Nonnegative Tensor Completion: step-sizes for an accelerated variation of the stochastic gradient descent

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

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

Christos Kolomvakis
Department of Mathematics and Operations Research
Faculty of Engineering, University of Mons
Mons, Belgium
christos.kolomvakis@umons.ac.be

***Abstract*—We consider the problem of nonnegative tensor completion. We adopt the alternating optimization framework and solve each nonnegative matrix least-squares problem via an accelerated variation of the stochastic gradient descent. The step-sizes used by the algorithm determine, to a high extent, its behavior. We propose two new strategies for the computation of step-sizes and we experimentally test their effectiveness using both synthetic and real-world data.**

***Index Terms*—tensors, nonnegative tensor completion, stochastic gradient descent, accelerated gradient, step-size selection, Armijo line-search, parallel algorithms, OpenMP.**

I. Introduction

Tensors have recently gained great popularity due to their ability to model multi-way data dependencies [1], [2], [3], [4]. The Canonical Polyadic Decomposition (CPD) is one of the most important tensor decomposition (TD) models. Tensor Completion (TC) arises in many modern applications such as machine learning, signal processing, and scientific computing.

Very large scale TD and TC problems are very computationally demanding. Recently, various approaches have been proposed to deal with this problem. An effective approach is the development and implementation of parallel algorithms (distributed or shared-memory) [5], [6], [7]. From a different perspective, stochastic gradient descent-based algorithms have gained much attention, since they are relatively easy to implement, have low computational cost, and can guarantee accurate solutions.

In this work, 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 given and the estimated tensor. We adopt the Alternating Optimization (AO) framework and update each factor via an accelerated variant (Nesterov-type) of the stochastic gradient descent [8].

All authors were partially supported by the European Regional Development Fund of the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship, and Innovation, under the call RESEARCH - CREATE - INNOVATE (project code : $\mathrm{T1E\Delta K - 03360}$).

A. Related Work

Works that employ Stochastic Gradient Descent (SGD) on shared-memory and distributed systems for sparse tensor factorization and completion include [9], [10], [11]. In [9], 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 [10], the authors examine three popular optimization algorithms: alternating least squares (ALS), SGD, and coordinate descent (CCD++), implemented on shared- and distributed-memory systems. In [11], the authors propose a GPU-accelerated parallel TC scheme (GPU-TC) for accurate and fast recovery of missing data via SGD.

In [12] and [13], a set of fibers is randomly selected at each iteration and a stochastic proximal gradient step is performed. In [14], the authors build upon the work of [13] and incorporate Nesterov acceleration at each iteration and a proximal term to deal with ill-conditioned cases. In [15], the authors propose an SGD algorithm for the Generalized CP decomposition. They propose various methods for the sampling of the elements and optimize the cost function using ADAM. The method can be applied to both sparse and dense tensors.

B. Contribution

The step-sizes used for the stochastic gradient step as well as the acceleration step are of great importance because they determine, to a great extent, the performance of the algorithm. Motivated by [16] and [17], we propose two strategies for the computation of the step-sizes. We test the effectiveness of our proposals using both synthetic and real-world data.

C. 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 $\boldsymbol{\mathcal{X}}(i_1, \ldots, i_N)$. Whenever convenient, 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 entries of the noise-corrupted tensor $\boldsymbol{\mathcal{X}}=\boldsymbol{\mathcal{X}}^o+\boldsymbol{\mathcal{E}}$. Our aim is to obtain estimates of the true factors $\mathbf{U}^{o(i)}$, for $i=1,\ldots,N$.

Let $\Omega\subseteq\{1,\ldots,I_1\}\times\cdots\times\{1,\ldots,I_N\}$ be the set of indices of the observed entries of $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{M}}$ be a tensor with the same size as $\boldsymbol{\mathcal{X}}$, with elements

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

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 $\lambda>0$ and

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

If $\boldsymbol{\mathcal{Y}}=[\![\mathbf{U}^{(1)},\ldots,\mathbf{U}^{(N)}]\!]$, then, for an arbitrary mode i, the corresponding matrix unfolding is given by [3]

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

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

$$f_\Omega(\mathbf{U}^{(1)},\ldots,\mathbf{U}^{(N)})=\frac{1}{2}\left\|\mathbf{M}_{(i)}\circledast\left(\mathbf{X}_{(i)}-\mathbf{Y}_{(i)}\right)\right\|_F^2,\quad(4)$$

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).

A. Nonnegative Matrix Least-Squares with Missing Elements

We consider the Nonnegative Matrix Least Squares problem with missing elements (NMLSME), 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 elements $\mathbf{M}(i,j)$ equal to one or zero based on the availability of the corresponding element of $\mathbf{X}$ [18]. We consider the problem

$$\min_{\mathbf{A}\in\mathbb{R}_+^{P\times R}} f_\Omega(\mathbf{A}):=\frac{1}{2}\left\|\mathbf{M}\circledast\left(\mathbf{X}-\mathbf{A}\mathbf{B}^T\right)\right\|_F^2+\frac{\lambda}{2}\|\mathbf{A}\|_F^2.\quad(5)$$

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

$$\nabla f_\Omega(\mathbf{A})=-\left(\mathbf{M}\circledast\mathbf{X}-\mathbf{M}\circledast(\mathbf{A}\mathbf{B}^T)\right)\mathbf{B}+\lambda\mathbf{A},\quad(6)$$

and

$$\nabla^2 f_\Omega(\mathbf{A})=(\mathbf{B}^T\otimes\mathbf{I}_P)\mathrm{diag}(\mathrm{vec}(\mathbf{M}))(\mathbf{B}\otimes\mathbf{I}_P)+\lambda\mathbf{I}_{PR}.\quad(7)$$

Algorithm 1: Accelerated stochastic gradient for NMLSME

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, $\mathbf{t}^{\text{init}}$

1 . $\mathbf{A}_0=\mathbf{Y}_0=\mathbf{A}_*$
2 $l=0$
3 **while** ($l<$MAX_INNER) **do**
4 **for** $p=1\ldots P$, **in parallel do**
5 $\widehat{\mathbf{M}}_l(p,:)=\mathrm{sample}(\mathbf{M}(p,:))$
6 Compute $\nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l(p,:))$
7 Compute step-size $t_{l,p}$ for gradient step
8 Compute $\mathbf{A}_{l+1}(p,:)$ via projected gradient step
9 Compute step-size $\beta_{l,p}$ for acceleration step
10 Compute $\mathbf{Y}_{l+1}(p,:)$ via acceleration step
11 $l=l+1$
12 **return** $\mathbf{A}_l$, $t_{l,p}$.

B. Accelerated stochastic gradient for NMLSME

We solve problem (5) via an accelerated variant (Nesterov-type) of the SGD algorithm which appears in Algorithm 1 and is described in detail in the sequel. Note that, according to the structure of the accelerated gradient algorithms, we update two matrix variables, denoted as $\mathbf{A}$ and $\mathbf{Y}$.

During each iteration of the "while" loop, we sample the available entries of matrix $\mathbf{X}$, leading to an analog of the batch SGD algorithm. More specifically, in the l-th iteration, we denote with $\widehat{\Omega}_l\subset\Omega$ the set of the indices of the sampled elements and with $\widehat{\mathbf{M}}_l$ the matrix, with the same size as $\mathbf{M}$, defined as

$$\widehat{\mathbf{M}}_l(i,j)=\begin{cases}1, & \text{if } (i,j)\in\widehat{\Omega}_l,\\ 0, & \text{otherwise.}\end{cases}\quad(8)$$

We define the cost function $f_{\widehat{\Omega}_l}$ analogously to (5), with $\widehat{\mathbf{M}}_l$ replacing $\mathbf{M}$.

We create $\widehat{\Omega}_l$ randomly. We define $B_l:=|\widehat{\Omega}_l|$ and set $c:=\frac{B_l}{|\Omega|}<1$. The algorithm works in a row-wise manner, for rows $p=1,\ldots,P$, as follows.

1) In line 5, we sample, uniformly at random, $B_{l,p}:=\lfloor c\|\mathbf{M}(p,:)\|_0\rfloor$ nonzero elements of $\mathbf{X}(p,:)$. If $B_{l,p}=0$, then we skip the p-th row.

2) In line 6, we compute the gradient $\nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l(p,:))$ as

$$\nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l(p,:)) = -\Big\{\widehat{\mathbf{M}}_l(p,:) \circledast \big[\mathbf{X}(p,:) - (\mathbf{Y}_l(p,:)\,\mathbf{B}^T)\big]\Big\}\mathbf{B} + \lambda \mathbf{Y}_l(p,:). \tag{9}$$

3) In line 7, compute the step-size $t_{l,p}$ for the gradient update (the initial value of the step-size is t_p^{init}). This is a very important step on which we will elaborate in the sequel.
4) In line 8, we perform the projected (stochastic) gradient update

$$\mathbf{A}_{l+1}(p,:) = \Big(\mathbf{Y}_l(p,:) - t_{l,p}\nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l(p,:))\Big)_+. \tag{10}$$

5) In line 9, we compute the momentum-related step-size, $\beta_{l,p}$. This is the second point on which we shall elaborate in the sequel.
6) In line 10, we perform the momentum step

$$\mathbf{Y}_{l+1}(p,:) = \mathbf{A}_{l+1}(p,:) + \beta_{l,p}(\mathbf{A}_{l+1}(p,:) - \mathbf{A}_l(p,:)).$$

For notational convenience, we denote Algorithm 1 as

$$(\mathbf{A}_{\text{new}}, \mathbf{t}_{\text{new}}) = \text{S\_NMLSME}(\mathbf{X}, \mathbf{M}, \mathbf{B}, \mathbf{A}_*, \lambda, \mathbf{t}^{\text{init}}).$$

The step-sizes $t_{l,p}$ and $\beta_{l,p}$ are very important parameters, which determine, to a great extent, the behavior of the algorithm.

III. Step-size selection

In this section, we motivate and present three different strategies for computing the required step-size values.

A. Step-sizes based on "local Hessian"

The first step-size we consider has been proposed in [19] and its computation is as follows. We define the second derivative of $f_{\widehat{\Omega}_l}$, with respect to $\mathbf{Y}(p,:)$, as

$$\mathbf{H}_{l,p} := \nabla^2 f_{\widehat{\Omega}_l}(\mathbf{Y}(p,:)). \tag{11}$$

A commonly used step-size, t, for the (projected) gradient method for the minimization of a smooth convex function $f : \mathbb{R}^n \to \mathbb{R}$, is $t = \frac{1}{L}$, where L satisfies $\nabla^2 f(\mathbf{x}) \preceq L\mathbf{I}_n$, for all $\mathbf{x} \in \mathbb{R}^n$ [8]. Let $L_{l,p}$ be the largest eigenvalue of $\mathbf{H}_{l,p}$. Then, we set

$$t_{l,p} = \frac{1}{L_{l,p}}. \tag{12}$$

Furthermore, by the definition of $f_{\widehat{\Omega}_l}$, we have that

$$\mathbf{H}_{l,p} \succeq \lambda \mathbf{I}. \tag{13}$$

We can perform the acceleration step using either the constant step scheme II of [8, p. 80] or the constant step scheme III of [8, p. 81], or we may choose to neglect the acceleration step (in this case, we just perform a stochastic gradient step). If we use scheme II, we compute $\alpha_{l+1,p} \in (0,1)$

$$\alpha_{l+1,p}^2 = (1 - \alpha_{l+1,p})\alpha_{l,p}^2 + q_p\alpha_{l+1,p}, \tag{14}$$

where $q_p = \lambda t_{l,p}$ and $\alpha_{0,p} = 1$, and set

$$\beta_{l,p} = \frac{\alpha_{l,p}(1 - \alpha_{l,p})}{\alpha_{l,p}^2 + \alpha_{l+1,p}}. \tag{15}$$

If we use scheme III, we set

$$\beta_{l,p} = \frac{\sqrt{L_{l,p}} - \sqrt{\lambda}}{\sqrt{L_{l,p}} + \sqrt{\lambda}}. \tag{16}$$

We note that the computation of $\mathbf{H}_{l,p}$ requires $O(B_{l,p}R^2)$ arithmetic operations (in total, $O(B_lR^2)$) and the computation of $L_{l,p}$, via the power method, requires $O(R^2)$ arithmetic operations (in total, $O(PR^2)$). Thus, more efficient methods for the computation of effective step-sizes are of great interest.

B. Step-sizes based on [16]

Our first proposal is motivated by [16, Section 6]. We start from an initial value $t_{l,p} = t_p^{\text{init}}$, and perform a projected gradient step

$$Y_l^+(p,:) = \Big(\mathbf{Y}_l(p,:) - t_{l,p}\nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l(p,:))\Big)_+. \tag{17}$$

We compute the gradient at the new point, $\nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l^+(p,:))$. If

$$\nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l^+(p,:))^T \nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l(p,:)) < 0, \tag{18}$$

then we set $t_{l,p} = \frac{t_{l,p}}{2}$ and repeat the procedure until relation (18) does not hold true. Then, we set $\mathbf{A}_{l+1}(p,:) = \mathbf{Y}_l^+(p,:)$. Note that if relation (18) holds true, then we have a strong indication that the gradient step from $\mathbf{Y}_l(p,:)$ to $\mathbf{Y}_l^+(p,:)$ is too long.

If $t_{l,p}$ is the step-size value we used for the computation of $\mathbf{A}_l(p,:)$, then we either set

$$\beta_{l,p} = \frac{\sqrt{1/t_{l,p}} - \sqrt{\lambda}}{\sqrt{1/t_{l,p}} - \sqrt{\lambda}}, \tag{19}$$

or compute the value of $\beta_{l,p}$ it according to (14) and (15).

C. Step-sizes based on Armijo-type Line Search

Our second proposal is an Armijo type line-search technique motivated by [17]. More specifically, starting from an initial value $t_{l,p} = t_p^{\text{init}}$, we are looking for a step-size $t_{l,p}$ such that

$$f_{\widehat{\Omega}_l}\Big((\mathbf{Y}_l(p,:) - t_{l,p}\nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l(p,:)))_+\Big) \leq f_{\widehat{\Omega}_l}(\mathbf{Y}_l(p,:)) - \gamma\, t_{l,p}\left\|\nabla f_{\widehat{\Omega}_l}(\mathbf{Y}_l(p,:))\right\|_2^2, \tag{20}$$

where γ is a hyper-parameter. If relation (20) does not hold true, then we backtrack by a constant factor δ, that is, we set $t_{l,p} = \delta\, t_{l,p}$, until the line-search succeeds. The last value of $t_{l,p}$ is used for the projected (stochastic) gradient step. Note that the line-search proposed in (20) requires the computation of the value of $f_{\widehat{\Omega}_l}$ instead of f_{Ω_l}.

For the acceleration step, we either set

$$\beta_{l,p} = \frac{\sqrt{1/t_{l,p}} - \sqrt{\lambda}}{\sqrt{1/t_{l,p}} - \sqrt{\lambda}}, \tag{21}$$

or set the value of $\beta_{l,p}$ according to (14) and (15).

Algorithm 2: AO accelerated stochastic NTC

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

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

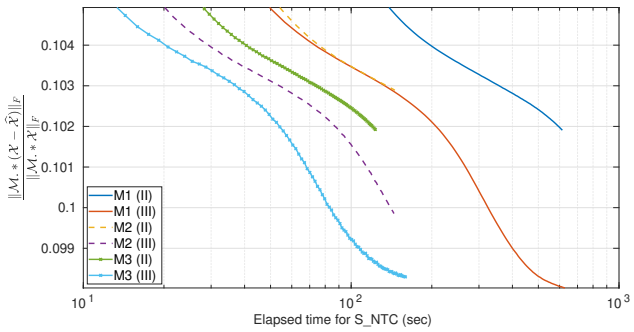

Fig. 1: Relative tensor estimation error vs elapsed CPU time for synthetic data.

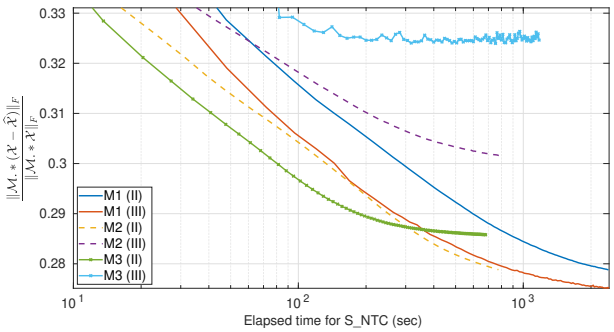

Fig. 2: Relative tensor estimation error vs elapsed CPU time for Chicago Crime (4D) Dataset ($R = 50$).

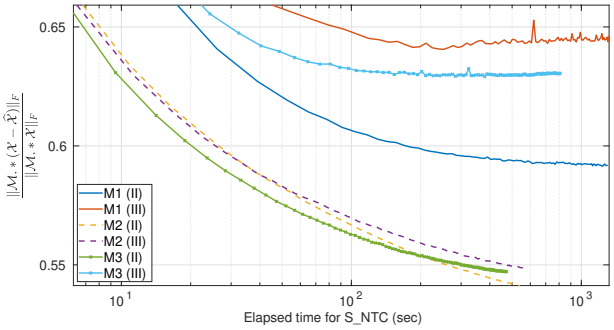

Fig. 3: Relative tensor estimation error vs elapsed CPU time for NIPS Dataset ($R = 40$).

D. Parallel Implementation

We use the same parallel multi-threaded implementation, via OpenMP, of Algorithm 1 as presented in [19]. In a nutshell, the update of each row in lines $5-10$ of Algorithm 1, can be computed separately by each available thread.

IV. AO accelerated stochastic NTC

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, NMLSME 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 $\mathbf{U}_k^{(i)}$ and $\mathbf{t}_k^{(i)}$ is attained by the function call

$$\text{S\_NMLSME}(\mathbf{X}_{(i)}, \mathbf{M}_{(i)}, \mathbf{K}_k^{(i)}, \mathbf{U}_k^{(i)}, \lambda, \mathbf{t}_k^{(i)}).$$

The Stochastic NTC algorithm appears in Algorithm 2.

V. Numerical Experiments

In this section, we test the effectiveness of the proposed step-sizes using both synthetic and real-world data. We denote as epoch the number of iterations required to access once all available tensor elements.

In our experiments, we run all algorithms for 100 epochs, on a multicore shared-memory system using 40 threads. The program is executed on a DELL PowerEdge R820 system with SandyBridge - Intel(R) Xeon(R) CPU E5 - 4650v2 (in total, 16 nodes with 40 cores each at 2.4 GHz) and 512 GB RAM per node. The matrix operations are implemented using routines of the C++ library Eigen [20].

In all experiments, we set MAX_INNER $= 5$. That is, each time we call the function S_NMLSME we perform 5 iterations. We set $c = 0.2$ (20%), $\gamma = 0.1$, and $\delta = 0.5$. Finally, we set $\lambda = 10^{-4}$ for the synthetic data and $\lambda = 10^{-2}$ for the real-world data. We use the initialization

$$t_{0,p}^{(i)} = \frac{1}{L_p^{\text{init},(i)}}, \text{ for } p = 1, \ldots, P, \ i = 1, \ldots, N, \qquad (22)$$

where $L_p^{\text{init},(i)}$ is the largest eigenvalue of the Hessian of the cost function, with respect to the p-th row of the i-th factor in the first AO iteration of the algorithm. Then, the initial values of the step-sizes propagate as shown in Line 4 of Algorithm 2.

In our plots, we denote the step-sizes proposed in subsections III-A, III-B, and III-C, as M1, M2, and M3, respectively.

A. Synthetic Data

We generate the rank-50 tensor $\mathcal{X}^o \in \mathbb{R}_+^{3000\times1700\times65}$ whose 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 $\mathcal{N}(0, \sigma_N^2)$. The observed incomplete tensor

$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{M}} \circledast (\boldsymbol{\mathcal{X}}^o + \boldsymbol{\mathcal{E}})$ has 5M nonzero elements. The Signal-to-Noise ratio is defined as

$$\mathrm{SNR} := \frac{\|\boldsymbol{\mathcal{M}} \circledast \boldsymbol{\mathcal{X}}^o\|_F^2}{\|\boldsymbol{\mathcal{M}} \circledast \boldsymbol{\mathcal{E}}\|_F^2}.$$

In Fig. 1, we plot the average, over 5 Monte-Carlo trials, relative tensor estimation error as a function of the elapsed CPU time, for $R = 50$ and $\mathrm{SNR} = 20\,\mathrm{dB}$. We observe that M2 and M3 lead to algorithms which demand much smaller CPU time than M1. The estimation performance depends on the acceleration scheme, with Scheme III being the most effective. M2 is slightly worse than M3, in terms of estimation error.

B. Real-world Data

We first consider the "Chicago Crime" Dataset, where the data form tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{6186\times 24\times 77\times 32}$ with $5.3\mathrm{M}$ nonzero elements,

In Fig. 2, we plot the relative tensor estimation error vs the elapsed CPU time, for $R = 50$. We observe that M2 and M3 lead to faster algorithms but attain slightly worse tensor estimation error than M1. Method M2 with acceleration scheme II seems the most prominent, in the sense that it attains slightly worse estimation error than M1, but much faster.

Finally, we consider the "NIPS Publications" Dataset, where the data form tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{2482\times 2862\times 14036\times 17}$, with $3.1\mathrm{M}$ nonzero elements. In Fig. 3, we plot the relative tensor estimation error vs the elapsed CPU time, for $R = 40$. In this case, M2 and M3 lead to faster algorithms but also achieve better estimation error than M1. Again, M2 with acceleration scheme II seems the most prominent method.

VI. Conclusion

We considered the NTC problem. We adopted the AO concept and updated each factor by using a stochastic variant of the accelerated gradient descent. Motivated by previous works in related contexts, we proposed step-size strategies for the stochastic gradient steps for the solution of NMLSME problems. We tested the data reconstruction effectiveness as well as the convergence speed of our proposals using both synthetic and real-world data.

Acknowledgment

This work was also supported by computational time granted from the National Infrastructures for Research and Technology S.A. (GRNET S.A.) in the National HPC facility - ARIS - under project ID PR008040-PARTENSOR SPARSE.

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

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

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] C. Battaglino, G. Ballard, and T. G. Kolda, "A practical randomized cp tensor decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 39, no. 2, pp. 876–901, 2018.

[13] X. Fu, S. Ibrahim, H.-T. Wai, C. Gao, and K. Huang, "Block-randomized stochastic proximal gradient for low-rank tensor factorization," *IEEE Transactions on Signal Processing*, 2020.

[14] I. Siaminou and A. P. Liavas, "An accelerated stochastic gradient for canonical polyadic decomposition," in *2021 29th European Signal Processing Conference (EUSIPCO)*, 2021, pp. 1785–1789.

[15] T. G. Kolda and D. Hong, "Stochastic gradients for large-scale tensor decomposition," *SIAM Journal on Mathematics of Data Science*, vol. 2, no. 4, pp. 1066–1095, 2020. [Online]. Available: https://doi.org/10.1137/19M1266265

[16] Y. Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems," *Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), CORE Discussion Papers*, vol. 22, 01 2010.

[17] S. Vaswani, A. Mishkin, I. Laradji, M. Schmidt, G. Gidel, and S. Lacoste-Julien, "Painless stochastic gradient: Interpolation, line-search, and convergence rates," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[18] A. Buchanan and A. Fitzgibbon, "Damped newton algorithms for matrix factorization with missing data," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, 2005, pp. 316–322 vol. 2.

[19] I. Siaminou, I. M. Papagiannakos, C. Kolomvakis, and A. P. Liavas, "Accelerated stochastic gradient for nonnegative tensor completion and parallel implementation," in *2021 29th European Signal Processing Conference (EUSIPCO)*, 2021, pp. 1790–1794.

[20] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.