A Flexible and Efficient Algorithmic Framework for Constrained Matrix and Tensor Factorization

Kejun Huang, *Student Member, IEEE*, Nicholas D. Sidiropoulos, *Fellow, IEEE*, and Athanasios P. Liavas, *Member, IEEE*

***Abstract*—We propose a general algorithmic framework for constrained matrix and tensor factorization, which is widely used in signal processing and machine learning. The new framework is a hybrid between alternating optimization (AO) and the alternating direction method of multipliers (ADMM): each matrix factor is updated in turn, using ADMM, hence the name AO-ADMM. This combination can naturally accommodate a great variety of constraints on the factor matrices, and almost all possible loss measures for the fitting. Computation caching and warm start strategies are used to ensure that each update is evaluated efficiently, while the outer AO framework exploits recent developments in block coordinate descent (BCD)-type methods which help ensure that every limit point is a stationary point, as well as faster and more robust convergence in practice. Three special cases are studied in detail: non-negative matrix/tensor factorization, constrained matrix/tensor completion, and dictionary learning. Extensive simulations and experiments with real data are used to showcase the effectiveness and broad applicability of the proposed framework.**

***Index Terms*—Constrained matrix/tensor factorization, non-negative matrix/tensor factorization, canonical polyadic decomposition, PARAFAC, matrix/tensor completion, dictionary learning, alternating optimization, alternating direction method of multipliers.**

I. Introduction

CONSTRAINED matrix and tensor factorization techniques are widely used for latent parameter estimation and blind source separation in signal processing, dimensionality reduction and clustering in machine learning, and numerous other applications in diverse disciplines, such as chemistry and psychology. Least-squares low-rank factorization of matrices and tensors without additional constraints is relatively well-studied, as in the matrix case the basis of any solution is simply the principal components of the singular value decomposition (SVD) [2], also known as principal component analysis (PCA), and in the tensor case alternating least squares (ALS) and other algorithms usually yield satisfactory results [3]. ALS is also used for matrix factorization, especially when the size is so large that performing the exact PCA is too expensive.

Whereas unconstrained matrix and tensor factorization algorithms are relatively mature, their *constrained* counterparts leave much to be desired as of this writing, and a unified framework that can easily and naturally incorporate multiple constraints on the latent factors is sorely missing. Existing algorithms are usually only able to handle one or at most few specialized constraints, and/or the algorithm needs to be redesigned carefully if new constraints are added. Commonly adopted constraints imposed on the latent factors include non-negativity [4], sparsity (usually via sparsity-inducing ℓ_1 regularization [5]), and simplex constraints [6], to name just a few.

On top of the need to incorporate constraints on the latent factors, many established and emerging signal processing applications entail cost (*loss*) functions that differ from classical least-squares. Important examples include matrix completion [7] where missing values are ignored by the loss function, and robust PCA [8] where the ℓ_1 loss is used. In the matrix case without constraints on the latent factors, these can be formulated as convex problems via nuclear norm regularization and solved in polynomial-time [9]. With explicit constraints imposed on the latent factors, and/or for tensor data, however, non-convex (multi-linear) formulations are unavoidable, and a unified algorithmic framework that can handle a variety of constraints and loss functions would be very welcome.

In this paper, we propose a general algorithmic framework that seamlessly and relatively effortlessly incorporates many common types of constraints and loss functions, building upon and bridging together the alternating optimization (AO) framework and the alternating direction method of multipliers (ADMM), hence the name AO-ADMM.

While combining these frameworks may seem conceptually straightforward at first sight, what is significant is that AO-ADMM outperforms all prior algorithms for constrained matrix and tensor factorization under nonparametric constraints on the latent factors. One example is non-negative matrix factorization, where the prior art includes decades of research. This is the biggest but not the only advantage of AO-ADMM. Carefully developing various aspects of this combination, we show that

- AO-ADMM converges to a stationary point of the original NP-hard problem;
- Using computation caching, warm-start, and good parameter settings, its per-iteration complexity is similar to that of ALS;

Manuscript received October 22, 2015; revised April 19, 2016; accepted May 20, 2016. Date of publication June 07, 2016; date of current version August 06, 2016. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Wei Liu. Part of this work was presented at EUSIPCO 2015 [1]. Their work was supported in part by NSF IIS-1247632, IIS-1447788, and a UM Informatics Institute fellowship.

K. Huang and N. D. Sidiropoulos are with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: huang663@umn.edu; nikos@umn.edu).

A. P. Liavas is with the Department of Electronic and Computer Engineering, Technical University of Crete, Chania 73100, Greece (e-mail: liavas@telecom.tuc.gr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2016.2576427

- AO-ADMM can incorporate a wide-range of constraints and regularization penalties on the latent factors at essentially the same complexity;
- It can also accommodate a wide variety of cost/loss functions, with only moderate increase in complexity relative to the classical least-squares loss; and
- The core computations are exactly the same as ALS for unconstrained factorization, with some additional element-wise operations to handle constraints, making it easy to incorporate smart implementations of ALS, including sparse, parallel, and high-performance computing enhancements.

A. Notation

We denote the (approximate) factorization of a matrix $\boldsymbol{Y} \approx \boldsymbol{W}\boldsymbol{H}^T$, where $\boldsymbol{Y}$ is $m \times n$, $\boldsymbol{W}$ is $m \times k$, and $\boldsymbol{H}$ is $n \times k$, with $k \leq m, n$, and in most cases much smaller. Note that adding constraints on $\boldsymbol{W}$ and $\boldsymbol{H}$ may turn the solution from easy to find (via SVD) but non-unique, to NP-hard to find but essentially unique. It has been shown that simple constraints like non-negativity and sparsity can make the factors identifiable, but at the same time, computing the optimal solution becomes NP-hard—see [10] and references therein.

An N-way array of dimension $n_1 \times n_2 \times \cdots \times n_N$, with $N \geq 3$, is denoted with an underscore, e.g., $\underline{\boldsymbol{Y}}$. In what follows, we focus on the so-called parallel factor analysis (PARAFAC) model, also known as canonical decomposition (CANDECOMP) or canonical polyadic decomposition (CPD), which is essentially unique under mild conditions [11], but constraints certainly help enhance estimation performance, and even identifiability. The factorization is denoted as $\underline{\boldsymbol{Y}} \approx [\boldsymbol{H}_d]_{d=1}^N$, which is a concise way of representing the model

$$\underline{\boldsymbol{Y}}(i_1, \ldots, i_N) \approx \sum_{j=1}^{k} \prod_{d=1}^{N} \boldsymbol{H}_d(i_d, j), \quad \forall i_1, \ldots, i_N.$$

Each matrix $\boldsymbol{H}_d$ is of size $n_d \times k$, corresponding to the factor of the d-th mode.

B. Multi-Linear Algebra Basics

With the increasing interest in tensor data processing, there exist many tutorials on this topic, for example, [12][13][14]. Here we briefly review some basic multi-linear operations that will be useful for the purposes of this paper, and refer the readers to those tutorials and the references therein for a more comprehensive introduction.

The **mode-d matricization**, also known as mode-d matrix unfolding, of $\underline{\boldsymbol{Y}}$, denoted as $\boldsymbol{Y}_{(d)}$, is a matrix of size $\prod_{j=1, j\neq d}^{N} n_j \times n_d$. Each row of $\boldsymbol{Y}_{(d)}$ is a vector obtained by fixing all the indices of $\underline{\boldsymbol{Y}}$ except the d-th one, and the matrix is formed by stacking these row vectors by traversing the rest of the indices from N back to 1. As an example, for $N = 3$, the three matricizations are

$$\boldsymbol{Y}_{(1)} = \begin{bmatrix} \underline{\boldsymbol{Y}}(:,1,:)^T \\ \vdots \\ \underline{\boldsymbol{Y}}(:,n_2,:)^T \end{bmatrix}, \boldsymbol{Y}_{(2)} = \begin{bmatrix} \underline{\boldsymbol{Y}}(1,:,:)^T \\ \vdots \\ \underline{\boldsymbol{Y}}(n_1,:,:)^T \end{bmatrix} \boldsymbol{Y}_{(3)} = \begin{bmatrix} \underline{\boldsymbol{Y}}(1,:,:) \\ \vdots \\ \underline{\boldsymbol{Y}}(n_1,:,:) \end{bmatrix},$$

where $\underline{\boldsymbol{Y}}(i,:,:)$, $\underline{\boldsymbol{Y}}(:,i,:)$, $\underline{\boldsymbol{Y}}(:,:,i)$ are the i-th matrix slabs of the three-way tensor $\underline{\boldsymbol{Y}}$, of size $n_2 \times n_3$, $n_1 \times n_3$, $n_1 \times n_2$, respectively. Notice that, though essentially in the same spirit, this definition of mode-d matricization may be different from other expressions that have appeared in the literature, but we adopt this one for ease of our use.

The **Khatri-Rao product** of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ having the same number of columns, denoted as $\boldsymbol{A} \odot \boldsymbol{B}$, is defined as the column-wise Kronecker product of $\boldsymbol{A}$ and $\boldsymbol{B}$. More explicitly, if $\boldsymbol{A}$ is of size $n \times k$, then

$$\begin{aligned} \boldsymbol{A} \odot \boldsymbol{B} &= \begin{bmatrix} \boldsymbol{A}(:,1) \otimes \boldsymbol{B}(:,1) & \cdots & \boldsymbol{A}(:,k) \otimes \boldsymbol{B}(:,k) \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{A}(1,1)\boldsymbol{B}(:,1) & \cdots & \boldsymbol{A}(1,k)\boldsymbol{B}(:,k) \\ \vdots & \cdots & \vdots \\ \boldsymbol{A}(n,1)\boldsymbol{B}(:,1) & \cdots & \boldsymbol{A}(n,k)\boldsymbol{B}(:,k) \end{bmatrix}. \end{aligned}$$

The Khatri-Rao product is associative (although not commutative). We therefore generalize the operator $\odot$ to accept more than two arguments in the following way

$$\bigodot_{\substack{j=1 \\ j\neq d}}^{N} \boldsymbol{H}_i = \boldsymbol{H}_1 \odot \cdots \odot \boldsymbol{H}_{d-1} \odot \boldsymbol{H}_{d+1} \odot \cdots \odot \boldsymbol{H}_N.$$

With the help of this notation, if $\underline{\boldsymbol{Y}}$ admits an exact PARAFAC model $\underline{\boldsymbol{Y}} = [\boldsymbol{H}_d]_{d=1}^N$, then it can be expressed in matricized form as

$$\boldsymbol{Y}_{(d)} = \left(\bigodot_{\substack{j=1 \\ j\neq d}}^{N} \boldsymbol{H}_j \right) \boldsymbol{H}_d^T.$$

Lastly, a nice property of the Khatri-Rao product is that

$$(\boldsymbol{A} \odot \boldsymbol{B})^T (\boldsymbol{A} \odot \boldsymbol{B}) = \boldsymbol{A}^T\boldsymbol{A} \circledast \boldsymbol{B}^T\boldsymbol{B},$$

where $\circledast$ denotes the element-wise (Hadamard) matrix product. More generally, it holds that

$$\left(\bigodot_{\substack{j=1 \\ j\neq d}}^{N} \boldsymbol{H}_j \right)^T \left(\bigodot_{\substack{j=1 \\ j\neq d}}^{N} \boldsymbol{H}_j \right) = \bigcircledast_{\substack{j=1 \\ j\neq d}}^{N} \boldsymbol{H}_j^T \boldsymbol{H}_j.$$

II. Alternating Optimization Framework: Preliminaries

We start by formulating the factorization problem as an optimization problem in the most general form

$$\underset{\boldsymbol{H}_1, \ldots, \boldsymbol{H}_N}{\text{minimize}} \quad l\left(\underline{\boldsymbol{Y}} - [\boldsymbol{H}_d]_{d=1}^N\right) + \sum_{d=1}^{N} r_d(\boldsymbol{H}_d), \qquad (1)$$

with a slight abuse of notation by assuming N can also take the value of 2. In (1), $l(\cdot)$ can be any loss measure, most likely separable down to the entries of the argument, and $r_d(\boldsymbol{H}_d)$ is

the generalized regularization on $\boldsymbol{H}_d$, which may take the value of $+\infty$ so that any hard constraints can also be incorporated. For example, if we require that the elements of $\boldsymbol{H}_d$ are nonnegative, denoted as $\boldsymbol{H}_d \geq 0$, then

$$r_d\left(\boldsymbol{H}_d\right) = \begin{cases} 0, & \boldsymbol{H}_d \geq 0, \\ +\infty, & \text{otherwise.} \end{cases}$$

Because of the multi-linear term $[\boldsymbol{H}_d]_{d=1}^N$, the regularized fitting problem is non-convex, and in many cases NP-hard [15], [16]. A common way to handle this is to use the alternating optimization (AO) technique, i.e., update each factor $\boldsymbol{H}_d$ in a cyclic fashion. The popular ALS algorithm is a special case of this when $l\left(\cdot\right)$ is the least-squares loss, and there is no regularization. In this section, we will first revisit the ALS algorithm, with the focus on the per-iteration complexity analysis. Then, we will briefly discuss the convergence of the AO framework, especially some recent advances on the convergence of the traditional block coordinate descent (BCD) algorithm.

A. Alternating Least-Squares Revisited

Consider the unconstrained matrix factorization problem

$$\underset{\boldsymbol{W},\boldsymbol{H}}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{W}\boldsymbol{H}^T\|_F^2, \tag{2}$$

and momentarily ignore the fact that the optimal solution of (2) is given by the SVD. The problem (2) is non-convex in $\boldsymbol{W}$ and $\boldsymbol{H}$ jointly, but is convex if we fix one and treat only the other as variable. Supposing $\boldsymbol{W}$ is fixed, the sub-problem for $\boldsymbol{H}$ becomes the classical linear least squares and, if $\boldsymbol{W}$ has full column rank, the unique solution is given by

$$\boldsymbol{H}^T = \left(\boldsymbol{W}^T\boldsymbol{W}\right)^{-1}\boldsymbol{W}^T\boldsymbol{Y}. \tag{3}$$

In practice, the matrix inverse $\left(\boldsymbol{W}^T\boldsymbol{W}\right)^{-1}$ is almost never explicitly calculated. Instead, the Cholesky decomposition of the Gram matrix $\boldsymbol{W}^T\boldsymbol{W}$ is computed, and for each column of $\boldsymbol{W}^T\boldsymbol{Y}$, a forward and a backward substitution are performed to get the corresponding column of $\boldsymbol{H}^T$. Since $\boldsymbol{W}$ is $m \times k$ and $\boldsymbol{Y}$ is $m \times n$, forming $\boldsymbol{W}^T\boldsymbol{W}$ and $\boldsymbol{W}^T\boldsymbol{Y}$ takes $\mathcal{O}\left(mk^2\right)$ and $\mathcal{O}\left(mnk\right)$ flops, respectively, computing the Cholesky decomposition requires $\mathcal{O}\left(k^3\right)$ flops, and finally the back substitution step takes $\mathcal{O}\left(nk^2\right)$ flops, similar to a matrix multiplication. If $m, n > k$, then the overall complexity is $\mathcal{O}\left(mnk\right)$.

An important implication is the following. Clearly, if $n = 1$, the cost of solving a least-squares problem is $\mathcal{O}\left(mk^2\right)$. For $n > 1$, however, the complexity becomes $\mathcal{O}\left(mnk\right)$ instead of $O\left(mnk^2\right)$, because we can amortize a factor of k. The reason is that, although it seems we are now trying to solve n least-squares problems, they all share the same matrix $\boldsymbol{W}$, thus the Cholesky decomposition of $\boldsymbol{W}^T\boldsymbol{W}$ can be reused throughout. This is a very nice property of *unconstrained* least squares problems, which can be exploited to improve the computational efficiency of the ALS algorithm.

One may recall that another well-adopted method for least-squares is to compute the QR decomposition of $\boldsymbol{W}$ as $\boldsymbol{W} = \mathbf{Q}\mathbf{R}$, so that $\boldsymbol{H}^T = \mathbf{R}^{-1}\mathbf{Q}^T\boldsymbol{Y}$. This can be shown to give the same computational complexity as the Cholesky version, and is actually more stable numerically. However, if $\boldsymbol{W}$ has some special structure, it is easier to exploit this structure if we use Cholesky decomposition. Therefore, in this paper we only consider solving least-squares problems using the Cholesky decomposition.

One important structure that we encounter is in the tensor case. For the ALS algorithm for PARAFAC, the update of $\boldsymbol{H}_d$ is the solution of the following least squares problem

$$\underset{\boldsymbol{H}_d}{\text{minimize}} \quad \frac{1}{2}\left\|\boldsymbol{Y}_{(d)} - \left(\bigodot_{\substack{j=1\\ j\neq d}}^{N} \boldsymbol{H}_j\right)\boldsymbol{H}_d^T\right\|_F^2,$$

and the solution is given by

$$\boldsymbol{H}_d^T = \left(\circledast_{\substack{j=1\\ j\neq d}}^{N} \boldsymbol{H}_j^T\boldsymbol{H}_j\right)^{-1}\left(\bigodot_{\substack{j=1\\ j\neq d}}^{N} \boldsymbol{H}_j\right)^T\boldsymbol{Y}_{(d)}.$$

As we can see, the Gram matrix is computed efficiently by exploiting the structure, and its Cholesky decomposition can be reused. The most expensive operation is actually the computation of $\left(\odot_{j\neq d}\boldsymbol{H}_j\right)^T\boldsymbol{Y}_{(d)}$, but very efficient algorithms for this (that work without explicitly forming the Khatri-Rao product and the d-mode matricization) are available [17]–[22]. If we were to adopt the QR decomposition approach, however, none of these methods could be applied.

In summary, least squares is a very mature technique with many favorable properties that render the ALS algorithm very efficient. On the other hand, most of the algorithms that deal with problems with constraints on the factors or different loss measures do not inherit these good properties. The goal of this paper is to propose an AO-based algorithmic framework, which can easily handle many types of constraints on the latent factors and many loss functions, with per-iteration complexity essentially the same as the complexity of an ALS step.

B. The Convergence of AO

Consider the following (usually non-convex) optimization problem with variables separated into N blocks, each with its own constraint set

$$\begin{aligned} &\underset{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N}{\text{minimize}} \quad f\left(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N\right) \\ &\text{subject to} \quad \boldsymbol{x}_d \in \mathcal{X}_d, \quad \forall d = 1,\ldots,N. \end{aligned} \tag{4}$$

A classical AO method called block coordinate descent (BCD) cyclically updates $\boldsymbol{x}_d$ via solving

$$\begin{aligned} &\underset{\boldsymbol{\xi}}{\text{minimize}} \quad f\left(\boldsymbol{x}_1^{r+1},\ldots,\boldsymbol{x}_{d-1}^{r+1},\boldsymbol{\xi},\boldsymbol{x}_{d+1}^{r},\ldots,\boldsymbol{x}_N^{r}\right) \\ &\text{subject to} \quad \boldsymbol{\xi} \in \mathcal{X}_d, \end{aligned} \tag{5}$$

at the $(r+1)$-th iteration [23, Sec. 2.7]. Obviously, this will decrease the objective function monotonically. If some additional assumptions are satisfied, then we can have stronger convergence claims [23, Proposition 2.7.1]. Simply put, if the sub-problem (5) is convex and has a *unique* solution, then every limit point is a stationary point; furthermore, if $\mathcal{X}_1,\ldots,\mathcal{X}_N$ are

all compact, which implies that the sequence generated by BCD is bounded, then BCD is guaranteed to converge to a stationary point, even if (4) is non-convex [24].

In many cases (5) is convex, but the uniqueness of the solution is very hard to guarantee. A special case that does not require uniqueness, first noticed by Grippo and Sciandrone [25], is when $N = 2$. On hindsight, this can be explained by the fact that for $N = 2$, BCD coincides with the so-called maximum block improvement (MBI) algorithm [26], which converges under very mild conditions. However, instead of updating the blocks cyclically, MBI only updates the one block that decreases the objective the most, thus the per-iteration complexity is $(N-1)$ times higher than BCD; therefore MBI is not commonly used in practice when N is large.

Another way to ensure convergence, proposed by Razaviyayn *et al.* [27], is as follows. Instead of updating $\boldsymbol{x}_d$ as the solution of (5), the update is obtained by solving a majorized version of (5), called the block successive upper-bound minimization (BSUM). The convergence of BSUM is essentially the same, but now we can deliberately design the majorizing function to ensure that the solution is unique. One simple way to do this is to put a proximal regularization term

$$\begin{aligned} &\underset{\boldsymbol{\xi}}{\text{minimize}} f\left(\boldsymbol{x}_1^{r+1},\ldots,\boldsymbol{x}_{d-1}^{r+1},\boldsymbol{\xi},\boldsymbol{x}_{d+1}^{r},\ldots,\boldsymbol{x}_N^{r}\right)+\frac{\mu_d^r}{2}\|\boldsymbol{\xi}-\boldsymbol{x}_d^r\|^2 \\ &\text{subject to} \quad \boldsymbol{\xi}\in\mathcal{X}_d, \end{aligned} \tag{6}$$

for some $\mu_d^r > 0$ at every iteration for each block, where $\boldsymbol{x}_d^r$ is the update of $\boldsymbol{x}_d$ from the previous iteration. If (5) is convex, then (6) is strongly convex, which gives a unique minimizer. Thus, the algorithm is guaranteed to converge to a stationary point, as long as the sequence generated by the algorithm is bounded. In the context of ALS, this type of update strategy is independently shown in [28] to converge to a stationary point. Similar results are also proved in [29], where the authors used a different majorization for constrained matrix/tensor factorization; we shall compare with them in numerical experiments.

III. Solving the Sub-Problems Using ADMM

The AO algorithm framework is usually adopted when each of the sub-problems can be solved efficiently. This is indeed the case for the ALS algorithm, since each update is in closed-form. For the general factorization problem (1), we denote the sub-problem as

$$\underset{\boldsymbol{H}}{\text{minimize}} \quad l\left(\boldsymbol{Y}-\boldsymbol{W}\boldsymbol{H}^T\right)+r\left(\boldsymbol{H}\right). \tag{7}$$

For the matrix case, this is simply the sub-problem for the right factor, and one can easily figure out the update of the left factor by transposing everything; for the tensor case, this becomes the update of $\boldsymbol{H}_d$ by setting $\boldsymbol{Y}=\boldsymbol{Y}_{(d)}$ and $\boldsymbol{W}=\odot_{j\neq d}\boldsymbol{H}_j$. This is for ease of notation, as these matricizations and Khatri-Rao products need not be actually computed explicitly. Also notice that this is the sub-problem for the BCD algorithm, and for better convergence we may want to add a proximal regularization term to (7), which is very easy to handle, thus omitted here.

We propose to use the alternating direction method of multipliers (ADMM) to solve (7). ADMM, if used in the right way, inherits a lot of the good properties that appeared in each update of the ALS method. Furthermore, the AO framework naturally provides good initializations for ADMM, which further accelerates its convergence for the subproblem. As a preview, the implicit message here is that *closed-form solution is not necessary for computational efficiency*, as we will explain later. After a brief introduction of ADMM, we first apply it to (7) which has least-squares loss, and then generalize it to universal loss measures.

A. Alternating Direction Method of Multipliers

ADMM solves convex optimization problems that can be written in the form

$$\begin{aligned} &\underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimize}} \quad f\left(\boldsymbol{x}\right)+g\left(\boldsymbol{z}\right) \\ &\text{subject to} \quad \boldsymbol{A}\boldsymbol{x}+\boldsymbol{B}\boldsymbol{z}=\boldsymbol{c}, \end{aligned}$$

by iterating the following updates

$$\begin{aligned} \boldsymbol{x} &\leftarrow \arg\min_{\boldsymbol{x}} f\left(\boldsymbol{x}\right)+(\rho/2)\left\|\boldsymbol{A}\boldsymbol{x}+\boldsymbol{B}\boldsymbol{z}-\boldsymbol{c}+\boldsymbol{u}\right\|_2^2, \\ \boldsymbol{z} &\leftarrow \arg\min_{\boldsymbol{z}} g\left(\boldsymbol{z}\right)+(\rho/2)\left\|\boldsymbol{A}\boldsymbol{x}+\boldsymbol{B}\boldsymbol{z}-\boldsymbol{c}+\boldsymbol{u}\right\|_2^2, \\ \boldsymbol{u} &\leftarrow \boldsymbol{u}+\left(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{B}\boldsymbol{z}-\boldsymbol{c}\right), \end{aligned}$$

where $\boldsymbol{u}$ is a scaled version of the dual variables corresponding to the equality constraint $\boldsymbol{A}\boldsymbol{x}+\boldsymbol{B}\boldsymbol{z}=\boldsymbol{c}$, and ρ is specified by the user.

A comprehensive review of the ADMM algorithm can be found in [30] and the references therein. The beauty of ADMM is that it converges under mild conditions (in the convex case), while artful splitting of the variables into the two blocks $\boldsymbol{x}$ and $\boldsymbol{z}$ can yield very efficient updates, and/or distributed implementation. Furthermore, if f is strongly convex and Lipschitz continuous, then linear convergence of ADMM can be achieved; cf. guidelines on the optimal step-size ρ in [31, Sec. 9.3], and [32] for an analysis of ADMM applied to quadratic programming.

B. Least-Squares Loss

We start by considering $l\left(\cdot\right)$ in (7) to be the least-squares loss $(1/2)\left\|\cdot\right\|_F^2$. The problem is reformulated by introducing a $k\times n$ auxiliary variable $\tilde{\boldsymbol{H}}$

$$\begin{aligned} &\underset{\boldsymbol{H},\tilde{\boldsymbol{H}}}{\text{minimize}} \quad \frac{1}{2}\left\|\boldsymbol{Y}-\boldsymbol{W}\tilde{\boldsymbol{H}}\right\|_F^2+r\left(\boldsymbol{H}\right) \\ &\text{subject to} \quad \boldsymbol{H}=\tilde{\boldsymbol{H}}^T. \end{aligned} \tag{8}$$

It is easy to adopt the ADMM algorithm and derive the following iterates:

$$\begin{aligned} \tilde{\boldsymbol{H}} &\leftarrow \left(\boldsymbol{W}^T\boldsymbol{W}+\rho\boldsymbol{I}\right)^{-1}\left(\boldsymbol{W}^T\boldsymbol{Y}+\rho(\boldsymbol{H}+\boldsymbol{U})^T\right), \\ \boldsymbol{H} &\leftarrow \arg\min_{\boldsymbol{H}} r\left(\boldsymbol{H}\right)+\frac{\rho}{2}\left\|\boldsymbol{H}-\tilde{\boldsymbol{H}}^T+\boldsymbol{U}\right\|_F^2, \\ \boldsymbol{U} &\leftarrow \boldsymbol{U}+\boldsymbol{H}-\tilde{\boldsymbol{H}}^T. \end{aligned} \tag{9}$$

One important observation is that, throughout the iterations the same matrix $\boldsymbol{W}^T\boldsymbol{Y}$ and matrix inverse $\left(\boldsymbol{W}^T\boldsymbol{W}+\rho\boldsymbol{I}\right)^{-1}$ are used. Therefore, to save computations, we can cache $\boldsymbol{W}^T\boldsymbol{Y}$ and the Cholesky decomposition of $\boldsymbol{W}^T\boldsymbol{W}+\rho\boldsymbol{I}=\boldsymbol{L}\boldsymbol{L}^T$. Then the update of $\tilde{\boldsymbol{H}}$ is dominated by one forward substitution and one backward substitution, resulting in a complexity of $\mathcal{O}\left(k^2n\right)$.

The update of $\boldsymbol{H}$ is the so-called *proximity operator* of the function $(1/\rho)\,r\left(\cdot\right)$ around point $(\tilde{\boldsymbol{H}}^T-\boldsymbol{U})$, and in particular if $r\left(\cdot\right)$ is the indicator function of a convex set, then the update of $\boldsymbol{H}$ becomes a projection operator, a special case of the proximity operator. For a lot of regularizations/constraints, especially those that are often used in matrix/tensor factorization problems, the update of $\boldsymbol{H}$ boils down to element-wise updates, i.e., costing $\mathcal{O}\left(kn\right)$ flops. Here we list some of the most commonly used constraints/regularizations in the matrix factorization problem, and refer the reader to [33, Sec. 6]. For simplicity of notation, let us define $\bar{\boldsymbol{H}}=\tilde{\boldsymbol{H}}^T-\boldsymbol{U}$.

- Non-negativity. In this case $r\left(\cdot\right)$ is the indicator function of $\mathbb{R}_+$, and the update is simply zeroing out the negative values of $\bar{\boldsymbol{H}}$. In fact, any element-wise bound constraints can be handled similarly, since element-wise projection is trivial.
- Lasso regularization. For $r\left(\boldsymbol{H}\right)=\lambda\|\boldsymbol{H}\|_1$, the sparsity inducing regularization, the update is the well-known *soft-thresholding* operator: $h_{ij}=\left[1-\left(\lambda/\rho\right)|\bar{h}_{ij}|^{-1}\right]_+\bar{h}_{ij}$. The element-wise thresholding can also be converted to block-wise thresholding if one wants to impose structured sparsity, leading to the group Lasso regularization.
- Simplex constraint. In some probabilistic model analysis we need to constrain the columns or rows to be element-wise non-negative and sum up to one. As described in [34], this projection can be done with a randomized algorithm with linear-time complexity on average.
- Smoothness regularization. We can encourage the columns of $\boldsymbol{H}$ to be smooth by adding the regularization $r\left(\boldsymbol{H}\right)=\left(\lambda/2\right)\|\boldsymbol{T}\boldsymbol{H}\|_F^2$ where $\boldsymbol{T}$ is obtained from an $n\times n$ tri-diagonal matrix with 2 on the diagonal and -1 on the super- and sub-diagonal by removing its first and last row. Its proximity operator is given by $\boldsymbol{H}=\rho\left(\lambda\boldsymbol{T}^T\boldsymbol{T}+\rho\boldsymbol{I}\right)^{-1}\bar{\boldsymbol{H}}$. Although it involves a large matrix inversion, notice that it has a fixed bandwidth of 2, thus can be efficiently calculated in $\mathcal{O}\left(kn\right)$ time [35, Sec. 4.3].
- It is also possible to define projections onto non-convex constraints, for example cardinality constraints can be handled by hard thresholding (as opposed to soft thresholding for lasso regularization). However, ADMM is not guaranteed to converge to the conditionally optimal solution in this case, therefore non-convex constraints are not further discussed in this paper. We only mention in passing that, in the cursory experiments that we conducted for this case, AO-ADMM performance is not bad compared to the alternatives.

We found empirically that by setting $\rho=\|\boldsymbol{W}\|_F^2/k$, the ADMM iterates for the regularized least-squares problem (8) converge very fast. This choice of ρ can be seen as an approximation to the optimal ρ given in [31], but much cheaper to obtain. With a good initialization, naturally provided by the AO framework, the update of $\boldsymbol{H}$ usually does not take more than 5 or 10 ADMM iterations, and very soon reduces down to only 1 iteration. The proposed algorithm for the sub-problem (8) is summarized in Algorithm 1. As we can see, the pre-calculation step takes $\mathcal{O}\left(k^2m+k^3\right)$ flops to form the Cholesky decomposition, and $\mathcal{O}\left(mnk\right)$ flops to form $\boldsymbol{F}$. Notice that these are actually the only computations in Algorithm 1 that involve $\boldsymbol{W}$ and $\boldsymbol{Y}$, which implies that in the tensor case, all the tricks to compute $\boldsymbol{W}^T\boldsymbol{W}$ and $\boldsymbol{W}^T\boldsymbol{Y}$ can be applied here, and then we do not need to worry about them anymore. The computational load of each ADMM iteration is dominated by the $\tilde{\boldsymbol{H}}$-update, with complexity $\mathcal{O}\left(k^2n\right)$.

Algorithm 1: Solve (8) using ADMM.

Input: $\boldsymbol{Y}$, $\boldsymbol{W}$, $\boldsymbol{H}$, $\boldsymbol{U}$, k
Initialize $\boldsymbol{H}$ and $\boldsymbol{U}$;
$\boldsymbol{G}=\boldsymbol{W}^T\boldsymbol{W}$;
$\rho=\mathrm{trace}(\boldsymbol{G})/k$;
Calculate $\boldsymbol{L}$ from the Cholesky decomposition of $\boldsymbol{G}+\rho\boldsymbol{I}=\boldsymbol{L}\boldsymbol{L}^T$;
$\boldsymbol{F}=\boldsymbol{W}^T\boldsymbol{Y}$;
repeat
  $\tilde{\boldsymbol{H}}\leftarrow(\boldsymbol{L}^T)^{-1}\boldsymbol{L}^{-1}(\boldsymbol{F}+\rho(\boldsymbol{H}+\boldsymbol{U})^T)$ using forward/backward substitution ;
  $\boldsymbol{H}\leftarrow\arg\min_{\boldsymbol{H}}r(\boldsymbol{H})+\frac{\rho}{2}\|\boldsymbol{H}-\tilde{\boldsymbol{H}}^T+\boldsymbol{U}\|_F^2$;
  $\boldsymbol{U}\leftarrow\boldsymbol{U}+\boldsymbol{H}-\tilde{\boldsymbol{H}}^T$;
until $r<\varepsilon$ and $s<\varepsilon$; r and s defined in (10) and (11)
return $\boldsymbol{H}$ and $\boldsymbol{U}$.

It is interesting to compare Algorithm 1 with an update of the ALS algorithm, whose complexity is essentially the same as the pre-calculation step plus one iteration. For a small number of ADMM iterations, the complexity of Algorithm 1 is of the same order as an ALS step.

For declaring termination, we adopted the general termination criterion described in [30, Sec. 3.3.1]. After some calibration, we define the relative primal residual

$$r=\left\|\boldsymbol{H}-\tilde{\boldsymbol{H}}^T\right\|_F^2/\|\boldsymbol{H}\|_F^2, \tag{10}$$

and the relative dual residual

$$s=\|\boldsymbol{H}-\boldsymbol{H}_0\|_F^2/\|\boldsymbol{U}\|_F^2, \tag{11}$$

where $\boldsymbol{H}_0$ is $\boldsymbol{H}$ from the previous ADMM iteration, and terminate Algorithm 1if both of them are smaller than some threshold.

Furthermore, if the BSUM framework is adopted, we need to solve a proximal regularized version of (8), and that term can easily be absorbed into the update of $\tilde{\boldsymbol{H}}$.

C. General Loss

Now let us derive an ADMM algorithm to solve the more general problem (7). For this case, we reformulate the problem

by introducing two auxiliary variables $\tilde{\boldsymbol{H}}$ and $\widetilde{\boldsymbol{Y}}$

$$\begin{aligned} \underset{\boldsymbol{H},\tilde{\boldsymbol{H}},\widetilde{\boldsymbol{Y}}}{\text{minimize}} \quad & l\left(\boldsymbol{Y}-\widetilde{\boldsymbol{Y}}\right)+r\left(\boldsymbol{H}\right) \\ \text{subject to} \quad & \boldsymbol{H}=\tilde{\boldsymbol{H}}^{T},\widetilde{\boldsymbol{Y}}=\boldsymbol{W}\tilde{\boldsymbol{H}}. \end{aligned} \tag{12}$$

To apply ADMM, let $\tilde{\boldsymbol{H}}$ be the first block, and $(\widetilde{\boldsymbol{Y}},\boldsymbol{H})$ be the second block, and notice that in the second block update $\widetilde{\boldsymbol{Y}}$ and $\boldsymbol{H}$ can in fact be updated independently. This yields the following iterates:

$$\begin{aligned} &\tilde{\boldsymbol{H}}\leftarrow\left(\boldsymbol{W}^{T}\boldsymbol{W}+\rho\boldsymbol{I}\right)^{-1}\left(\boldsymbol{W}^{T}\left(\widetilde{\boldsymbol{Y}}+\boldsymbol{V}\right)+\rho(\boldsymbol{H}+\boldsymbol{U})^{T}\right) \\ &\begin{cases}\boldsymbol{H}\leftarrow\arg\min\limits_{\boldsymbol{H}} r\left(\boldsymbol{H}\right)+\frac{\rho}{2}\left\|\boldsymbol{H}-\tilde{\boldsymbol{H}}^{T}+\boldsymbol{U}\right\|_{F}^{2},\\ \widetilde{\boldsymbol{Y}}\leftarrow\arg\min\limits_{\widetilde{\boldsymbol{Y}}} l\left(\boldsymbol{Y}-\widetilde{\boldsymbol{Y}}\right)+\frac{1}{2}\left\|\widetilde{\boldsymbol{Y}}-\boldsymbol{W}\tilde{\boldsymbol{H}}+\boldsymbol{V}\right\|_{F}^{2},\end{cases} \\ &\begin{cases}\boldsymbol{U}\leftarrow\boldsymbol{U}+\boldsymbol{H}-\tilde{\boldsymbol{H}}^{T},\\ \boldsymbol{V}\leftarrow\boldsymbol{V}+\widetilde{\boldsymbol{Y}}-\boldsymbol{W}\tilde{\boldsymbol{H}}.\end{cases}\end{aligned} \tag{13}$$

where $\boldsymbol{U}$ is the scaled dual variable corresponding to the constraint $\boldsymbol{H}=\tilde{\boldsymbol{H}}^{T}$, and $\boldsymbol{V}$ is the scaled dual variable corresponding to the equality constraint $\widetilde{\boldsymbol{Y}}=\boldsymbol{W}\tilde{\boldsymbol{H}}$. Notice that we set the penalty parameter ρ corresponding to the second constraint to be 1, since it works very well in practice, and also leads to very intuitive results for some loss functions. This can also be interpreted as first pre-conditioning this constraint to be $\frac{1}{\sqrt{\rho}}\widetilde{\boldsymbol{Y}}=\frac{1}{\sqrt{\rho}}\boldsymbol{W}\tilde{\boldsymbol{H}}$, and then a common ρ is used. Again we set $\rho=\|\boldsymbol{W}\|_{F}^{2}/k$.

As we can see, the update of $\tilde{\boldsymbol{H}}$ is simply a linear least squares problem, and all the previous discussion about caching the Cholesky decomposition applies. It is also easy to absorb an additional proximal regularization term into the update of $\tilde{\boldsymbol{H}}$, if the BSUM framework is adopted. The update of $\widetilde{\boldsymbol{Y}}$ is (similar to the update of $\boldsymbol{H}$) a proximity operator, and since almost all loss functions we use are element-wise, the update of $\widetilde{\boldsymbol{Y}}$ is also very easy. The updates for some of the most commonly used non-least-squares loss functions are listed below. For simplicity, we define $\overline{\boldsymbol{Y}}=\boldsymbol{W}\tilde{\boldsymbol{H}}-\boldsymbol{V}$, similar to the previous sub-section.

- Missing values. In the case that only a subset of the entries in $\boldsymbol{Y}$ are available, a common way to handle this is to simply fit the low-rank model only to the available entries. Let $\mathcal{A}$ denote the index set of the available values in $\boldsymbol{Y}$, then the loss function becomes $l(\boldsymbol{Y}-\widetilde{\boldsymbol{Y}})=\frac{1}{2}\sum_{(i,j)\in\mathcal{A}}\left(y_{ij}-\tilde{y}_{ij}\right)^{2}$. Thus, the update of $\widetilde{\boldsymbol{Y}}$ in (13) becomes
$$\tilde{y}_{ij}=\begin{cases}\frac{1}{2}\left(y_{ij}+\overline{y}_{ij}\right), & (i,j)\in\mathcal{A},\\ \overline{y}_{ij}, & \text{otherwise.}\end{cases}$$
- Robust fitting. In the case that data entries are not uniformly corrupted by noise but only sparingly corrupted by outliers, or when the noise is dense but heavy-tailed (e.g., Laplacian-distributed), we can use the ℓ_1 norm as the loss function for robust (resp. maximum-likelihood) fitting, i.e., $l(\boldsymbol{Y}-\widetilde{\boldsymbol{Y}})=\|\boldsymbol{Y}-\widetilde{\boldsymbol{Y}}\|_{1}$. This is similar to the ℓ_1 regularization, and the element-wise update is
$$\tilde{y}_{ij}=\begin{cases}y_{ij}, & |\overline{y}_{ij}-y_{ij}|\leq 1,\\ \overline{y}_{ij}-1, & \overline{y}_{ij}-y_{ij}>1,\\ \overline{y}_{ij}+1, & \overline{y}_{ij}-y_{ij}<-1.\end{cases}$$
- Huber fitting. Another way to deal with possible outliers in $\boldsymbol{Y}$ is to use the Huber function to measure the loss $l(\boldsymbol{Y}-\widetilde{\boldsymbol{Y}})=\sum_{i,j}\phi_{\lambda}(y_{ij}-\tilde{y}_{ij})$ where
$$\phi_{\lambda}\left(z\right)=\begin{cases}\frac{1}{2}z^{2}, & |z|\leq\lambda,\\ \lambda|z|-\frac{1}{2}\lambda^{2}, & \text{otherwise.}\end{cases}$$
The element-wise closed-form update is
$$\tilde{y}_{ij}=\begin{cases}\frac{1}{2}\left(\overline{y}_{ij}+y_{ij}\right), & |\overline{y}_{ij}-y_{ij}|\leq 2\lambda,\\ \overline{y}_{ij}-\lambda, & \overline{y}_{ij}-y_{ij}>2\lambda,\\ \overline{y}_{ij}+\lambda, & \overline{y}_{ij}-y_{ij}<-2\lambda.\end{cases}$$
- Kullback-Leibler divergence. A commonly adopted loss function for non-negative integer data is the Kullback-Leibler (K-L) divergence defined as
$$D\left(\boldsymbol{Y}||\widetilde{\boldsymbol{Y}}\right)=\sum_{i,j}\left(y_{ij}\log\frac{y_{ij}}{\tilde{y}_{ij}}-y_{ij}+\tilde{y}_{ij}\right)$$
for which the proximity operator is
$$\widetilde{\boldsymbol{Y}}=\frac{1}{2}\left(\left(\overline{\boldsymbol{Y}}-1\right)+\sqrt{\left(\overline{\boldsymbol{Y}}-1\right)^{2}+4\boldsymbol{Y}}\right),$$
where all the operations are taken element-wise [36]. Furthermore, the K-L divergence is a special case of certain families of divergence functions, such as α-divergence and β-divergence [37], whose corresponding updates are also very easy to derive (boil down to the proximity operator of a scalar function).

An interesting observation is that if the loss function is in fact the least-squares loss, the matrix $(\widetilde{\boldsymbol{Y}}+\boldsymbol{V})$ that $\tilde{\boldsymbol{H}}$ is trying to fit in (13) is the data matrix $\boldsymbol{Y}$ *per se*. Therefore, the update rule (13) boils down to the update rule (9) in the least-squares loss case, with some redundant updates of $\widetilde{\boldsymbol{Y}}$ and $\boldsymbol{V}$. The detailed ADMM algorithm for (12) is summarized in Algorithm 2. We use the same termination criterion as in Algorithm 1.

Everything seems to be in place to seamlessly move from the least-squares loss to arbitrary loss. Nevertheless, closer scrutiny reveals that some compromises must be made to take this leap. One relatively minor downside is that with a general loss function we may lose the linear convergence rate of ADMM—albeit with the good initialization naturally provided by the AO framework and our particular choice of ρ, it still converges very fast in practice. The biggest drawback is that, by introducing the auxiliary variable $\widetilde{\boldsymbol{Y}}$ and its dual variable $\boldsymbol{V}$, the big matrix product $\boldsymbol{W}^{T}(\widetilde{\boldsymbol{Y}}+\boldsymbol{V})$ must be re-computed in each ADMM iteration, whereas in the previous case one only needs to compute $\boldsymbol{W}^{T}\boldsymbol{Y}$ once. This is the price we must pay; but it can be moderated by controlling the maximum number of ADMM iterations.

Scalability considerations: As big data analytics become increasingly common, it is important to keep scalability issues in mind as we develop new analysis methodologies and algorithms.

Algorithm 2: Solve (12) using ADMM.

Input: $\boldsymbol{Y}$, $\boldsymbol{W}$, $\boldsymbol{H}$, $\boldsymbol{U}$, $\tilde{\boldsymbol{Y}}$, $\boldsymbol{V}$, k
Initialize $\boldsymbol{H}$, $\boldsymbol{U}$, $\tilde{\boldsymbol{Y}}$, and $\boldsymbol{V}$;
$\boldsymbol{G} = \boldsymbol{W}^T\boldsymbol{W}$;
$\rho = \text{trace}(\boldsymbol{G})/k$;
Calculate $\boldsymbol{L}$ from the Cholesky decomposition of $\boldsymbol{G} + \rho\boldsymbol{I} = \boldsymbol{L}\boldsymbol{L}^T$;
repeat
  $\tilde{\boldsymbol{H}} \leftarrow (\boldsymbol{L}^T)^{-1}\boldsymbol{L}^{-1}(\boldsymbol{W}^T(\tilde{\boldsymbol{Y}} + \boldsymbol{V}) + \rho(\boldsymbol{H} + \boldsymbol{U})^T)$ using forward/backward substitution ;
  $\boldsymbol{H} \leftarrow \arg\min_{\boldsymbol{H}} r(\boldsymbol{H}) + \frac{\rho}{2}\|\boldsymbol{H} - \tilde{\boldsymbol{H}}^T + \boldsymbol{U}\|_F^2$;
  $\tilde{\boldsymbol{Y}} \leftarrow \arg\min_{\tilde{\boldsymbol{Y}}}\ l(\boldsymbol{Y} - \tilde{\boldsymbol{Y}}) + \frac{1}{2}\|\tilde{\boldsymbol{Y}} - \boldsymbol{W}\tilde{\boldsymbol{H}} + \boldsymbol{V}\|_F^2$;
  $\boldsymbol{U} \leftarrow \boldsymbol{U} + \boldsymbol{H} - \tilde{\boldsymbol{H}}^T$;
  $\boldsymbol{V} \leftarrow \boldsymbol{V} + \tilde{\boldsymbol{Y}} - \boldsymbol{W}\tilde{\boldsymbol{H}}$;
until $r < \varepsilon$ and $s < \varepsilon$; r and s defined in (10) and (11)
return $\boldsymbol{H}$, $\boldsymbol{U}$, $\tilde{\boldsymbol{Y}}$, and $\boldsymbol{V}$.

Algorithm 3: Proposed algorithm for (1).

Initialize $\boldsymbol{H}_1$,...,$\boldsymbol{H}_N$;
Initialize $\boldsymbol{U}_1$,...,$\boldsymbol{U}_N$ to be all zero matrices;
if *least-squares loss* **then**
  repeat
    for $d = 1, ..., N$ **do**
      $\boldsymbol{Y} = \boldsymbol{Y}_{(d)}$ and $\boldsymbol{W} = \odot_{j\neq d}\boldsymbol{H}_j$; // not necessarily formed explicitly
      update $\boldsymbol{H}_d$ and $\boldsymbol{U}_d$ using Alg. 1 initialized with the previous $\boldsymbol{H}_d$ and $\boldsymbol{U}_d$;
    end
    update μ if necessary ; // refer to (14)
  until *some termination criterion is reached*;
else
  Initialize $\tilde{\underline{\boldsymbol{Y}}} \leftarrow \underline{\boldsymbol{Y}}$, $\underline{\boldsymbol{V}} \leftarrow \underline{\boldsymbol{0}}$;
  repeat
    for $d = 1, ..., N$ **do**
      $\boldsymbol{Y} = \boldsymbol{Y}_{(d)}$ and $\boldsymbol{W} = \odot_{j\neq d}\boldsymbol{H}_j$; // not necessarily formed explicitly
      update $\boldsymbol{H}_d$,$\boldsymbol{U}_d$,$\tilde{\boldsymbol{Y}}_{(d)}$,$\boldsymbol{V}_{(d)}$ using Alg. 2 initialized with the previous $\boldsymbol{H}_d$,$\boldsymbol{U}_d$,$\tilde{\boldsymbol{Y}}_{(d)}$,$\boldsymbol{V}_{(d)}$;
    end
    update μ if necessary ; // refer to (14)
  until *some termination criterion is reached*;
end

Big data $\underline{\boldsymbol{Y}}$ is usually stored as a sparse array, i.e., a list of (`index,value`) pairs, with the unlisted entries regarded as zeros or missing. With the introduction of $\widetilde{\boldsymbol{Y}}$ and $\boldsymbol{V}$, both of size($\underline{\boldsymbol{Y}}$), one hopes to be able to avoid dense operations. Fortunately, for some commonly used loss functions, this is possible. Notice that by defining $\overline{\boldsymbol{Y}} = \boldsymbol{W}\tilde{\boldsymbol{H}} - \boldsymbol{V}$, the $\boldsymbol{V}$-update essentially becomes

$$\boldsymbol{V} \leftarrow \widetilde{\boldsymbol{Y}} - \overline{\boldsymbol{Y}},$$

which means a significant portion of entries in $\boldsymbol{V}$ are constants—0 if the entries are regarded as missing, ± 1 or $\pm\lambda$ in the robust fitting or Huber fitting case if the entries are regarded as "corrupted"—thus they can be efficiently stored as a sparse array. As for $\widetilde{\boldsymbol{Y}}$, one can simply generate it "on-the-fly" using the closed-form we provided earlier (notice that $\overline{\boldsymbol{Y}}$ has the memory-efficient "low-rank plus sparse" structure). The only occasion that $\widetilde{\boldsymbol{Y}}$ is needed is when computing $\boldsymbol{W}^T\widetilde{\boldsymbol{Y}}$.

IV. Summary of the Proposed Algorithm

We propose to use Algorithm 1or 2 as the core sub-routine for alternating optimization. The proposed "universal" multi-linear factorization algorithm is summarized as Algorithm 3. A few remarks on implementing Algorithm 3 are in order.

Since each factor $\boldsymbol{H}_d$ is updated in a cyclic fashion, one expects that after a certain number of cycles $\boldsymbol{H}_d$ (and its dual variable $\boldsymbol{U}_d$) obtained in the previous iteration will not be very far away from the update for the current iteration. In this sense, the outer AO framework naturally provides a good initial point to the inner ADMM iteration. With this warm-start strategy, the optimality gap for the sub-problem is then bounded by the per-step improvement of the AO algorithm, which is small. This mode of operation is crucial for insuring the efficiency of Algorithm 3. Our experiments suggest that soon after an initial transient stage, the sub-problems can be solved in just one ADMM iteration (with reasonable precision).

Similar ideas can be used for $\widetilde{\boldsymbol{Y}}$ and $\boldsymbol{V}$ in the matrix case if we want to deal with non-least-squares loss, and actually only one copy of them is needed in the updates of both factors. A few different options are available in the tensor case. If memory is not an issue in terms of the size of $\underline{\boldsymbol{Y}}$, a convenient approach that is commonly adopted in ALS implementations is to store all N matricizations $\boldsymbol{Y}_{(1)}, \ldots, \boldsymbol{Y}_{(N)}$, so they are readily available without need for repetitive data re-shuffling during run-time. If this practice is adopted, then it makes sense to also have N copies of $\widetilde{\underline{\boldsymbol{Y}}}$ and $\underline{\boldsymbol{V}}$, in order to save computation. Depending on the size and nature of the data and how it is stored, it may be completely unrealistic to keep multiple copies of the data and the auxiliary variables, at which point our earlier discussion on scalable implementation of Algorithm 2 for big but sparse data can be instrumental.

Sometimes an additional proximal regularization is added to the sub-problems. The benefit is two-fold: it helps the convergence of the AO outer-loop when $N \geq 3$; while for the ADMM inner-loop it improves the conditioning of the sub-problem, which may accelerate the convergence of ADMM, especially in the general loss function case when we do not have strong convexity. The convergence of AO-ADMM is summarized in Proposition 1.

Proposition 1: If the sequence generated by AO-ADMM in Algorithm 3 is bounded, then for

1) $N = 2$,
2) $N > 2, \mu > 0$,

AO-ADMM converges to a stationary point of (1).

Proof: The first case with $\mu = 0$ is covered in [26, Theorem 3.1], and the cases when $\mu > 0$ are covered in [27, Theorem 2]. ■

Note that for $N = 2$, using $\mu = 0$ yields faster convergence than $\mu > 0$. For $N > 2$, i.e., for tensor data, we can update μ as follows

$$\mu \leftarrow 10^{-7} + 0.01 \frac{\left\| \underline{\boldsymbol{Y}} - [\boldsymbol{H}_d]_{d=1}^{N} \right\|}{\|\underline{\boldsymbol{Y}}\|}, \tag{14}$$

which was proposed in [27] for unconstrained tensor factorization, and works very well in our context as well.

The convergence result in Proposition 1 has an additional assumption that the sequence generated by the algorithm is bounded. For unconstrained PARAFAC, diverging components may be encountered during AO iterations [38], [39], but adding Frobenius norm regularization for each matrix factor (with a small weight) ensures that the iterates remain bounded.

As we can see, the ADMM is an appealing sub-routine for alternating optimization, leading to a simple plug-and-play generalization of the workhorse ALS algorithm. Theoretically, they share the same per-iteration complexity if the number of inner ADMM iterations is small, which is true in practice, after an initial transient. Efficient implementation of the overall algorithm should include data-structure-specific algorithms for $\boldsymbol{W}^T\boldsymbol{Y}$ or $(\odot_{j\neq d}\boldsymbol{H}_j)^T\boldsymbol{Y}_{(d)}$, which dominate the per-iteration complexity, and may include parallel/distributed computation along the lines of [40].

Finally, if a non-least-squares loss is to be used, we suggest that the least-squares loss is first employed to get preliminary estimates (using Algorithm 3 calling Algorithm 1) which can then be fed as initialization to run Algorithm 3 calling Algorithm 2. The main disadvantage of Algorithm 2 compared to Algorithm 1 is that the big matrix (or tensor) multiplication $\boldsymbol{W}^T(\widetilde{\boldsymbol{Y}} + \boldsymbol{V})$ needs to be calculated in each ADMM iteration. Therefore, this strategy can save a significant amount of computations at the initial stage.

V. Case Studies and Numerical Results

In this section we will study some well-known constrained matrix/tensor factorization problems, derive the corresponding update for $\boldsymbol{H}$ in Algorithm 1 or $\boldsymbol{H}$ and $\widetilde{\boldsymbol{Y}}$ in Algorithm 2, and compare it to some of the state-of-the-art algorithms for that problem. In all examples we denote our proposed algorithm as **AO-ADMM**. All experiments are performed in MATLAB 2015a on a Linux server with 32 Xeon 2.00 GHz cores and 128 GB memory.

A. Non-Negative Matrix and Tensor Factorization

Perhaps the most common constraint imposed on the latent factors is non-negativity—which is often supported by physical considerations (e.g., when the latent factors represent chemical concentrations, or power spectral densities) or other prior information, or simply because non-negativity sometimes yields interpretable factors [4]. Due to the popularity and wide range of applications of NMF, numerous algorithms have been proposed for fitting the NMF model, and most of them can be easily generalized to the tensor case. After a brief review of the existing algorithms for NMF, we compare our proposed algorithm to some of the best algorithms reported in the literature to showcase the efficiency of AO-ADMM.

Let us start by considering NMF with least-squares loss, which is the prevailing loss function in practice. By adopting the alternating optimization framework, the sub-problem that emerges for each matrix factor is non-negative (linear) least-squares (NNLS). Some of the traditional methods for NNLS are reviewed in [41] (interestingly, not including ADMM), and most of them have been applied to NMF or non-negative PARAFAC, e.g., the active-set (AS) method [42], [43] and block-principle-pivoting (BPP) [44], [45]. Recall that in the context of the overall multi-linear factorization problem we actually need to solve a large number of (non-negative) least-squares problems sharing the same mixing matrix $\boldsymbol{W}$, and in the unconstrained case this means we only need to calculate the Cholesky factorization of $\boldsymbol{W}^T\boldsymbol{W}$ once. Unfortunately, this good property that enables high efficiency implementation of ALS is not preserved by either AS or BPP. Sophisticated methods that group similar rows to reduce the number of inversions have been proposed [46], although as k grows larger this does not seem appealing in the worst case. Some other methods, like the multiplicative-update (MU) [47] or hierarchical alternating least squares (HALS) [37], ensure that the per-iteration complexity is dominated by calculating $\boldsymbol{W}^T\boldsymbol{W}$ and $\boldsymbol{W}^T\boldsymbol{Y}$, although more outer-loops are needed for convergence. These are actually one step majorization-minimization or block coordinate descent applied to the NNLS problem. An accelerated version of MU and HALS is proposed in [48], which essentially does a few more inner-loops after computing the most expensive $\boldsymbol{W}^T\boldsymbol{Y}$.

ADMM, on the other hand, may not be the fastest algorithm for a single NNLS problem, yet its overhead can be amortized when there are many NNLS problem instances sharing the same mixing matrix, especially if good initialization is readily available. This is in contrast to an earlier attempt to adopt ADMM to NMF [49], which did not use Cholesky caching, warm start, and a good choice of ρ to speed up the algorithm. Furthermore, ADMM can seamlessly incorporate different regularizations as well as non-least-squares loss.

We should emphasize that AO forms the backbone of our proposed algorithm—ADMM is only applied to the sub-problems. There are also algorithms that directly apply an ADMM approach to the whole problem [36], [40], [50]. The per-iteration complexity of those algorithms is also the same as the unconstrained alternating least-squares. However, due to the non-convexity of the whole problem, the loss is not guaranteed to decrease monotonically, unlike alternating optimization. Moreover, both ADMM and AO guarantee that every limit point is a stationary point, but in practice AO almost always converges (as long as the updates stay bounded), which is not the case for ADMM applied to the whole problem.

In another recent line of work [29], a similar idea of using an improved AO framework to ensure convergence is used. When [29] is specialized to non-negative matrix/tensor factorization, each update becomes a simple proximal-gradient step with an

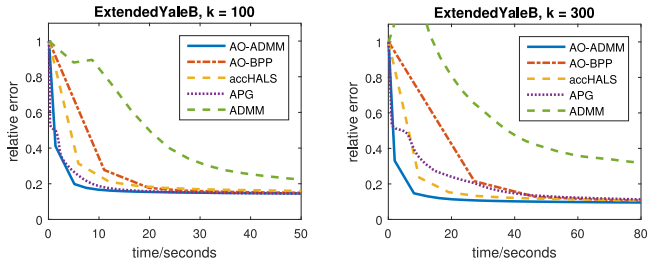


Fig. 1. Convergence of some NMF algorithms on the Extended Yale B dataset.

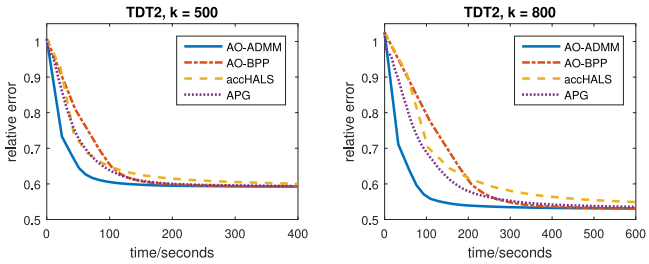


Fig. 2. Convergence of some NMF algorithms on the TDT2 dataset.

extrapolation. The resulting algorithm is also guaranteed to converge (likewise assuming that the iterates remain bounded), but it turns out to be slower than our algorithm, as we will show in our experiments. Some interesting work on non-negative PARAFAC can also been found in [51] and the references therein.

To apply our proposed algorithm to NMF or non-negative PARAFAC with least-squares loss, Algorithm 1 is used to solve the sub-problems, with line 8 customized as

$$\boldsymbol{H} \leftarrow \left[\tilde{\boldsymbol{H}}^T - \boldsymbol{U}\right]_+,$$

i.e., zeroing out the negative values of $(\tilde{\boldsymbol{H}}^T - \boldsymbol{U})$. The tolerance for the ADMM inner-loop is set to 0.01.

1) Non-Negative Matrix Factorization: We compare AO-ADMM with the following algorithms:

- **AO-BPP.** AO using block principle pivoting [44][1];
- **accHALS.** Accelerated HALS [48][2];
- **APG.** Alternating proximal gradient [29][3];
- **ADMM.** ADMM applied to the whole problem [50][4].

AO-BPP and HALS are reported in [44] to outperform other methods, accHALS is proposed in [48] to improve HALS, APG is reported in [29] to outperform AO-BPP, and we include ADMM applied to the whole problem to compare the convergence behavior of AO and ADMM for this non-convex factorization problem.

The aforementioned NMF algorithms are tested on two datasets. One is a dense image data set, the Extended Yale Face Database B[5], of size 32256×1932, where each column is a vectorized 168×192 image of a face, and the dataset is a collection of face images of 29 subjects under various poses and illumination conditions. The other one is the Topic Detection and Tracking 2 (TDT2) text corpus[6], of size 10212×36771, which is a sparse document-term matrix where each entry counts the frequency of a term in one document.

The convergence of the relative error $\|\boldsymbol{Y} - \boldsymbol{W}\boldsymbol{H}^T\|_F / \|\boldsymbol{Y}\|_F$ versus time in seconds for the Extended Yale B dataset is shown in Fig. 1, with $k = 100$ on the left and $k = 300$ on the right; and for the TDT2 dataset in Fig. 2, with $k = 500$ on the left and $k = 800$ on the right. The ADMM algorithm [50] is not included for TDT2 because the code provided online is geared towards imputation of matrices with missing values—it does not treat a sparse input matrix as the full data, unless we fill-in all zeros.

TABLE I
AVERAGED PERFORMANCE OF NMF ALGORITHMS ON SYNTHETIC DATA

Algorithm	$\|\boldsymbol{Y} - \boldsymbol{W}\boldsymbol{H}^T\|_F$	run time	iterations
AO-ADMM	193.1026	21.7 s	86.9
AO-BPP	193.1516	40.9 s	52.2
accHALS	193.1389	26.8 s	187.0
APG	193.1431	25.3 s	240.2
ADMM	193.6808	31.9 s	125.2

We also tested these algorithms on synthetic data. For $m = n = 2000$ and $k = 100$, the true $\boldsymbol{W}$ and $\boldsymbol{H}$ are generated by drawing their elements from an i.i.d. exponential distribution with mean 1, and then 50% of the elements are randomly set to 0. The data matrix $\boldsymbol{Y}$ is then set to be $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{H}^T + \boldsymbol{N}$, where the elements of $\boldsymbol{N}$ are drawn from an i.i.d. Gaussian distribution with variance 0.01. The averaged results of 100 Monte-Carlo trials are shown in Table I. As we can see, AO-based methods are able to attain smaller fitting errors than directly applying ADMM to this non-convex problem, while AO-ADMM provides the most efficient per-iteration complexity.

2) Non-Negative PARAFAC: Similar algorithms are compared in the non-negative PARAFAC case:

- **AO-BPP.** AO using block principle pivoting [45][1];
- **HALS.** Hierarchical alternating least-squares [37][1];
- **APG.** Alternating proximal gradient [29][2];
- **ADMM.** ADMM applied to the whole problem [40];
- **SDF.** Structured data fusion provided by `tensorlab` [52], using "all-at-once" updates based on quasi-Newton or Gauss-Newton method [53], [54].

For our proposed AO-ADMM algorithm, a diminishing proximal regularization term in the form (6) is added to each sub-problem to enhance the overall convergence, with the regularization parameter μ updated as (14).

Two real datasets are being tested: one is a dense CT image dataset[7] of size $260 \times 190 \times 150$, which is a collection of 150 CT images of a female's ankle, each with size 260×190; the other one is a sparse social network dataset—Facebook Wall Posts[8], of size $46952 \times 46951 \times 1592$, that collects the number

[1] http://www.cc.gatech.edu/~hpark/nmfsoftware.php
[2] https://sites.google.com/site/nicolasgillis/code
[3] http://www.math.ucla.edu/~wotaoyin/papers/bcu/matlab.html
[4] http://mcnf.blogs.rice.edu/
[5] http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html
[6] http://www.cad.zju.edu.cn/home/dengcai/Data/TextData.html
[7] http://www.nlm.nih.gov/research/visible/
[8] http://konect.uni-koblenz.de/networks/facebook-wosn-wall

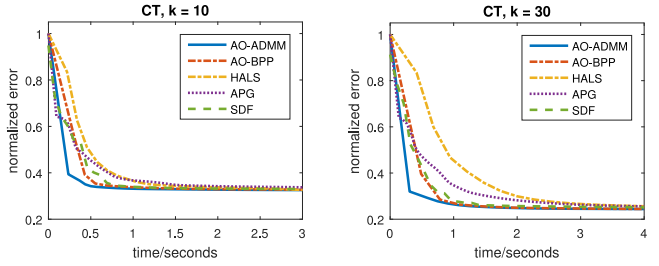


Fig. 3. Convergence of some non-negative PARAFAC algorithms on the CT dataset.

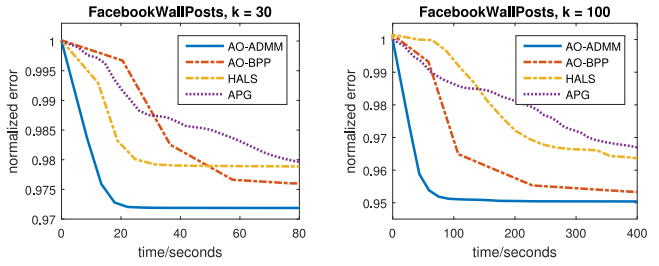


Fig. 4. Convergence of some non-negative PARAFAC algorithms on the Facebook Wall Posts dataset.

of wall posts from one Facebook user to another over a period of 1592 days. The sparse tensor is stored in the `sptensor` format supported by the `tensor_toolbox` [55], and all the aforementioned algorithms use this toolbox to handle sparse tensor data, except SDF, which only accepts the sparse tensor structure defined by `tensorlab`. However, due to the algorithms being used by SDF, the memory requirement exceeded the limit for the latter case, thus it is omitted for the Facebook wall posts dataset.

Similar to the matrix case, the normalized root mean squared error versus time in seconds for the CT dataset is shown in Fig. 3, with $k = 10$ on the left and $k = 30$ on the right, and that for the Facebook Wall Posts data is shown in Fig. 4, with $k = 30$ on the left and $k = 100$ on the right. As we can see, AO-ADMM again converges the fastest, not only because of the efficient per-iteration update from Algorithm 1, but also thanks to the additional proximal regularization to help the algorithm avoid swamps, which are not uncommon in alternating optimization-based algorithms for tensor decomposition.

Monte-Carlo simulations were also conducted using synthetic data for 3-way non-negative tensors with $n_1 = n_2 = n_3 = 500$ and $k = 100$, with the latent factors generated in the same manner as for the previous NMF synthetic data, and the tensor data generated as the low-rank model synthesized from those factors plus i.i.d. Gaussian noise with variance 0.01. The averaged result over 100 trials is given in Table II. As we can see, AO-ADMM again outperforms all other algorithms in all cases considered.

TABLE II
AVERAGED PERFORMANCE OF NON-NEGATIVE PARAFAC ALGORITHMS ON SYNTHETIC DATA

Algorithm	$\|\underline{\boldsymbol{Y}} - [\boldsymbol{H}_1, \boldsymbol{H}_2, \boldsymbol{H}_3]\|$	run time	iterations
AO-ADMM	1117.597	145.2 s	25.1
AO-BPP	1117.728	679.0 s	22.6
HALS	1117.655	1838.7 s	137.7
APG	1117.649	1077.4 s	156.3
ADMM	1156.799	435.9 s	77.2
SDF	1118.427	375.8 s	N/A

B. Constrained Matrix and Tensor Completion

As discussed before, real-world data are often stored as a sparse array, i.e., in the form of (`index,value`) pairs. Depending on the application, the unlisted entries in the array can be treated as zeros, or as not (yet) observed but possibly nonzero. A well-known example of the latter case is the *Netflix prize problem*, which involves an array of movie ratings indexed by customer and movie. The data is extremely sparse, but the fact that a customer did not rate a movie does not mean that the customer's rating of that movie would be zero—and the goal is actually to predict those unseen ratings to provide good movie recommendations.

For matrix data with no constraints on the latent factors, convex relaxation techniques that involve the matrix nuclear norm have been proposed with provable matrix reconstruction bounds [7]. Some attempts have been made to generalize the matrix nuclear norm to tensor data [56], [57], but that boils down to the Tucker model rather than the PARAFAC model that we consider here. A key difference is that Tucker modeling can only hope to impute (recover missing values) in the data, whereas PARAFAC can uniquely recover the latent factors—the important 'dimensions' of consumer preference in this context. Another key difference is that the aforementioned convex relaxation techniques cannot incorporate constraints on the latent factors, which can improve the estimation performance. Taking the Netflix problem as an example, *user-bias* and *movie-bias* terms are often successfully employed in recommender systems; these can be easily subsumed in the factorization formulation by constraining, say, the first column of $\boldsymbol{W}$ and the second column of $\boldsymbol{H}$ to be equal to the all-one vector. Moreover, interpreting each column of $\boldsymbol{W}$ ($\boldsymbol{H}$) as the appeal of a certain movie genre to the different users (movie ratings for a given type of user, respectively), it is natural to constrain the entries of $\boldsymbol{W}$ and $\boldsymbol{H}$ to be non-negative.

When matrix/tensor completion is formulated as a constrained factorization problem using a loss function as in Section III.C, there are traditionally two ways to handle it. One is directly using alternating optimization, although due to the random positions of the missing values, the least-squares problem for each row of $\boldsymbol{H}$ will involve a different subset of the rows of $\boldsymbol{W}$, thus making the update inefficient even in the unconstrained case. A more widely used way is an instance of expectation-maximization (EM): one starts by filling the missing values with zeros, and then iteratively fits a (constrained) low-rank model and imputes the originally missing values with predictions from the interim low-rank model. More recently, an ADMM approach that uses an auxiliary variable for the full data was proposed [50], although if we look carefully at that

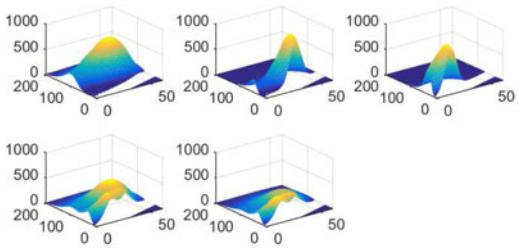


Fig. 5. Illustration of the missing values in the Amino acids fluorescence data.

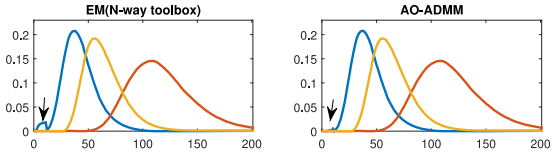


Fig. 6. The emission loadings ($\boldsymbol{H}_2$) produced by the N-way toolbox on the left, which uses EM, and by AO-ADMM on the right.

auxiliary variable, it is exactly equal to the filled-in data given by the EM method.

In fact, the auxiliary variable $\widetilde{\boldsymbol{Y}}$ that we introduce is similar to that of [50], thus also related to the way that EM imputes the missing values—one can treat our method as imputing the missing values per ADMM inner-loop, the method in [50] as imputing per iteration, and EM as imputing after several iterations. However, our proposed AO-ADMM is able to give better results than EM, despite the similarities. As an illustrative example, consider the Amino acids fluorescence data[9], which is a $5 \times 201 \times 61$ tensor known to be generated by a rank-3 non-negative PARAFAC model [58]. However, some of the entries are known to be badly contaminated, and are thus deleted, as shown in Fig. 5. Imposing non-negativity on the latent factors, the emission loadings $\boldsymbol{H}_2$ of the three chemical components provided by the EM method using the N-way toolbox [59] and AO-ADMM are shown in Fig. 6. While both results are satisfactory, AO-ADMM is able to suppress the artifacts caused by the systematically missing values in the original data, as indicated by the arrows in Fig. 6.

We now evaluate our proposed AO-ADMM on a movie rating dataset called MovieLens[10], which consists of 100,000 movie ratings from 943 users on 1682 movies. MovieLens includes 5 sets of 80%–20% splits of the ratings for training and testing, and for each split we fit a matrix factorization model based on the 80% training data, and evaluate the correctness of the model on the 20% testing data. The averaged performance on this 5-fold cross validation is shown in Fig. 7, where we used the mean absolute error (MAE) for comparison with the classical collaborative filtering result [60] (which attains a MAE of 0.73). On the left of Fig. 7, we used the traditional least-squares criterion to fit the available ratings, whereas on the right we used the Kullback-Leibler divergence for fitting, since it is a meaningful statistical model for integer data. For each fitting criterion, we compared the performance by imposing Tikhonov

[9]http://www.models.kvl.dk/Amino_Acid_fluo

[10]http://grouplens.org/datasets/movielens/

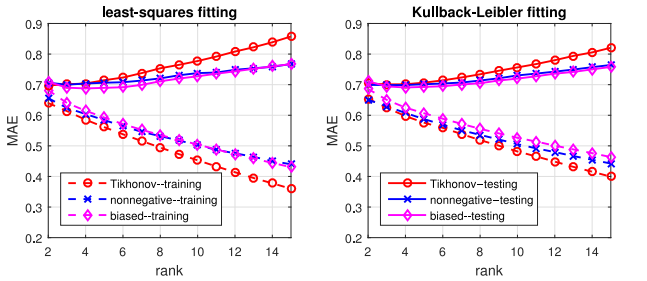


Fig. 7. Training and testing mean absolute error (MAE) versus model rank of the MovieLens data, averaged over a 5-fold cross validation, comparing least-squares fitting (on the left) and Kullback-Leibler fitting (on the right), with Tikhonov regularization, non-negativity constraint, or non-negativity with biases on the latent factors.

regularization $(\lambda/2)\,\|\cdot\|_F^2$ with $\lambda = 0.1$, or non-negativity, or non-negativity with biases (i.e., in addition constraining the first column of $\boldsymbol{W}$ and second column of $\boldsymbol{H}$ to be all ones). Some observations are as follows:

- Low-rank indeed seems to be a good model for this movie rating data, and the right rank seems to be 4 or 5, higher rank leads to over-fitting, as evident from Fig. 7;
- Imposing non-negativity reduces the over-fitting at higher ranks, whereas the fitting criterion does not seem to be playing a very important role in terms of performance;
- By adding biases, the best case prediction MAE at rank 4 is less than 0.69, an approximately 6% improvement over the best result reported in [60].

Notice that our aim here is to showcase how AO-ADMM can be used to explore possible extensions to the matrix completion problem formulation, rather than come up with the best recommender system method, which would require significant exploration in its own right. We believe with the versatileness of AO-ADMM, researchers can easily test various models for matrix/tensor completion, and quickly narrow down the one that works the best for their specific application.

C. Dictionary Learning

Many natural signals can be represented as an (approximately) sparse linear combination of some (possibly over-complete) basis, for example the Fourier basis for speech signals and the wavelet basis for images. If the basis (or *dictionary* when over-complete) is known, one can directly do data compression via greedy algorithms or convex relaxations to obtain the sparse representation [61], or even design the sensing procedure to reduce the samples required for signal recovery [62]. If the dictionary is not known, then one can resort to the so called *dictionary learning* (DL) to try to learn a sparse representation [63], if one exists. The well-known benchmark algorithm for DL is called k-SVD [64], which is a geometry-based algorithm, and can be viewed as a generalization of the clustering algorithms k-means and k-planes. However, as noted in the original paper, k-SVD does not scale well as the size of the dictionary increases. Thus k-SVD is often used to construct a dictionary of small image patches of size 8×8, with a few hundreds of atoms.

DL can also be formulated as a matrix factorization problem

$$\begin{aligned} \underset{\boldsymbol{D},\boldsymbol{S}}{\text{minimize}} \quad & \frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{D}\boldsymbol{S}\|_F^2 + r(\boldsymbol{S}) \\ \text{subject to} \quad & \boldsymbol{D}\in\mathcal{D}, \end{aligned} \tag{15}$$

where $r(\cdot)$ is a sparsity inducing regularization, e.g., the cardinality, the ℓ_1 norm, or the log penalty; conceptually there is no need for a constraint on $\boldsymbol{D}$, however, due to the scaling ambiguity inherent in the matrix factorization problem, we need to impose some norm constraint on the scaling of $\boldsymbol{D}$ to make the problem better defined. For example, we can bound the norm of each atom in the dictionary, $||\boldsymbol{d}_i|| \leq 1, \forall i = 1,\ldots,k$, where $\boldsymbol{d}_i$ is the i-th column of $\boldsymbol{D}$, and we adopt this constraint here.

Although bounding the norm of the columns of $\boldsymbol{D}$ works well, it also complicates the update of $\boldsymbol{D}$—without this constraint, each row of $\boldsymbol{D}$ is the solution of an independent least-squares problem sharing the same mixing matrix, while the constraint couples the columns of $\boldsymbol{D}$, making the problem non-separable. Existing algorithms either solve it approximately [65] or by sub-optimal methods like cyclic column updates [66]. On the other hand, this is not a problem at all for our proposed ADMM sub-routine Algorithm 1: the row separability of the cost function and the column separability of the constraints are handled separately by the two primal variable blocks, while our previously discussed Cholesky caching, warm starting, and good choice of ρ ensure that an exact dictionary update can be done very efficiently.

The update of $\boldsymbol{S}$, sometimes called the sparse coding step, is a relatively well-studied problem for which numerous algorithms have been proposed. We mainly focus on the ℓ_1 regularized formulation, in which case the sub-problem becomes the well-known LASSO, and in fact a large number of LASSOs sharing the same mixing matrix. Algorithm 1 can be used by replacing the proximity step with the soft-thresholding operator. Furthermore, if an over-complete dictionary is trained, the least-squares step can also be accelerated by using the matrix inversion lemma:

$$\left(\boldsymbol{D}^T\boldsymbol{D}+\rho\boldsymbol{I}\right)^{-1} = \rho^{-1}\boldsymbol{I} - \rho^{-1}\boldsymbol{D}^T\left(\rho\boldsymbol{I}+\boldsymbol{D}\boldsymbol{D}^T\right)^{-1}\boldsymbol{D}.$$

Thus, if $m \ll k$, one can cache the Cholesky of $\rho\boldsymbol{I}+\boldsymbol{D}\boldsymbol{D}^T = \boldsymbol{L}\boldsymbol{L}^T$ instead, and replace the least-squares step in Algorithm 1with

$$\widetilde{\boldsymbol{S}} \leftarrow \rho^{-1}\left(\boldsymbol{B} - \boldsymbol{D}^T\left(\boldsymbol{L}^T\right)^{-1}\boldsymbol{L}^{-1}\boldsymbol{D}\boldsymbol{B}\right),$$

where $\boldsymbol{B} = \boldsymbol{D}^T\boldsymbol{Y} + \rho(\boldsymbol{S}+\boldsymbol{U})$. The use of ADMM for LASSO is also discussed in [67]–[69], and [30], and we generally followed the one described in [30, Sec. 7]. Again, one should notice that compared to a plain LASSO, our LASSO sub-problem in the AO framework comes with a good initialization, therefore only a very small number of ADMM-iterations are required for convergence.

It is interesting to observe that for the particular constraints and regularization used in DL, incorporating non-negativity maintains the simplicity of our proposed algorithm—for both the norm bound constraint and ℓ_1 regularization, the proximity operator in Algorithm 1 with non-negativity constraint simply requires zeroing out the negative values before doing the same operations. In some applications non-negativity can greatly help the identification of the dictionary [70].

Fig. 8. Trained dictionary from the MNIST handwritten digits dataset.

As an illustrative example, we trained a dictionary from the MNIST handwritten digits dataset[11], which is a collection of gray-scale images of handwritten digits of size 28×28, and for each digit we randomly sampled 1000 images, forming a matrix of size 784×10000. Non-negativity constraints are imposed on both the dictionary and the sparse coefficients. For $k = 100$, and by setting the ℓ_1 penalty parameter $\lambda = 0.5$, the trained dictionary after 100 AO-ADMM (outer-)iterations is shown in Fig. 8. On average approximately 11 atoms are used to represent each image, and the whole model is able to describe approximately 60% of the energy of the original data, and the entire training time takes about 40 seconds. Most of the atoms in the dictionary remain readable, which shows the good interpretability afforded by the additional non-negativity constraint.

For comparison, we tried the same data set with the same parameter settings with the popular and well-developed DL package SPAMS[12]. For fair comparison, we used SPAMS in batch mode with batch size equal to the size of the training data, and run it for 100 iterations (same number of iterations as AO-ADMM). The quality of the SPAMS dictionary is almost the same as that of AO-ADMM, but it takes SPAMS about 3 minutes to run through these 100 iterations, versus 40 seconds for AO-ADMM. The performance does not change much if we remove the non-negativity constraint when using SPAMS, although the resulting dictionary then loses interpretability. Notice that SPAMS is fully developed in C++, whereas our implementation is simply written in MATLAB, which leaves considerable room for speed improvement using a lower-level language compiler.

VI. Conclusion

In this paper we proposed a novel AO-ADMM algorithmic framework for matrix and tensor factorization under a variety of constraints and loss functions. The main advantages of the proposed AO-ADMM framework are:

- *Efficiency.* By carefully adopting AO as the optimization backbone and ADMM for the individual sub-problems, a

[11] http://www.cs.nyu.edu/~roweis/data.html

[12] http://spams-devel.gforge.inria.fr/index.html

significant part of the required computations can be effectively cached, leading to a per-iteration complexity similar to the workhorse ALS algorithm for unconstrained factorization. Warm-start that is naturally provided by AO together with judicious regularization and choice of parameters further reduce the number of inner ADMM and outer AO iterations.

- *Flexibility.* Thanks to ADMM, which is a special case of the proximal algorithm, non-least-squares terms can be handled efficiently with element-wise complexity using the well-studied proximity operators. This includes almost all non-parametric constraints and regularization penalties commonly imposed on the factors, and even non-least-squares fitting criteria.
- *Convergence.* AO guarantees monotone decrease of the loss function, which is a nice property for the NP-hard factorization problems considered. Moreover, recent advances on generalizations of the traditional BCD algorithms further guarantee convergence to a stationary point.

Case studies on non-negative matrix/tensor factorization, constrained matrix/tensor completion, and dictionary learning, with extensive numerical experiments using real data, corroborate our main claims. We believe that AO-ADMM can serve as a plug-and-play framework that allows easy exploration of different types of constraints and loss functions, as well as different types of matrix and tensor (co-)factorization models.

REFERENCES

[1] K. Huang, N. D. Sidiropoulos, and A. P. Liavas, "Efficient algorithms for 'universally' constrained matrix and tensor factorization," presented at the EUSIPCO, Nice, France, Aug. 31–Sept. 4 2015.

[2] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[3] G. Tomasi and R. Bro, "A comparison of algorithms for fitting the PARAFAC model," *Comput. Statist. Data Anal.*, vol. 50, no. 7, pp. 1700–1734, 2006.

[4] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[5] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by V1?," *Vision Res.*, vol. 37, no. 23, pp. 3311–3325, 1997.

[6] T. Hofmann, "Probabilistic latent semantic indexing," in *Proc. ACM SIGIR Conf.*, 1999, pp. 50–57.

[7] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Found. Comput. Math.*, vol. 9, no. 6, pp. 717–772, 2009.

[8] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?," *J. ACM*, vol. 58, no. 3, p. 11, 2011.

[9] B. Recht, M. Fazel, and P. A. Parrilo, "Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization," *SIAM Rev.*, vol. 52, no. 3, pp. 471–501, 2010.

[10] K. Huang, N. D. Sidiropoulos, and A. Swami, "Non-negative matrix factorization revisited: Uniqueness and algorithm for symmetric decomposition," *IEEE Trans. Signal Process.*, vol. 62, no. 1, pp. 211–224, Jan. 2014.

[11] N. D. Sidiropoulos and R. Bro, "On the uniqueness of multilinear decomposition of N-way arrays," *J. Chemometr.*, vol. 14, no. 3, pp. 229–239, 2000.

[12] A. Smilde, R. Bro, and P. Geladi, *Multi-Way Analysis: Applications in the Chemical Sciences*. New York, NY, USA: USA, 2005.

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

[14] P. Comon, "Tensors: A brief introduction," *IEEE Signal Process. Mag.*, vol. 31, no. 3, pp. 44–53, 2014.

[15] S. A. Vavasis, "On the complexity of nonnegative matrix factorization," *SIAM J. Optim.*, vol. 20, no. 3, pp. 1364–1377, 2009.

[16] C. J. Hillar and L.-H. Lim, "Most tensor problems are NP-hard," *J. ACM*, vol. 60, no. 6, p. 45, 2013.

[17] B. W. Bader and T. G. Kolda, "Efficient MATLAB computations with sparse and factored tensors," *SIAM J. Scientif. Comput.*, vol. 30, no. 1, pp. 205–231, 2007.

[18] U. Kang, E. E. Papalexakis, A. Harpale, and C. Faloutsos, "GigaTensor: Scaling tensor analysis up by 100 times-algorithms and discoveries," in *Proc. ACM SIGKDD*, 2012, pp. 316–324.

[19] A.-H. Phan, P. Tichavsky, and A. Cichocki, "Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations," *IEEE Trans. Signal Process.*, vol. 61, no. 19, pp. 4834–4846, 2013.

[20] N. Ravindran, N. D. Sidiropoulos, S. Smith, and G. Karypis, "Memory-efficient parallel computation of tensor and matrix products for big tensor decomposition," in *Proc. Asilomar Conf. Signals, Syst., Comput.*, 2014, pp. 581–585.

[21] J. H. Choi and S. V. N. Vishwanathan, "DFacTo: Distributed factorization of tensors," in *Adv. Neural Inf. Process. Syst.*, 2014, pp. 1296–1304.

[22] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "SPLATT: Efficient and parallel sparse tensor-matrix multiplication," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2015, pp. 61–70.

[23] D. P. Bertsekas, *Nonlinear Programming*. Belmont, MA, USA: Athena Scientific, 1999.

[24] P. Tseng, "Convergence of a block coordinate descent method for non-differentiable minimization," *J. Optim. Theory Appl.*, vol. 109, no. 3, pp. 475–494, 2001.

[25] L. Grippo and M. Sciandrone, "On the convergence of the block nonlinear Gauss-Seidel method under convex constraints," *Oper. Res. Lett.*, vol. 26, no. 3, pp. 127–136, 2000.

[26] B. Chen, S. He, Z. Li, and S. Zhang, "Maximum block improvement and polynomial optimization," *SIAM J. Optim.*, vol. 22, no. 1, pp. 87–107, 2012.

[27] M. Razaviyayn, M. Hong, and Z.-Q. Luo, "A unified convergence analysis of block successive minimization methods for nonsmooth optimization," *SIAM J. Optim.*, vol. 23, no. 2, pp. 1126–1153, 2013.

[28] N. Li, S. Kindermann, and C. Navasca, "Some convergence results on the regularized alternating least-squares method for tensor decomposition," *Linear Algebra Appl.*, vol. 438, no. 2, pp. 796–812, 2013.

[29] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM J. Imag. Sci.*, vol. 6, no. 3, pp. 1758–1789, 2013.

[30] S. P. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.

[31] E. Ryu and S. P. Boyd, "A primer on monotone operator methods," *Appl. Comput. Math.*, vol. 15, no. 1, pp. 3–43, 2016.

[32] E. Ghadimi, A. Teixeira, I. Shames, and M. Johansson, "Optimal parameter selection for the alternating direction method of multipliers (ADMM): Quadratic problems," *IEEE Trans. Autom. Control*, vol. 60, no. 3, pp. 644–658, Mar. 2015.

[33] N. Parikh and S. P. Boyd, "Proximal algorithms," *Found. Trends Optim.*, vol. 1, no. 3, pp. 123–231, 2014.

[34] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra, "Efficient projections onto the ℓ_1-ball for learning in high dimensions," in *Proc. ACM ICML*, 2008, pp. 272–279.

[35] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore, MD, USA: Johns Hopkins Univ. Press, 1996.

[36] D. L. Sun and C. Fevotte, "Alternating direction method of multipliers for non-negative matrix factorization with the beta-divergence," in *Proc. IEEE ICASSP*, 2014, pp. 6201–6205.

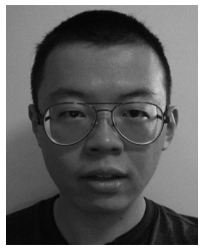
[37] A. Cichocki and A.-H. Phan, "Fast local algorithms for large scale nonnegative matrix and tensor factorizations," *IEICE Trans. Fund. Electron., Commun., Comput. Sci.*, vol. 92, no. 3, pp. 708–721, 2009.

[38] J. B. Kruskal, R. A. Harshman, and M. E. Lundy, "How 3-MFA data can cause degenerate PARAFAC solutions, among other relationships," *Multiway Data Anal.*, pp. 115–122, 1989.

[39] A. Stegeman, "Finding the limit of diverging components in three-way candecomp/parafac-a demonstration of its practical merits," *Comput. Stat. Data Anal.*, vol. 75, pp. 203–216, Jul. 2014.

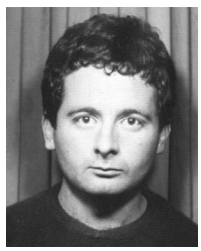
[40] A. P. Liavas and N. D. Sidiropoulos, "Parallel algorithms for constrained tensor factorization via the alternating direction method of multipliers," *IEEE Trans. Signal Process.*, vol. 63, no. 20, pp. 5450–5463, 2015.

[41] D. Chen and R. J. Plemmons, "Nonnegativity constraints in numerical analysis," in *Proc. Symp. Birth Numer. Anal.*, 2009, pp. 109–140.
[42] R. Bro and S. De Jong, "A fast non-negativity-constrained least squares algorithm," *J. Chemometr.*, vol. 11, no. 5, pp. 393–401, 1997.
[43] H. Kim and H. Park, "Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 2, pp. 713–730, 2008.
[44] J. Kim and H. Park, "Fast nonnegative matrix factorization: An active-set-like method and comparisons," *SIAM J. Scientif. Comput.*, vol. 33, no. 6, pp. 3261–3281, 2011.
[45] J. Kim and H. Park, "Fast nonnegative tensor factorization with an active-set-like method," *High-Performance Scientific Computing*. New York, NY, USA: Springer, 2012, pp. 311–326.
[46] M. H. Van Benthem and M. R. Keenan, "Fast algorithm for the solution of large-scale non-negativity-constrained least squares problems," *J. Chemometr.*, vol. 18, no. 10, pp. 441–450, 2004.
[47] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Adv. Neural Inf. Process. Syst. (NIPS)*, 2001, pp. 556–562.
[48] N. Gillis and F. Glineur, "Accelerated multiplicative updates and hierarchical als algorithms for nonnegative matrix factorization," *Neural Comput.*, vol. 24, no. 4, pp. 1085–1105, 2012.
[49] X. Cai, Y. Chen, and D. Han, "Nonnegative tensor factorizations using an alternating direction method," *Frontiers Math. China*, vol. 8, no. 1, pp. 3–18, 2013.
[50] Y. Xu, W. Yin, Z. Wen, and Y. Zhang, "An alternating direction algorithm for matrix completion with nonnegative factors," *Frontiers Math. China*, vol. 7, no. 2, pp. 365–384, 2012.
[51] J. E. Cohen, R. C. Farias, and P. Comon, "Fast decomposition of large nonnegative tensors," *IEEE Signal Process. Lett.*, vol. 22, no. 7, pp. 862–866, 2015.
[52] L. Sorber, M. Van Barel, and L. De Lathauwer, *Tensorlab v2.0*, Jan. 2014 http://www.tensorlab.net/.
[53] L. Sorber, M. Van Barel, and L. De Lathauwer, "Structured data fusion," *IEEE J. Sel. Topics Signal Process.*, vol. 9, no. 4, pp. 586–600, 2015.
[54] N. Vervliet, O. Debals, L. Sorber, and L. De Lathauwer, "Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 71–79, 2014.
[55] B. W. Bader *et al.*, MATLAB tensor toolbox, vers. 2.6, Feb. 2015, [Online]. Available: http://www.sandia.gov/∼tgkolda/TensorToolbox/
[56] S. Gandy, B. Recht, and I. Yamada, "Tensor completion and low-n-rank tensor recovery via convex optimization," *Inverse Problems*, vol. 27, no. 2, p. 025010, 2011.
[57] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 208–220, 2013.
[58] X. Fu, K. Huang, W.-K. Ma, N. D. Sidiropoulos, and R. Bro, "Joint tensor factorization and outlying slab suppression with applications," *IEEE Trans. Signal Process.*, vol. 63, no. 23, pp. 6315–6328, 2015.
[59] C. A. Andersson and R. Bro, "The N-way toolbox for MATLAB," *Chemometr. Intell. Lab. Syst.*, vol. 52, no. 1, pp. 1–4, 2000.
[60] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proc. 10th Int. Conf. World Wide Web*, 2001, pp. 285–295.
[61] A. M. Bruckstein, D. L. Donoho, and M. Elad, "From sparse solutions of systems of equations to sparse modeling of signals and images," *SIAM Rev.*, vol. 51, no. 1, pp. 34–81, 2009.
[62] E. J. Candès and M. B. Wakin, "An introduction to compressive sampling," *IEEE Signal Process. Mag.*, vol. 25, no. 2, pp. 21–30, 2008.
[63] I. Tosic and P. Frossard, "Dictionary learning," *IEEE Signal Process. Mag.*, vol. 28, no. 2, pp. 27–38, 2011.
[64] M. Aharon, M. Elad, and A. Bruckstein, "k-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, 2006.
[65] M. Razaviyayn, H.-W. Tseng, and Z.-Q. Luo, "Dictionary learning for sparse representation: Complexity and algorithms," in *Proc. IEEE ICASSP*, 2014, pp. 5247–5251.
[66] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online learning for matrix factorization and sparse coding," *J. Mach. Learn. Res.*, vol. 11, pp. 19–60, 2010.
[67] M. V. Afonso, J. M. Bioucas-Dias, and M. A. T. Figueiredo, "Fast image recovery using variable splitting and constrained optimization," *IEEE Trans. Image Process.*, vol. 19, no. 9, pp. 2345–2356, 2010.
[68] J. Yang and Y. Zhang, "Alternating direction algorithms for l_1-problems in compressive sensing," *SIAM J. Scientif. Comput.*, vol. 33, no. 1, pp. 250–278, 2011.
[69] E. Esser, Y. Lou, and J. Xin, "A method for finding structured sparse solutions to nonnegative least squares problems with applications," *SIAM J. Imag. Sci.*, vol. 6, no. 4, pp. 2010–2046, 2013.
[70] P. O. Hoyer, "Non-negative matrix factorization with sparseness constraints," *J. Mach. Learn. Res.*, vol. 5, pp. 1457–1469, 2004.



Kejun Huang (S'13) received the B.Eng. in communication engineering from Nanjing University of Information Science and Technology, Nanjing, China, in 2010. He received the Ph.D. degree in electrical engineering from University of Minnesota, Minneapolis, MN, USA, in 2016. His research interests include signal processing, machine learning, and optimization.

Nicholas D. Sidiropoulos (F'09) received the Diploma in electrical engineering from the Aristotelian University of Thessaloniki, Greece, and M.S. and Ph.D. degrees in electrical engineering from the University of Maryland—College Park, in 1988, 1990 and 1992, respectively. He served as assistant professor at the University of Virginia, associate professor at the University of Minnesota, and professor at TU Crete, Greece. Since 2011, he has been at the University of Minnesota, where he currently holds an ADC Chair in digital technology. His research spans topics in signal processing theory and algorithms, optimization, communications, and factor analysis—with a long-term interest in tensor decomposition and its applications. His current focus is primarily on signal and tensor analytics for learning from big data. He received the NSF/CAREER award in 1998, and the IEEE Signal Processing (SP) Society Best Paper Award in 2001, 2007, and 2011. He served as IEEE SP Society Distinguished Lecturer (2008–2009), and as Chair of the IEEE Signal Processing for Communications and Networking Technical Committee (2007–2008). He received the 2010 IEEE SP Society Meritorious Service Award, and the 2013 Distinguished Alumni Award from the Department of ECE, University of Maryland. He is a Fellow of EURASIP (2014).



Athanasios P. Liavas (M'89) received the Diploma and the Ph.D. from the Department of Computer Engineering and Informatics, University of Patras, Greece, in 1989 and 1993, respectively. He served as Assistant Professor at the Department of Mathematics of the University of the Aegean (2001–2004); Associate Professor (2004–2009); and Professor (2009–present) at the School of Electronic and Computer Engineering of the Technical University of Crete. His research interests lie in the area of signal processing and machine learning.

Dr. Liavas is a member of the Technical Chamber of Greece.