques such as SVD, PCA, and POD.

In many cases, tensor data has a streaming character where data is gradually observed over time. In some cases the data stream may never end (i.e., infinite streaming) or maybe too large to fit in memory. Thus approaches that incrementally update tensor decompositions are needed.

Common tensor decomposition approaches include Canonical Polyadic (CP) and Tucker. In this work, we consider streaming CP decompositions for sparse tensors and streaming Tucker decompositions for dense tensors.

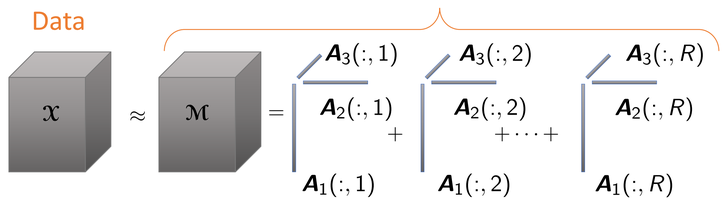
Streaming Tensor Setup

- Updates are processed in discrete batches indexed by time t = 1,2, …
- At each time t, a d-way tensor is observed
- All tensor dimensions are fixed throughout time
- By stacking observed tensors along a new time mode, a $d+1$-way tensor is created with an ever growing time mode



Streaming Generalized CP Decompositions[1]

CP decompositions discover important relationships in data, and are useful for unsupervised tasks such as pattern identification and anomaly detection.



$$\mathcal{X}(i_1, i_2, \ldots, i_d) \approx \mathcal{M}(i_1, i_2, \ldots, i_d) = \sum_{j=1}^{R} \boldsymbol{A}_1(i_1, j)\boldsymbol{A}_2(i_2, j) \ldots \boldsymbol{A}_d(i_d, j)$$

Generalized CP (GCP)[2] allows for flexible use of data fit functions in defining optimization problem for a broader range of statistical models:

$$\min \quad F(\mathcal{X}, \mathcal{M}) = \sum_i f(\mathcal{X}_i, \mathcal{M}_i)$$
$$\text{s.t.} \quad \mathcal{M} = [\![\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots \boldsymbol{A}_d]\!]$$

Example Loss Functions

Normal ($x, m \in \mathbb{R}$)	$f(x,m) = (x-m)^2$
Poisson ($x \in \mathbb{N}, m > 0$)	$f(x,m) = m - x \log m$
Bernoulli ($x \in \{0,1\}, m > 0$)	$f(x,m) = \log(m+1) - x \log m$
β-divergence ($x > 0, m > 0, \beta = \frac{1}{2}$)	$f(x,m) = x/\sqrt{m} + \sqrt{m}$

Minimization problem solved via derivative-based optimization. Cost of gradient computation grows exponentially with tensor dimensions, so solve using stochastic gradient descent (SGD)[3].

Streaming GCP problem solved after receiving each streamed tensor $\mathcal{Y}$:

GCP loss for tensor $\mathcal{Y}$ — History regularization term

$$\min_{\mathcal{M}_t} \sum_i f(\mathcal{Y}_i, \mathcal{M}_i^{(t)}) + \frac{1}{2} \sum_{h \in \mathcal{H}_t} w_h \|\tilde{\mathcal{M}}^{(h)} - \mathcal{M}^{(h)}\|_F^2$$

Temporal history window

$$\text{s.t.} \quad \mathcal{M}^{(t)} = [\![\boldsymbol{A}_1, \ldots, \boldsymbol{A}_d, \boldsymbol{A}_{d+1}(t,:)]\!]$$

CP Model for time step t

$$\mathcal{M}^{(h)} = [\![\boldsymbol{A}_1, \ldots, \boldsymbol{A}_d, \boldsymbol{A}_{d+1}(h,:)]\!]$$ ← CP Model using current factors and historical time step h

$$\tilde{\mathcal{M}}^{(h)} = [\![\tilde{\boldsymbol{A}}_1, \ldots, \tilde{\boldsymbol{A}}_d, \boldsymbol{A}_{d+1}(h,:)]\!]$$ ← CP Model using prior factors and historical time step h

Problem solved using two-step optimization process (each step using SGD):

- Solve for new row of temporal factor holding non-temporal factors fixed (old rows are not updated)
- Solve for updated non-temporal factors holding temporal factor fixed

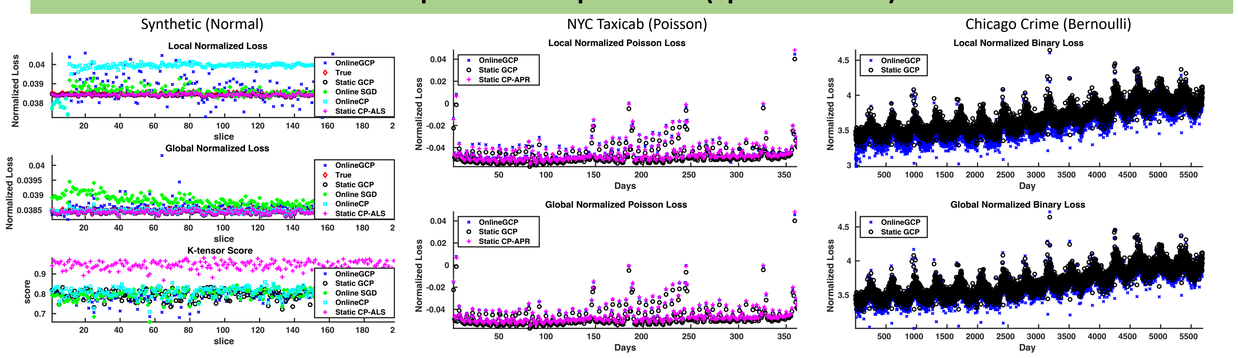
Leverage ADAM SGD optimizer in both steps:

- Temporal factors assumed to change significantly over time, so reset ADAM moments each time step
- Non-temporal factors assumed to change slowly, so don't reset ADAM moments after each time step

Approach implemented in GenTen[4], for parallel GCP decompositions targeting HPC architectures

- Shared memory parallelism (CPU, GPU) via Kokkos[5]
- Distributed memory parallelism via MPI

Computational Experiments (Sparse Tensors)

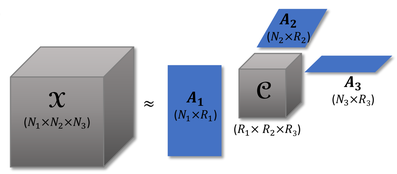


Challenges & Approach

- Data stream potentially unbounded, so must compute decompositions without revisiting sufficiently old data
- Compute a batch decomposition on initial data
- Update factorization after each new tensor arrives
- Balance reconstruction of old data with new

Streaming Tucker Decompositions[6]

Tucker decompositions capture high-variance subspaces and are useful for surrogate modeling and data compression.



$$\mathcal{X} \approx \mathcal{M} = \mathcal{C} \times_1 \boldsymbol{A}_1 \times_2 \boldsymbol{A}_2 \cdots \times_d \boldsymbol{A}_d$$
$$\iff \boldsymbol{X}_{(n)} \approx \boldsymbol{A}_n \boldsymbol{C}_{(n)} (\boldsymbol{A}_d \otimes \cdots \otimes \boldsymbol{A}_{n+1} \otimes \boldsymbol{A}_{n-1} \otimes \cdots \otimes \boldsymbol{A}_1)^T$$
$$\mathcal{Z} = \mathcal{X} \times_n \boldsymbol{A} \iff \boldsymbol{Z}_{(n)} = \boldsymbol{A}\boldsymbol{X}_{(n)}$$

n-mode product — Matricization along mode n

Tucker decompositions often computed through sequence of SVD/eigenvector calculations:

Algorithm 1 Sequentially truncated higher order SVD (ST-HOSVD)

Require: Data tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_d}$, tolerance $\tau > 0$
Ensure: Core $\mathcal{C} \in \mathbb{R}^{R_1 \times \cdots \times R_d}$, bases $\mathbf{A}_k \in \mathbb{R}^{N_k \times R_k}$ for $1 \le k \le d$
Compute truncation: $\delta \leftarrow \tau \|\mathcal{X}\|_F / \sqrt{d}$
Initialize core: $\mathcal{C} \leftarrow \mathcal{X}$
for $k = 1$ to d **do**
  Compute Gram matrix: $\mathbf{G} \leftarrow \mathbf{C}_{(k)} \mathbf{C}_{(k)}^\top$
  Eigendecompose: $\boldsymbol{\lambda}, \mathbf{A} \leftarrow \text{eig}(\mathbf{G})$
  Determine rank: $R_k \leftarrow \min_\ell \{\ell : \lambda_{\ell+1} + \cdots + \lambda_{N_k} \le \delta^2\}$
  Truncate: $\mathbf{A}_k \leftarrow \mathbf{A}(:, 1:R_k)$
  Update core: $\mathcal{C} \leftarrow \mathcal{C} \times_k \mathbf{A}_k^\top$
end for

Challenges for streaming Tucker decompositions:

- Update factor matrices and core tensor without revisiting prior data
- Allow Tucker ranks to evolve to ensure truncation satisfies prescribed error tolerance:

$$\|\mathcal{X} - \mathcal{M}\|_F \le \tau \|\mathcal{X}\|_F$$

Outline of streaming ST-HOSVD algorithm implemented in TuckerMPI[7]:

- Assume decomposition $\mathcal{X} \approx \mathcal{C} \times_1 \boldsymbol{A}_1 \cdots \times_d \boldsymbol{A}_d \times_{d+1} \boldsymbol{A}_{d+1}$ has been computed and new slice $\mathcal{Y}$ is observed
- Sequentially update factor matrices for non-streaming modes, e.g., for mode-1:

$$\boldsymbol{X}_{(1)} \approx \boldsymbol{A}_1 \boldsymbol{C}_{1,(1)}, \quad \boldsymbol{P} \equiv \boldsymbol{A}_1^T \boldsymbol{Y}_{(1)}, \quad \boldsymbol{E} \equiv \boldsymbol{Y}_{(1)} - \boldsymbol{A}_1 \boldsymbol{P} \approx \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T \implies \tilde{\boldsymbol{X}}_{(1)} = \begin{bmatrix} \boldsymbol{X}_{(1)} & \boldsymbol{Y}_{(1)} \end{bmatrix} \approx \tilde{\boldsymbol{A}}_1 \tilde{\boldsymbol{C}}_{1,(1)} = \begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{U} \end{bmatrix} \begin{bmatrix} \boldsymbol{C}_{1,(1)} & \boldsymbol{P} \\ & \boldsymbol{U}^T\boldsymbol{E} \end{bmatrix}$$

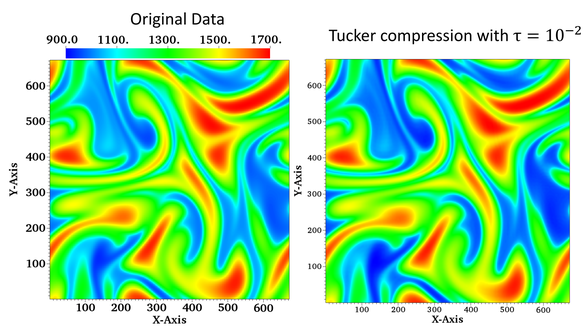
Prior partial core — Updated factor matrix — Updated partial core

- Can show updated core obtained by row/column augmentation of prior (full) core (so no need to save prior cores)
- Choose $\boldsymbol{U}$ so that $\|\boldsymbol{E} - \boldsymbol{U}\boldsymbol{U}^T\boldsymbol{E}\|_F \le \tau \|\mathcal{Y}\|_F / \sqrt{d}$
- For streaming mode, compute updated SVD of $\tilde{\boldsymbol{X}}_{(d+1)} = \begin{bmatrix} \boldsymbol{X}_{(d+1)} \\ \boldsymbol{y}^T \end{bmatrix}$, $\boldsymbol{y} = \text{vec}(\mathcal{Y})$, given SVD $\boldsymbol{X}_{(d+1)} \approx \boldsymbol{A}_d \boldsymbol{\Sigma} \boldsymbol{W}^T$
- This can be done, without accessing $\boldsymbol{X}_{(d+1)}$, using *incremental-SVD*[8] to obtain $\tilde{\boldsymbol{A}}_{d+1}$
- Updated core:

$$\begin{bmatrix} \boldsymbol{X}_{(d+1)} \\ \boldsymbol{y}^T \end{bmatrix} \approx \begin{bmatrix} \boldsymbol{A}^{(d+1)} \boldsymbol{C}_{(d+1)} (\boldsymbol{A}_d \otimes \cdots \otimes \boldsymbol{A}_1)^T \\ \boldsymbol{d}^T (\boldsymbol{A}_d \otimes \cdots \otimes \boldsymbol{A}_1)^T \end{bmatrix} = \tilde{\boldsymbol{A}}_{d+1} \tilde{\boldsymbol{C}}_{(d+1)} (\boldsymbol{A}_d \otimes \cdots \otimes \boldsymbol{A}_1)^T \implies \tilde{\boldsymbol{C}}_{(d+1)} = \tilde{\boldsymbol{A}}_{d+1}^T \begin{bmatrix} \boldsymbol{A}_{d+1} \boldsymbol{C}_{(d+1)} \\ \boldsymbol{d}^T \end{bmatrix}$$

Computational Experiments (Dense Tensors)

Compression of Homogeneous Charge Compression Ignition (HCCI) simulation data[9] provided by S3D DNS code



Comparison of batch and streaming algorithm performance in TuckerMPI (1 MPI rank)

Tolerance (τ)	Algorithm	Rank	Memory	Time	Relative Error
10^{-1}	Batch	$(35, 27, 8, 7)$	33.68	20.91	9.6×10^{-2}
	Streaming (110)	$(46, 37, 9, 7)$	14.06	20.59	9.4×10^{-2}
	Streaming (130)	$(44, 36, 9, 8)$	16.57	19.55	9.6×10^{-2}
	Streaming (150)	$(43, 34, 9, 9)$	19.09	21.23	1.0×10^{-1}
10^{-2}	Batch	$(122, 110, 21, 29)$	38.70	22.65	9.6×10^{-3}
	Streaming (110)	$(139, 126, 24, 28)$	16.34	29.24	9.6×10^{-3}
	Streaming (130)	$(137, 124, 23, 27)$	19.24	28.93	9.9×10^{-3}
	Streaming (150)	$(134, 121, 23, 28)$	22.09	28.15	9.6×10^{-3}
10^{-3}	Batch	$(231, 213, 29, 67)$	46.42	24.43	9.5×10^{-4}
	Streaming (110)	$(249, 230, 30, 96)$	19.64	65.65	9.8×10^{-4}
	Streaming (130)	$(248, 227, 30, 77)$	23.15	56.51	9.9×10^{-4}
	Streaming (150)	$(245, 225, 30, 64)$	26.59	50.90	9.8×10^{-4}

1. E. Phipps, N. Johnson, T. Kolda, "Streaming Generalized Canonical Polyadic Tensor Decompositions," accepted to PASC'23, https://arxiv.org/abs/2110.14514.
2. D. Hong, T. Kolda, J.Duersch, "Generalized Canonical Polyadic Tensor Decomposition," SIREV, 2020, https://doi.org/10.1137/18M1203626.
3. T. Kolda, D. Hong, "Stochastic Gradients for Large-Scale Tensor Decomposition," SIMODS, 2020, https://doi.org/10.1137/19M1266265.
4. GenTen software: https://gitlab.com/tensors/genten
5. Kokkos software: https://github.com/kokkos/kokkos
6. S. De, H. Kolla, Z. Li, E. Phipps "Efficient Computation of Tucker Decomposition for Streaming Scientific Data Compression," submitted to ISC'23.
7. TuckerMPI software: https://gitlab.com/tensors/TuckerMPI
8. M. Brand, "Incremental Singular value Decomposition of Uncertain Data with Missing Values," Euro. Conf. on Comp. Vision, 2002, https://doi.org/10.1007/3-540-47969-4_47.
9. H. Kolla, K. Aditya, J.H. Chen, "Higher Order Tensors for DNS Data Analysis and Compression. In: Pitsch, H., Attili, A. (eds) Data Analysis for Direct Numerical Simulations of Turbulent Combustion. Springer, 2020, https://doi.org/10.1007/978-3-030-44718-2_6.

U.S. DEPARTMENT OF ENERGY — NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
SAND

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.