Streaming Tensor Factorization for Infinite Data Sources

Shaden Smith - Intel Parallel Computing Lab

Kejun Huang - University of Minnesota

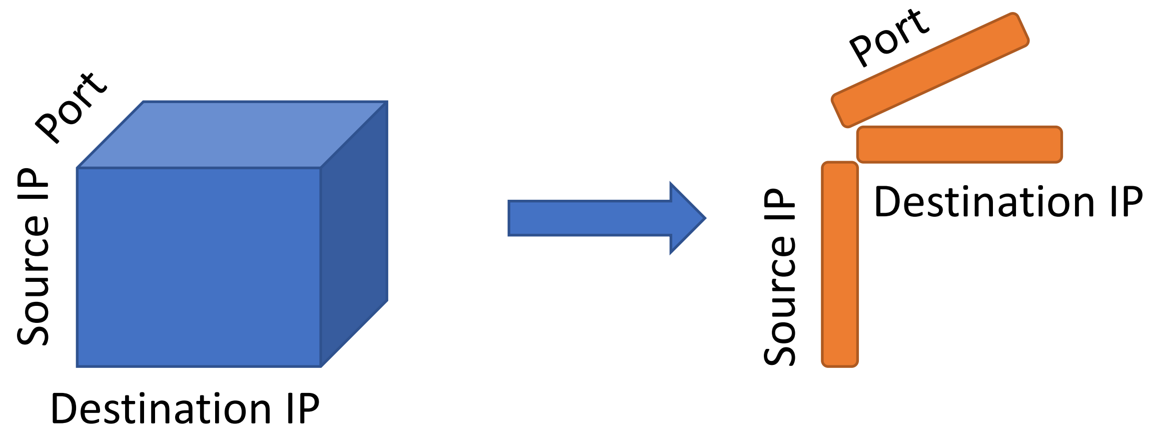
Nicholas D. Sidiropoulos - University of Virginia

George Karypis - University of Minnesota

Shaden.Smith@intel.com

Tensor factorization

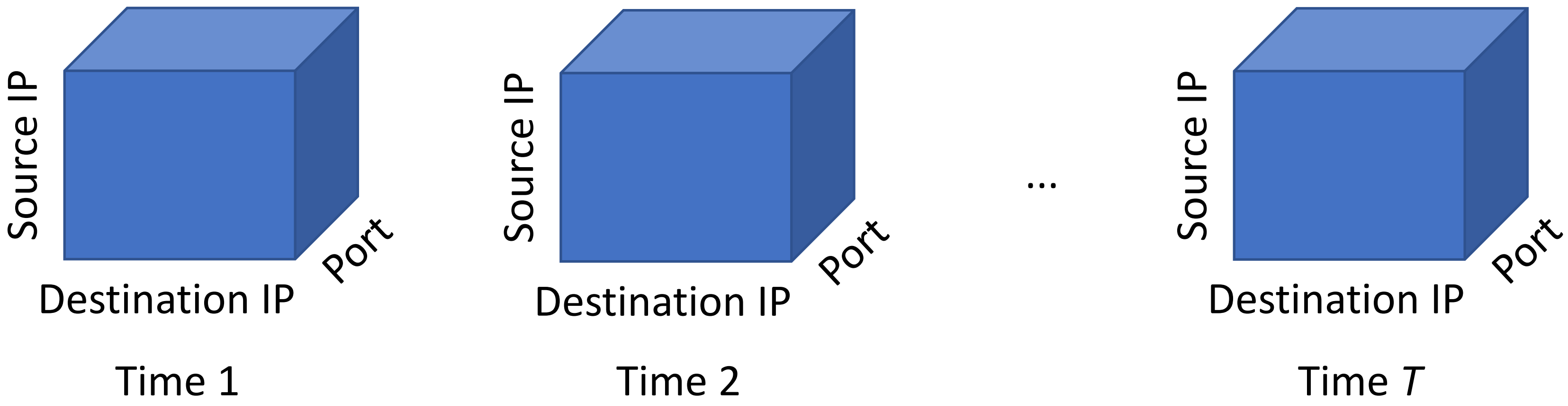
- Multi-way data can be naturally represented as a *tensor*.
- Tensor factorizations are powerful tools for facilitating the analysis of multi-way data.
  - Think: singular value decomposition, principal component analysis.



Canonical Polyadic Decomposition

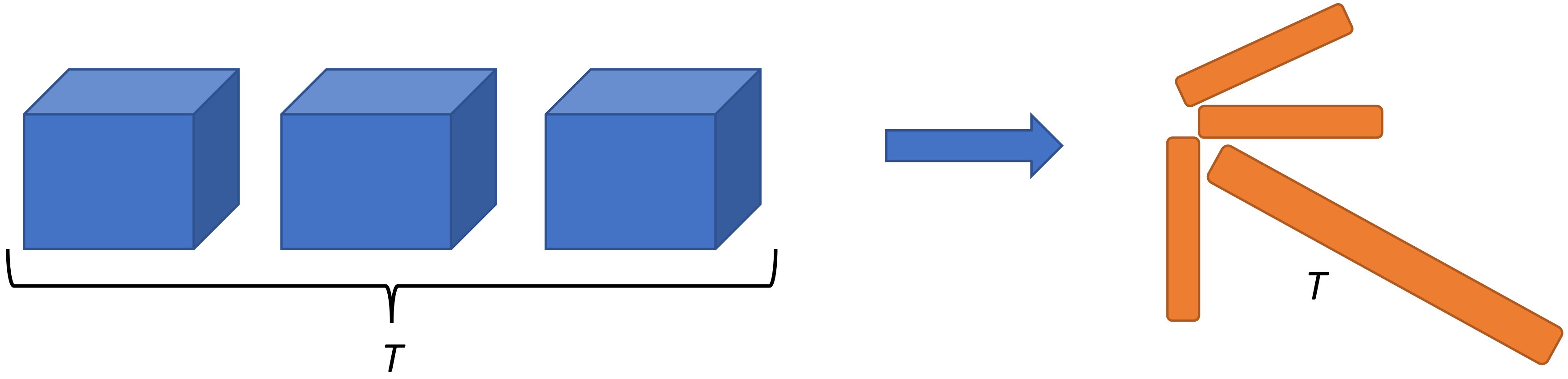
Streaming data

- We often need to analyze multi-way data that is *streamed*.
  - Applications include: cybersecurity, discussion tracking, traffic analysis, video monitoring, …
- A batch of data arrives each timestep 1, …, *T*.
  - ***T* may be infinite!**
- Batches are assumed to come from the same generative model.
  - In practice, we must account for the model slowly changing over time.



Streaming tensor factorization

- The collection of *N*-dimensional tensors can be viewed as an *(N+1)*-dimensional tensor observed over time.
- We want to cheaply *update* an existing factorization each timestep to incorporate the latest batch of data.
- **Challenge:** storing historical tensor *or* factorization data that grows with time is infeasible.
- **Challenge:** we would like to apply constraints such as non-negativity to the factorization.



CP-stream: optimization problem

- We start from the following non-convex optimization problem over all timesteps:

$$\begin{array}{ll} \underset{\{\boldsymbol{A}^{(n)} \in \mathbb{R}^{I_n \times K}\}, \{\boldsymbol{s}_t \in \mathbb{R}^K\}}{\text{minimize}} & \frac{1}{2} \sum_{t=1}^{T} \left(\left\| \boldsymbol{\mathcal{X}}_t - [\![\{\boldsymbol{A}^{(n)}\}; \boldsymbol{s}_t]\!] \right\|^2 + \lambda \|\boldsymbol{s}_t\|^2 \right) \\ \text{subject to} & \boldsymbol{A}^{(n)} \in \mathcal{C}, \end{array}$$

- We constrain the factor matrices to have column norms ≤ 1.
  - This improves stability due to a scaling ambiguity in the CPD.
- The $\boldsymbol{s}_t \in \mathbb{R}^K$ vectors form the rows of $\boldsymbol{S}$, the temporal factor matrix.

CP-stream: formulation

- To avoid storing historic tensor data, we follow (Vandecappelle et al. 2017) and instead use the historical factorization:

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

- μ is a *forgetting factor* used to down-weight the importance of older data.
- **Limitation:** this still requires $\boldsymbol{S} \in \mathbb{R}^{T \times K}$.

CP-stream: algorithm (details in paper/poster)

When a new batch of data arrives at time t:

1. Compute $\boldsymbol{s}_t$. This has a closed-form solution involving the new batch of tensor data and the previous factor matrices.
    - Complexity does not depend on *T.*
2. Update the factor matrices. We use alternating optimization with ADMM (AO-ADMM; Huang & Sidiropoulos 2016).
    - The temporal factor $\boldsymbol{S}$ is only used in its **compact** Gramian form $\boldsymbol{S}^T\boldsymbol{S}$, which is computed recursively:

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

Extensions

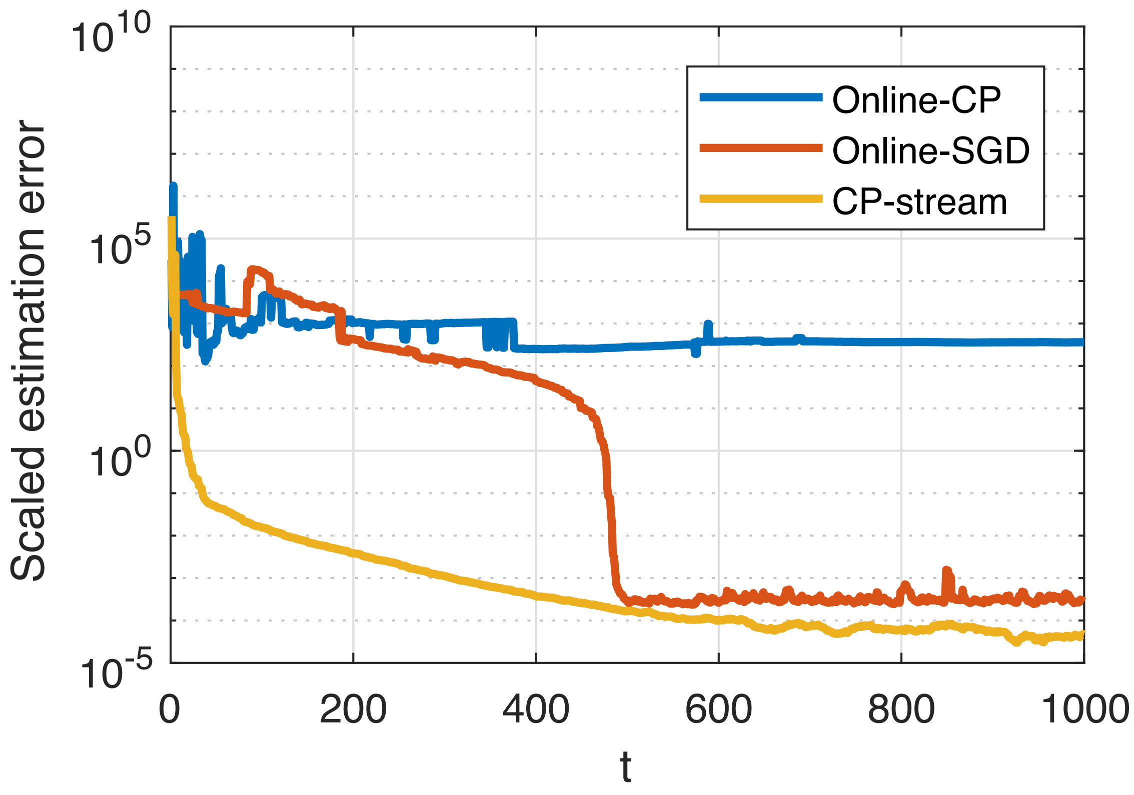
- CP-stream supports additional constraints/regularizations. For stability, they are combined with the column norm constraint (*proof of convergence in paper*).
  - Non-negativity
  - ℓ_1 regularization to promote sparse factors

- Tensor sparsity:
  - CP-stream scales linearly in the number of non-zeros and makes use of the existing optimized kernels.
  - Sparsity is not treated as *missing*, because absence of activity also carries meaning in our applications.

Evaluation

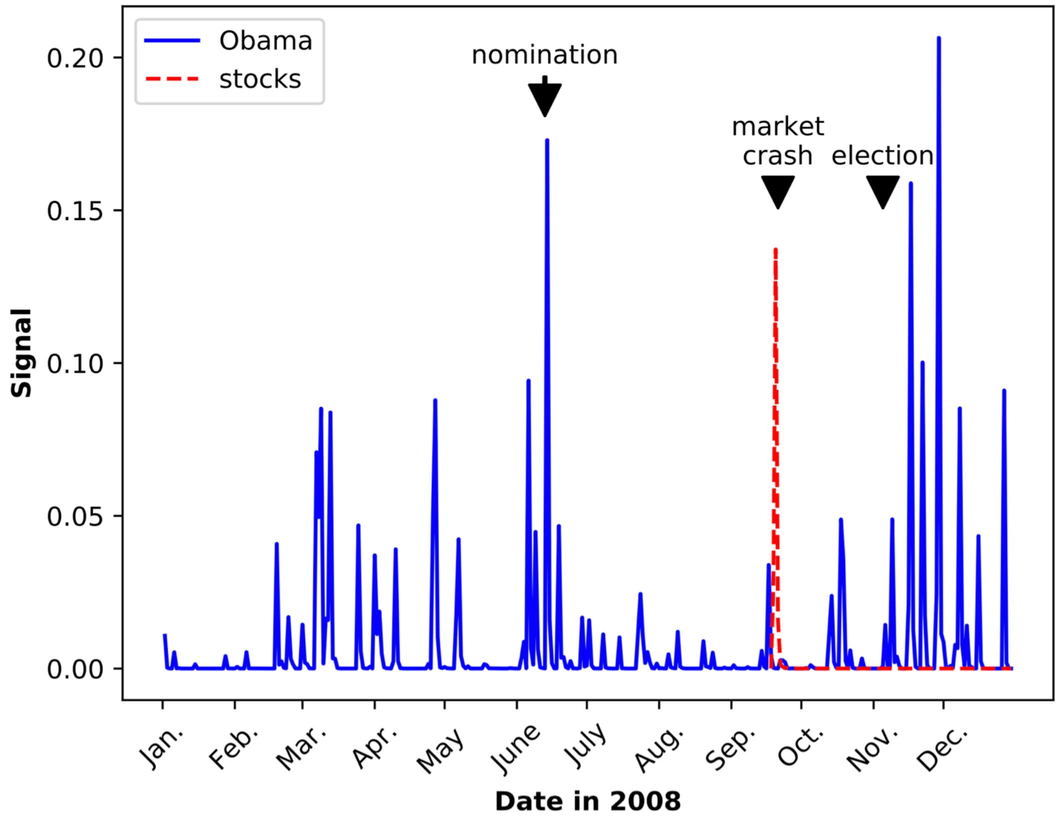
- We generated a dense 100x100x1000 tensor from rank-10 factors (plus noise).
- We compare against:
  - Online-CP (Zhou et al., 2016)
  - Online-SGD (Mardani et al., 2015)
- Shown is the estimation error of the known ground-truth factors:

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



Case study: discussion tracking

- Comments on reddit.com form a (*user, community, word)* tensor.
  - A new batch arrives each day.
  - 65M non-zeros over one year.
- Each user, community, and word are represented by a low-rank vector in the factorization.
- Tracking the vectors representing the word "Obama" and the stocks community reveals events in 2008.



Wrapping up

- Streaming tensor factorization has applications in areas such as cybersecurity, discussion tracking, and traffic analysis.
- CP-stream uses a formulation suitable for long-term streaming, and supports sparsity and constraints.
- Our source code is to be open sourced as part of SPLATT
  - https://github.com/ShadenSmith/splatt
- Sparse tensor datasets available in FROSTT:
  - http://frostt.io/
- Contact: Shaden.Smith@intel.com or shaden@cs.umn.edu

Backup

AO-ADMM

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

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

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

and

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

Algorithm 1 CP-stream

Require: $\boldsymbol{\mathcal{X}}_1, \boldsymbol{\mathcal{X}}_2, ..., \boldsymbol{\mathcal{X}}_T$; forgetting factor μ

```
initialize A_0^(1), ..., A_0^(N)
G_0 ← 0
for t = 1, ..., T do
    s_t ← least-squares update (4.3)
    repeat
        for n = 1, ..., N do
            construct Φ^(n) and Ψ^(n) per (4.5) and (4.6)
            ρ = tr(Φ^(n))/K
            A_t^(n) ← ADMM iterates (4.7)
        end for
    until convergence
    G_t = μG_{t-1} + s_t s_t^T
end for
```

AO-ADMM (2)

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

Algorithm 1 CP-stream

Require: $\boldsymbol{\mathcal{X}}_1, \boldsymbol{\mathcal{X}}_2, ..., \boldsymbol{\mathcal{X}}_T$; forgetting factor μ

initialize $\boldsymbol{A}_0^{(1)}, ..., \boldsymbol{A}_0^{(N)}$
$\boldsymbol{G}_0 \leftarrow \mathbf{0}$
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)
      $\rho = \mathrm{tr}(\boldsymbol{\Phi}^{(n)})/K$
      $\boldsymbol{A}_t^{(n)} \leftarrow$ 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