Accelerated Stochastic Gradient for Nonnegative Tensor Completion and Parallel Implementation

Ioanna Siaminou
School of Electrical and Computer Engineering
Technical University of Crete
Chania, Greece
isiaminou@isc.tuc.gr

Ioannis Marios Papagiannakos
School of Electrical and Computer Engineering
Technical University of Crete
Chania, Greece
ipapagiannakos@isc.tuc.gr

Christos Kolomvakis
School of Electrical and Computer Engineering
Technical University of Crete
Chania, Greece
ckolomvakis@isc.tuc.gr

Athanasios P. Liavas
School of Electrical and Computer Engineering
Technical University of Crete
Chania, Greece
liavas@telecom.tuc.gr

Abstract—We consider the problem of nonnegative tensor completion. We adopt the alternating optimization framework and solve each nonnegative matrix completion problem via a stochastic variation of the accelerated gradient algorithm. We experimentally test the effectiveness and the efficiency of our algorithm using both real-world and synthetic data. We develop a shared-memory implementation of our algorithm using the multi-threaded API OpenMP, which attains significant speedup. We believe that our approach is a very competitive candidate for the solution of very large nonnegative tensor completion problems.

Index Terms—tensors, stochastic gradient, nonnegative tensor completion, optimal first-order optimization algorithms, parallel algorithms, OpenMP.

I. INTRODUCTION

Tensors have recently gained great popularity due to their ability to model multiway data dependencies [1], [2], [3], [4]. Tensor decomposition (TD) into latent factors is very important for numerous tasks, such as feature selection, dimensionality reduction, compression, data visualization and interpretation. The Canonical Polyadic Decomposition (CPD) is one of the most important tensor decomposition models. Tensor Completion (TC) arises in many modern applications such as machine learning, signal processing, and scientific computing.

We focus on the CPD model and consider the nonnegative tensor completion (NTC) problem, using as quality metric the Frobenius norm of the difference between the true and the estimated tensor. We adopt the Alternating Optimization (AO) framework, that is, we work in a circular manner and update each factor by keeping all other factors fixed. We update each factor by solving a nonnegative matrix completion (NMC)

problem via a stochastic variant of the accelerated (Nesterov-type) gradient [5].

Recent tensor applications, such as social network analysis, recommendation systems, and targeted advertising, need to handle very large sparse tensors. We propose a shared-memory implementation of our algorithm that attains significant speedup and can efficiently handle very large problems.

A. Related Work

Most of the papers that consider sparse TD and TC focus on unconstrained problems. One of the earliest works is Gigatensor [6], which was followed by DFacTo [7]. In [8], two parallel algorithms for the unconstrained TC have been developed and results concerning the speedup attained by their MPI implementations on a linear processor array have been reported. In [9], the authors introduce fine- and medium-grained partitionings for the TC problem, while [10] incorporates dimension trees into the developed parallel algorithms. In [11] and [12], the AO-ADMM framework has been adopted for constrained matrix/tensor factorization and completion. In [13], the medium-grained approach of [9] was used for the solution of the nonnegative TD problem in distributed memory systems. The same problem has been considered in [14], where the authors incorporate the dimension trees and observe performance gains due to reduced computational load. The works in [15], [16], and [17] make use of either the Map-Reduce programming model or the Spark engine. In [18], a hypergraph model for general medium–grain partitioning has been presented.

Works that employ Stochastic Gradient Descent (SGD) on shared memory and distributed systems for sparse tensor factorization and completion include [19], [20], [21], [22]. In [19], the authors describe a TC approach which uses the CPD model and employs a proximal SGD algorithm, that can be implemented in a distributed environment. In [20], the authors examine three popular optimization algorithms:

alternating least squares (ALS), SGD, and coordinate descent (CCD++), implemented on shared- and distributed-memory systems. They conclude that SGD is most competitive in a serial environment, ALS is recommended for shared-memory systems, and both ALS and CCD++ are competitive on distributed systems. In [21], the authors propose a GPU-accelerated parallel TC scheme (GPU-TC) for accurate and fast recovery of missing data via SGD. Finally, [22] presents GentenMPl, a toolkit for sparse CPD that is designed to run effectively on distributed-memory high-performance computers. The authors use the Trilinos libraries and they present implementations of the CPD-ALS and an SGD method.

B. Notation

Vectors, matrices, and tensors are denoted by small, capital, and calligraphic capital letters, respectively; for example, $\mathbf{x}$, $\mathbf{X}$, and $\boldsymbol{\mathcal{X}}$. $\mathbb{R}_+^{I_1 \times \cdots \times I_N}$ denotes the set of $(I_1 \times \cdots \times I_N)$ nonnegative tensors. The elements of tensor $\boldsymbol{\mathcal{X}}$ are denoted as $\boldsymbol{\mathcal{X}}(i_1, \ldots, i_N)$. In many cases, we use Matlab-like notation, for example, $\mathbf{A}(j, :)$ denotes the j-th row of matrix $\mathbf{A}$. The outer product of vectors $\mathbf{a}$ and $\mathbf{b}$ is defined as $\mathbf{a} \circ \mathbf{b}$. The Kronecker, Khatri-Rao, and Hadamard product of matrices $\mathbf{A}$ and $\mathbf{B}$, of compatible dimensions, are defined, respectively, as $\mathbf{A} \otimes \mathbf{B}$, $\mathbf{A} \odot \mathbf{B}$ and $\mathbf{A} \circledast \mathbf{B}$; extensions to the cases with more than two arguments are obvious. $\mathbf{I}_P$ denotes the $(P \times P)$ identity matrix, $\| \cdot \|_F$ denotes the Frobenius norm of the matrix or tensor argument, and $(\mathbf{X})_+$ denotes the matrix derived after the projection of the elements of $\mathbf{X}$ onto $\mathbb{R}_+$.

II. NONNEGATIVE TENSOR COMPLETION

Let $\boldsymbol{\mathcal{X}}^o \in \mathbb{R}_+^{I_1 \times \cdots \times I_N}$ be an N-th order tensor which admits the rank-R CPD [3], [4]

$$\boldsymbol{\mathcal{X}}^o = [\![\mathbf{U}^{o(1)}, \ldots, \mathbf{U}^{o(N)}]\!] = \sum_{r=1}^{R} \mathbf{u}_r^{o(1)} \circ \cdots \circ \mathbf{u}_r^{o(N)}, \quad (1)$$

where $\mathbf{U}^{o(i)} = [\mathbf{u}_1^{o(i)} \ \cdots \ \mathbf{u}_R^{o(i)}] \in \mathbb{R}_+^{I_i \times R}$, for $i = 1, \ldots, N$. We observe $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}}^o + \boldsymbol{\mathcal{E}}$, where $\boldsymbol{\mathcal{E}}$ is additive noise. Let $\Omega \subseteq \{1, \ldots, I_1\} \times \cdots \times \{1, \ldots, I_N\}$ be the set of indices of the observed entries of $\boldsymbol{\mathcal{X}}$. Also, let $\boldsymbol{\mathcal{M}}$ be a tensor with the same size as $\boldsymbol{\mathcal{X}}$, with elements $\boldsymbol{\mathcal{M}}(i_1, i_2, \ldots, i_N)$ equal to one or zero based on the availability of the corresponding element of $\boldsymbol{\mathcal{X}}$. That is

$$\boldsymbol{\mathcal{M}}(i_1, i_2, \ldots, i_N) = \begin{cases} 1, & \text{if } (i_1, i_2, \ldots, i_N) \in \Omega, \\ 0, & \text{otherwise}. \end{cases} \quad (2)$$

We consider the NTC problem

$$\min_{\left\{ \mathbf{U}^{(i)} \in \mathbb{R}_+^{I_i \times R} \right\}_{i=1}^{N}} f_\Omega \left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)} \right) + \frac{\lambda}{2} \sum_{i=1}^{N} \left\| \mathbf{U}^{(i)} \right\|_F^2, \quad (3)$$

where

$$f_\Omega \left(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)} \right) = \frac{1}{2} \left\| \boldsymbol{\mathcal{M}} \circledast \left(\boldsymbol{\mathcal{X}} - [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!] \right) \right\|_F^2.$$

If $\boldsymbol{\mathcal{Y}} = [\![\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]\!]$, then, for an arbitrary mode i, the corresponding matrix unfolding is given by

$$\mathbf{Y}_{(i)} = \mathbf{U}^{(i)} \left(\mathbf{U}^{(N)} \odot \cdots \odot \mathbf{U}^{(i+1)} \odot \mathbf{U}^{(i-1)} \odot \cdots \odot \mathbf{U}^{(1)} \right)^T. \quad (4)$$

Thus, for $i = 1, \ldots, N$, f_Ω can be expressed as

$$f_\Omega(\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}) = \frac{1}{2} \left\| \mathbf{M}_{(i)} \circledast \left(\mathbf{X}_{(i)} - \mathbf{Y}_{(i)} \right) \right\|_F^2, \quad (5)$$

where $\mathbf{M}_{(i)}$, and $\mathbf{X}_{(i)}$ are the matrix unfoldings of $\boldsymbol{\mathcal{M}}$ and $\boldsymbol{\mathcal{X}}$, with respect to the i-th mode, respectively. These expressions form the basis of the AO algorithm for the solution of (3). Namely, we solve

$$\min_{\mathbf{U}^{(i)} \in \mathbb{R}_+^{I_i \times R}} \frac{1}{2} \left\| \mathbf{M}_{(i)} \circledast \left(\mathbf{X}_{(i)} - \mathbf{Y}_{(i)} \right) \right\|_F^2, \ i = 1, \ldots, N. \quad (6)$$

A. Nonnegative Matrix Completion

We consider the NMC problem, which will be the building block of our AO NTC algorithm. Let $\mathbf{X} \in \mathbb{R}_+^{P \times Q}$, $\mathbf{A} \in \mathbb{R}_+^{P \times R}$, $\mathbf{B} \in \mathbb{R}_+^{Q \times R}$, $\Omega \subseteq \{1, \ldots, P\} \times \{1, \ldots, Q\}$ be the set of indices of the known entries of $\mathbf{X}$, and $\mathbf{M}$ be the matrix with the same size as $\mathbf{X}$, with element $\mathbf{M}(i, j)$ equal to one or zero based on the availability of the corresponding element of $\mathbf{X}$. We consider the problem

$$\min_{\mathbf{A} \in \mathbb{R}_+^{P \times R}} f_\Omega(\mathbf{A}) := \frac{1}{2} \left\| \mathbf{M} \circledast \left(\mathbf{X} - \mathbf{A}\mathbf{B}^T \right) \right\|_F^2 + \frac{\lambda}{2} \|\mathbf{A}\|_F^2. \quad (7)$$

The gradient and the Hessian of f_Ω, at point $\mathbf{A}$, are given by

$$\nabla f_\Omega(\mathbf{A}) = - \left(\mathbf{M} \circledast \mathbf{X} - \mathbf{M} \circledast (\mathbf{A}\mathbf{B}^T) \right) \mathbf{B} + \lambda \mathbf{A}, \quad (8)$$

and

$$\nabla^2 f_\Omega(\mathbf{A}) = (\mathbf{B}^T \otimes \mathbf{I}_P) \text{diag}(\text{vec}(\mathbf{M}))(\mathbf{B} \otimes \mathbf{I}_P) + \lambda \mathbf{I}_{PR}. \quad (9)$$

B. Accelerated stochastic gradient for NMC

We solve problem (7) via the stochastic variant of the accelerated (Nesterov-type) gradient algorithm which appears in Algorithm 1.

During each iteration of the "while" loop, we use a subset of the available entries of matrix $\mathbf{X}$. More specifically, at iteration l, we define a set of indices $\widehat{\Omega}_l \subset \Omega$ and a matrix $\widehat{\mathbf{M}}_l$, of the same size as $\mathbf{M}$, as

$$\widehat{\mathbf{M}}_l(i, j) = \begin{cases} 1, & \text{if } (i, j) \in \widehat{\Omega}_l, \\ 0, & \text{otherwise}. \end{cases} \quad (10)$$

We create $\widehat{\Omega}_l$ randomly. We define $B_l := |\widehat{\Omega}_l|$ and select $c := \frac{B_l}{|\Omega|} < 1$. For row p of matrix $\mathbf{X}$, for $p = 1, \ldots, P$, we sample, uniformly at random, $B_{l,p} := \lfloor c \|\mathbf{M}(p, :)\|_0 \rfloor$ nonzero elements of $\mathbf{X}(p, :)$ ($B_{l,p}$ denotes the blocksize per row). If $B_{l,p} = 0$, then we skip the p-th row.

We perform an accelerated gradient step using only the elements of $\mathbf{X}$ whose indices appear in $\widehat{\Omega}_l$. Thus, our cost function becomes $f_{\widehat{\Omega}_l}$ with gradient and Hessian similar to those in (8) and (9), with the only difference being that $\mathbf{M}$ is replaced by $\widehat{\mathbf{M}}_l$. We find it convenient to compute the gradient and update the matrix variable in a row-wise fashion.

A novel feature of our algorithm is that each row of the matrix variable is updated via a different step-size, determined by the parameter L_p (see rows 9–13 of Algorithm 1). This can be motivated as follows. It is well known that the optimal step size for the gradient (or accelerated gradient) algorithm

Algorithm 1: Accelerated stochastic gradient for NMC

Input: $\mathbf{X}, \mathbf{M} \in \mathbb{R}_+^{P \times Q}$, $\mathbf{B} \in \mathbb{R}_+^{Q \times R}$, $\mathbf{A}_* \in \mathbb{R}_+^{P \times R}$, λ.

1   $\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$
2   $l = 0$
3   **while** ($l <$ MAX_INNER) **do**
4   **for** $p = 1 \ldots P$ **do**
5       $\widehat{\mathbf{M}}_l(p,:) = \text{sample}(\mathbf{M}(p,:))$
6       $\mathbf{W}_l(p,:) = -\left(\widehat{\mathbf{M}}_l(p,:) \circledast \mathbf{X}(p,:)\right)\mathbf{B}$
7       $\mathbf{Z}_l(p,:) = \left(\widehat{\mathbf{M}}_l(p,:) \circledast \left(\mathbf{Y}_l(p,:)\mathbf{B}^T\right)\right)\mathbf{B}$
8       $\nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l(p,:)) = \mathbf{W}_l(p,:) + \mathbf{Z}_l(p,:) + \lambda\mathbf{Y}_l(p,:)$
9       $\mathbf{H}_{l,p} = \mathbf{B}^T \text{diag}\left(\widehat{\mathbf{M}}_l(p,:)\right)\mathbf{B} + \lambda\mathbf{I}_R$
10      $L_p = \max(\text{eig}(\mathbf{H}_{l,p}))$
11      $\mathbf{A}_{l+1}(p,:) = \left(\mathbf{Y}_l(p,:) - \frac{1}{L_p}\nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l(p,:))\right)_+$
12      $\beta_{l,p} = \frac{\sqrt{L_p}-\sqrt{\lambda}}{\sqrt{L_p}+\sqrt{\lambda}}$
13      $\mathbf{Y}_{l+1}(p,:) = \mathbf{A}_{l+1}(p,:) + \beta_{l,p}\left(\mathbf{A}_{l+1}(p,:) - \mathbf{A}_l(p,:)\right)$
14     $l = l + 1$
15   **return** $\mathbf{A}_l$.

Algorithm 2: AO accelerated stochastic NTC

Input: $\mathcal{X}$, Ω, $\left\{\mathbf{U}_0^{(i)}\right\}_{i=1}^N$, λ, R.

1   $k = 0$
2   **while** (1) **do**
3   **for** $i = 1, 2, \ldots N$ **do**
4       $\mathbf{U}_{k+1}^{(i)} = \text{S\_NMC}\left(\mathbf{X}_{(i)}, \mathbf{M}_{(i)}, \mathbf{K}_k^{(i)}, \mathbf{U}_k^{(i)}, \lambda\right)$
5   **if** (term_cond is TRUE) **then** break; **endif**
6     $k = k + 1$
7   **return** $\left\{\mathbf{U}_k^{(i)}\right\}_{i=1}^N$.

Algorithm 3: Parallel accelerated stochastic gradient for NMC

Input: $\mathbf{X}, \mathbf{M} \in \mathbb{R}_+^{P \times Q}$, $\mathbf{B} \in \mathbb{R}_+^{Q \times R}$, $\mathbf{A}_* \in \mathbb{R}_+^{P \times R}$, λ

1   $\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$
2   $l = 0$
3   **while** (1) **do**
4   **if** ($l \geq$ MAX_INNER) **then**
5   break
6   **else**
7   **in parallel for** $p = 1 \ldots P$ **do**
8   lines 5-13 of Algorithm 1
9       $l = l + 1$
10   **return** $\mathbf{A}_l$.

---

for the minimization of a smooth convex function $f : \mathbb{R}^n \to \mathbb{R}$ is equal to $\frac{1}{L}$, where L satisfies $\nabla^2 f(\mathbf{x}) \preceq L\mathbf{I}$, for all $\mathbf{x} \in \mathbb{R}^n$. In line 9 of Algorithm 1, we compute $\mathbf{H}_{l,p}$ which is the Hessian of $f_{\widehat{\Omega}_l}$, with respect to the p-th row of $\mathbf{A}$. L_p is the largest eigenvalue of $\mathbf{H}_{l,p}$. For very sparse cases, the smallest eigenvalue of $\mathbf{H}_{l,p}$ is equal or very close to λ. Thus, the parameters used in the gradient and acceleration steps (i.e., lines 11 and 13 of Algorithm 1) are those used by the constant step scheme III of [5, p. 81], and can be considered as "locally optimal" for the problem at hand.

The most demanding computations of the algorithm are as follows:

1) the computation of $\mathbf{W}_l(p,:)$ requires $O(B_{l,p}R)$ arithmetic operations (in total, $O(B_lR)$);
2) the computation of $\mathbf{Z}_l(p,:)$ requires $O(B_{l,p}R)$ arithmetic operations (in total, $O(B_lR)$);
3) the computation of $\mathbf{H}_{l,p}$ requires $O(B_{l,p}R^2)$ arithmetic operations (in total, $O(B_lR^2)$);
4) the computation of L_p, via the power method, requires $O(R^2)$ arithmetic operations (in total, $O(PR^2)$).

The computation of each of the matrices $\nabla f_{\widehat{\Omega}_l}$, $\mathbf{A}_{l+1}$ and $\mathbf{Y}_{l+1}$ requires $O(PR)$ arithmetic operations.

For notational convenience, we denote Algorithm 1 as

$$\mathbf{A}_{\text{opt}} = \text{S\_NMC}(\mathbf{X}, \mathbf{M}, \mathbf{B}, \mathbf{A}_*, \lambda).$$

We note that, in this work, $\mathbf{H}_{l,p}$ is used only for the determination of L_p. A very interesting topic is the development of algorithms that fully exploit $\mathbf{H}_{l,p}$. Initial efforts with a "projected Newton step" have not led to algorithms superior to the one presented in this paper, especially in the noisy cases. A related important topic is the development of more efficient methods for the estimation of L_p.

III. AO ACCELERATED STOCHASTIC NTC

In order to solve the NTC problem using our accelerated stochastic algorithm, we start from initial values $\mathbf{U}_0^{(1)}, \ldots, \mathbf{U}_0^{(N)}$ and solve, in a circular manner, NMC problems, based on the previous estimates. We define

$$\mathbf{K}_k^{(i)} = \left(\mathbf{U}_k^{(N)} \odot \cdots \odot \mathbf{U}_k^{(i+1)} \odot \mathbf{U}_{k+1}^{(i-1)} \odot \cdots \odot \mathbf{U}_{k+1}^{(1)}\right),$$

where k denotes the k-th AO iteration. The update of $\mathbf{U}_k^{(i)}$ is attained by the function call $\text{S\_NMC}(\mathbf{X}_{(i)}, \mathbf{M}_{(i)}, \mathbf{K}_k^{(i)}, \mathbf{U}_k^{(i)}, \lambda)$. The Stochastic NTC algorithm appears in Algorithm 2.

A. Parallel Implementation of AO accelerated stochastic NMC

In Algorithm 3, we provide a high level algorithmic sketch of the accelerated stochastic gradient for NMC. We employ the OpenMP API, which is suitable for our multi-threading approach. The update of each row can be done independently. Therefore, lines 5-13 of Algorithm 1, can be computed separately by each available thread.

IV. NUMERICAL EXPERIMENTS

In this section, we test the effectiveness of our algorithm in various test cases, using both real-world and synthetic data. We denote as epoch the number of iterations required to access once all available tensor elements. In subsections IV-B and IV-C, we compute averages over 5 Monte Carlo trials.
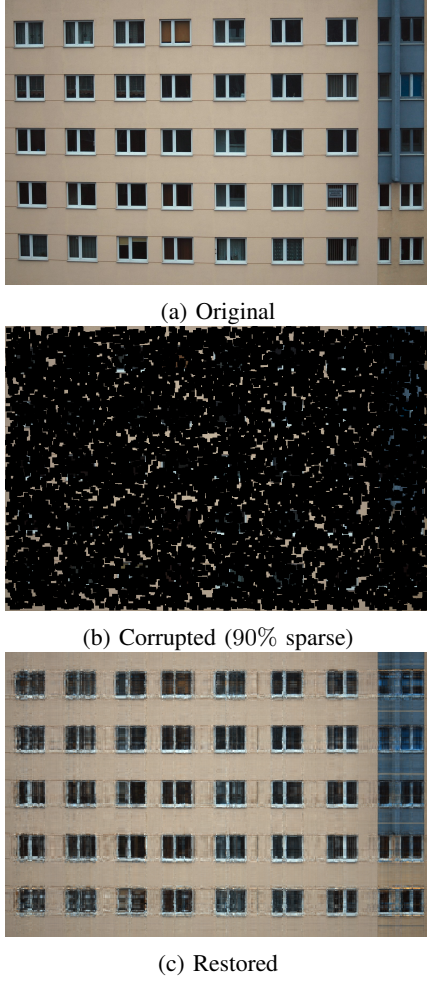
(a) Original

(b) Corrupted (90% sparse)

(c) Restored

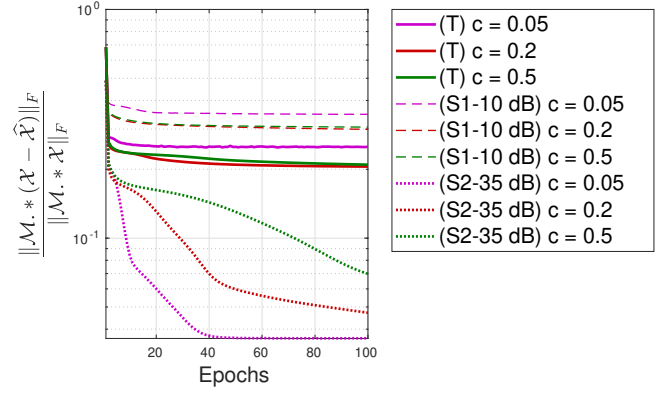Fig. 1: Tensor Completion on a corrupted image.

Fig. 2: Relative Reconstruction Error vs Epochs for real-world (T) and noisy synthetic (S1 and S2) data, using various values of $c = 0.05, 0.2, 0.5$.
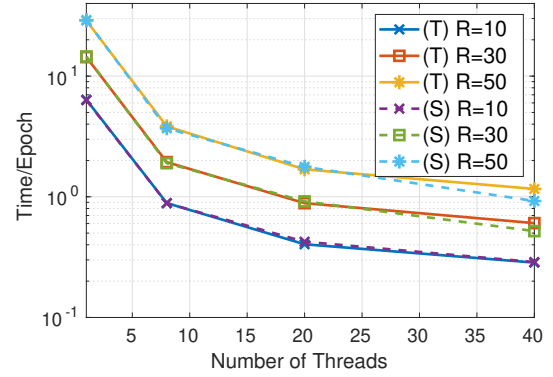

Fig. 3: Execution time per epoch (in seconds) for synthetic (S) and real-world (T) dataset with dimensions $183 \times 24 \times 1140 \times 1717$ and $c = 0.5$, for various values of rank R, versus number of threads.

A. Stochastic NTC on corrupted image

We start by estimating missing data in images,[1] following the RGB model. In Fig. 1, we depict the results obtained from the application of the AO Stochastic NTC on a corrupted image of dimensions $1063 \times 1599 \times 3$ (90% sparse). We set $c = 0.02$, number of epochs 500, MAX_INNER $= 1$, and rank $R = 50$. We observe that the algorithm is able to reconstruct the image even for small values of c.

B. Convergence speed of Stochastic NTC

We experimentally evaluate the convergence speed of our algorithm using both real-world and synthetic data. Concerning the real-world data, we use the dataset "MovieLens 10M" [23] of dimensions $71567 \times 65133 \times 730$. Concerning the synthetic data, we generate a rank-10 nonnegative tensor $\mathcal{X}^o$, of size equal to the real-world data, whose true latent factors have independent and identically distributed (i.i.d.) elements, drawn from $\mathcal{U}[0, 1]$. The additive noise $\mathcal{E}$ has i.i.d. elements $\mathcal{N}(0, \sigma_N^2)$. In order to create a tensor of the same sparsity level

[1]Image can be found at https://images.freeimages.com/images/large-previews/7bb/building-1222550.jpg

as the real-world dataset, we generate a tensor $\mathcal{M}$ of the same size as $\mathcal{X}^o$ according to (2). The observed incomplete tensor is expressed as $\mathcal{M} \circledast (\mathcal{X}^o + \mathcal{E})$. We define the Signal-to-Noise ratio as

$$\text{SNR} := \frac{\|\mathcal{M} \circledast \mathcal{X}^o\|_F^2}{\|\mathcal{M} \circledast \mathcal{E}\|_F^2}. \quad (11)$$

In Fig. 2, we plot the average Relative Reconstruction Error for 100 epochs, MAX_INNER $= 1$ and rank $R = 10$. We observe that, in noisy environments, choosing small values of c is not effective. On the contrary, in the high SNR cases, small values of c are more suitable.

C. Execution time for parallel stochastic NTC

We assess the performance of our algorithm in a shared-memory environment using both real-world and synthetic data (of the same dimensions). Concerning the real-world data, we use the dataset "Uber Pickups" [24], which can be represented as a 4–th order tensor of dimensions $183 \times 24 \times 1140 \times 1717$ and 3309490 nonzero elements. For the synthetic data, we create
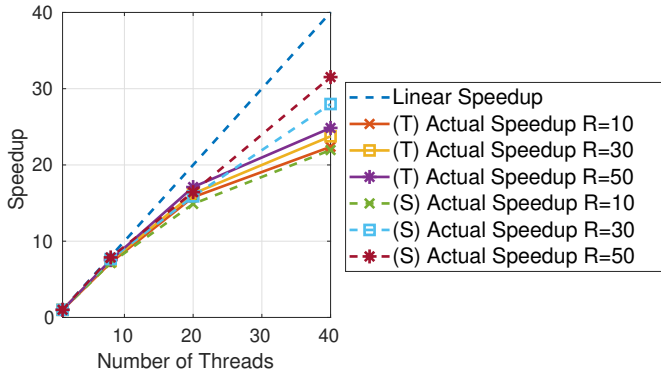
Fig. 4: Speedup attained for synthetic (S) and real-world (T) dataset with dimensions $183 \times 24 \times 1140 \times 1717$ and $c = 0.5$, for various values of rank R, versus number of threads.

a rank-10 tensor $\mathcal{X}^o$, whose latent factors have i.i.d elements $\mathcal{U}[0,1]$. Similarly, we generate a tensor $\mathcal{M}$ of the same size with $\mathcal{X}^o$. Thus, the observed incomplete and noiseless tensor is $\mathcal{M} \circledast \mathcal{X}^o$ with the same number of nonzeros as in the "Uber Pickups" dataset. We set $c = 0.5$, number of epochs 1, MAX_INNER $= 1$. In order to test the algorithm's performance, we examine various values of the rank $R = 10, 30, 50$. In Fig. 3, we illustrate the average execution time per epoch versus the number of threads. In Fig. 4, we present the average attained speedup. We observe that significant speedup is attained in all cases. We also observe that the speedup attained with synthetic data is somewhat higher than that attained with real-world data. This happens due to load imbalancing, caused by the nonuniform distribution of the nonzero elements of the real-world data, among the available threads.

V. Conclusion

We considered the NTC problem. First, we developed an accelerated stochastic algorithm for the NMC problem. A unique feature of our approach is that each row of the matrix variable is updated using a different step-size, specifically tailored to this row. Then, we used this algorithm and built an AO algorithm for the NTC problem. We tested the data reconstruction effectiveness as well as the convergence speed of our approach using both synthetic and real-world data. We implemented our algorithm using the OpenMP API, and observed significant speedup. Our method is an effective and efficient candidate for the solution of very large-scale NTC problems.

Acknowledgment

References

[1] P. M. Kroonenberg, *Applied Multiway Data Analysis*. Wiley-Interscience, 2008.

[2] A. Cichocki, R. Zdunek, A. H. Phan, and S. Amari, *Nonnegative Matrix and Tensor Factorizations*. Wiley, 2009.

[3] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, September 2009.

[4] 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*, vol. 65, no. 13, pp. 3551–3582, 2017.

[5] Y. Nesterov, *Introductory lectures on convex optimization*. Kluwer Academic Publishers, 2004.

[6] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: Scaling tensor analysis up by 100 times - algorithms and discoveries," *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD12) Beiging, China*, 2012.

[7] J. H. Choi and S. V. N. Vishwanathan, "Dfacto: Distributed factorization of tensors," *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[8] L. Karlsson, D. Kressner, and A. Uschmajew, "Parallel algorithms for tensor completion in the CP format," *Parallel Computing*, 2015.

[9] S. Smith and G. Karypis, "A medium-grained algorithm for distributed sparse tensor factorization," *30th IEEE International Parallel & Distributed Processing Symposium*, 2016.

[10] O. Kaya and B. Uçar, "Parallel candecomp/parafac decomposition of sparse tensors using dimension trees," *SIAM Journal on Scientific Computing*, vol. 40, no. 1, pp. C99–C130, 2018.

[11] K. Huang, N. D. Sidiropoulos, and A. P. Liavas, "A flexible and efficient framework for constrained matrix and tensor factorization," *IEEE Transactions on Signal Processing*, accepted for publication, May 2016.

[12] S. Smith, A. Beri, and G. Karypis, "Constrained tensor factorization with accelerated ao-admm," *2017 45th International Conference on Parallel Processing (ICPP), Bristol*, 2017.

[13] A. P. Liavas, G. Kostoulas, G. Lourakis, K. Huang, and N. D. Sidiropoulos, "Nesterov-based alternating optimization for nonnegative tensor factorization: Algorithm and parallel implementations," *IEEE Transactions on Signal Processing*, vol. 66, no. 4, pp. 944–953, Feb. 2018.

[14] G. Ballard, K. Hoyashi, and R. Kannan, "Parallel nonnegative cp decompositions of dense tensors," *2018 IEEE 25th International Conference on High Prformance Computing (HiPC), Bengaluru, India*, 2018.

[15] Z. Blanco, B. Liu, and M. M. Dehnavi, "Cstf: Large-scale sparse tensor factorizations on distributed platforms," in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3225058.3225133

[16] H. Ge, K. Zhang, M. Alfifi, X. Hu, and J. Caverlee, "Distenc: A distributed algorithm for scalable tensor completion on spark," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, 2018, pp. 137–148.

[17] K. Shin and U. Kang, "Distributed methods for high-dimensional and large-scale tensor factorization," in *IEEE International Conference on Data Mining, ICDM*, 2014, pp. 989–994.

[18] M. O. Karsavuran, S. Acer, and C. Aykanat, "Partitioning models for general medium-grain parallel sparse tensor decomposition," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 147–159, 2021.

[19] T. Papastergiou and V. Megalooikonomou, "A distributed proximal gradient descent method for tensor completion," in *2017 IEEE International Conference on Big Data (Big Data)*, 2017, pp. 2056–2065.

[20] S. Smith, J. Park, and G. Karypis, "An exploration of optimization algorithms for high performance tensor completion," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 359–371.

[21] K. Xie, Y. Chen, G. Wang, G. Xie, J. Cao, and J. Wen, "Accurate and fast recovery of network monitoring data: A gpu accelerated matrix completion," *IEEE/ACM Transactions on Networking*, vol. PP, pp. 1–14, 03 2020.

[22] K. D. Devine and G. Ballard, "Gentenmpi: Distributed memory sparse tensor decomposition." 8 2020.

[23] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, Dec. 2015. [Online]. Available: https://doi.org/10.1145/2827872

[24] S. Smith, J. W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, and G. Karypis. (2017) FROSTT: The formidable repository of open sparse tensors and tools. [Online]. Available: http://frostt.io/