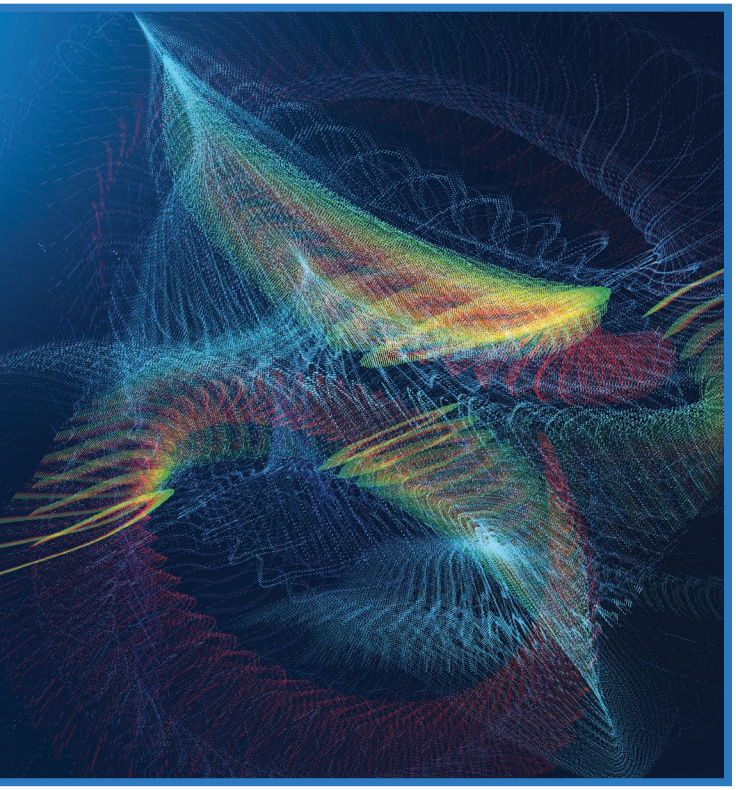
Xiao Fu, Nico Vervliet, Lieven De Lathauwer, Kejun Huang, and Nicolas Gillis

Computing Large-Scale Matrix and Tensor Decomposition With Structured Factors

A unified nonconvex optimization perspective



©ISTOCKPHOTO.COM/IN-FUTURE

Digital Object Identifier 10.1109/MSP.2020.3003544
Date of current version: 2 September 2020

During the past 20 years, low-rank tensor and matrix decomposition models (LRDMs) have become indispensable tools for signal processing, machine learning, and data science. LRDMs represent high-dimensional, multiaspect, and multimodal data using low-dimensional latent factors in a succinct and parsimonious way. LRDMs can serve a variety of purposes, e.g., data embedding (dimensionality reduction), denoising, latent variable analysis, model parameter estimation, and big data compression; see [1]–[5] for surveys of applications.

LRDMs often pose challenging optimization problems. This article aims at introducing recent advances and key computational aspects in structured low-rank matrix and tensor decomposition (SLRD). Here, *structured decomposition* refers to the techniques that impose structural requirements (e.g., nonnegativity, smoothness, and sparsity) onto the latent factors when computing the decomposition (see Figures 1–3 for a number of examples and the references therein). Incorporating structural information is well motivated in many cases. For example, adding constraints/regularization terms typically enhances performance in the presence of noise and modeling errors since constraints and regularization terms impose prior information on the latent factors. For certain tensor decompositions, such as the canonical polyadic decomposition (CPD), adding constraints (including nonnegativity and orthogonality) converts ill-posed optimization problems (where optimal solutions do not exist) into well-posed ones [6]. In addition, constraints and regularization terms can make the results more "interpretable"; e.g., if one aims at estimating probability mass functions (PMFs) or power spectra from data, adding probability simplex or nonnegativity constraints to the latent factors makes the outputs consistent with the design objectives. For matrix decomposition, adding constraints is even more critical (e.g., adding nonnegativity to the latent factors can make highly nonunique matrix decompositions have essentially unique latent factors [1], [4]) as model uniqueness is a core consideration in parameter identification, signal separation, and unsupervised machine learning.

Due to the importance of LRDMs, a plethora of algorithms has been proposed. The overview papers on tensor decomposition [2], [5] have discussed many relevant models and their algebraic properties

as well as popular decomposition algorithms (without emphasizing structured decomposition). In terms of incorporating structural information, nonnegativity- and sparsity-related algorithms have been given the most attention, due to their relevance in image, video, and text data analytics; see, e.g., the tutorial articles published in 2014 in [9] and [1] for LRDMs with nonnegativity constraints.

In this article, instead of offering a comprehensive overview of algorithms under different LRDMs and particular types of constraints, we will provide a unified and principled nonconvex, nonsmooth optimization perspective for SLRD. We will pay particular attention to the following two aspects. First, we will introduce how different nonconvex optimization tools [in particular, block coordinate descent (BCD), Gauss–Newton (GN) algorithms, and stochastic optimization] can be combined with tensor/matrix structures to come up with lightweight algorithms while considering various structural requirements. Second, we will touch upon the key considerations for ensuring that these algorithms have convergence guarantees (e.g., assurances for convergence to a stationary point) since convergence guarantees are important for designing stable and disciplined algorithms. Both nonconvex, nonsmooth optimization and tensor/matrix decomposition are nontrivial. We hope that this article will provide the readers (especially graduate students) with an entry point for understanding the key ingredients that are needed for designing structured decomposition algorithms in a disciplined way.

Notation

We follow the established conventions in signal processing and use $\mathcal{T}$, $\boldsymbol{X}$, and $\boldsymbol{x}$ to denote a tensor, a matrix, and a vector, respectively. The notations $\otimes, \odot, \circledast$, and $\circ$ denote the Kronecker product, Khatri–Rao product, Hadamard product, and outer product, respectively. The MATLAB notation $\boldsymbol{X}(m,:)$ is used to denote the mth row of $\boldsymbol{X}$, and other MATLAB notations, such as $\boldsymbol{X}(:,n)$ and $\boldsymbol{X}(i,j)$, are also used. In some cases, $[\boldsymbol{X}]_{i,j}$ and $[\boldsymbol{x}]_j$ denote the (i,j)th entry of $\boldsymbol{X}$ and the jth element of $\boldsymbol{x}$, respectively. The notation $\boldsymbol{X} = [\boldsymbol{X}_1;\ldots;\boldsymbol{X}_N] = [\boldsymbol{X}_1^\top,\ldots,\boldsymbol{X}_N^\top]^\top$ denotes the concatenation of the matrices $\{\boldsymbol{X}_n\}_{n=1}^N$.

Problem statement

Low-rank matrix and tensor decomposition models

Under a noiseless setting, matrix decomposition aims at finding the following representation of a data matrix $\boldsymbol{X}$:

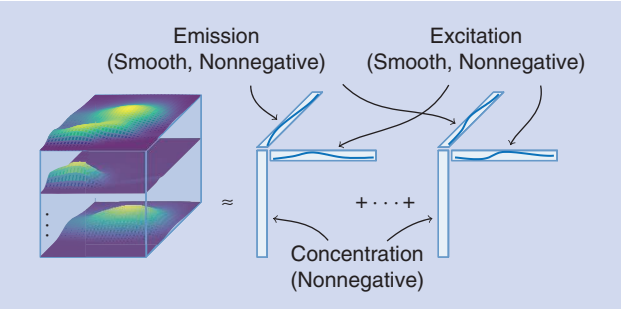


FIGURE 1. An SLRD model in fluorescence data analytics. The rank-1 components correspond to different analytes that constitute the data samples. The latent factors have a physical meaning, and using prior structural information improves the decomposition performance.

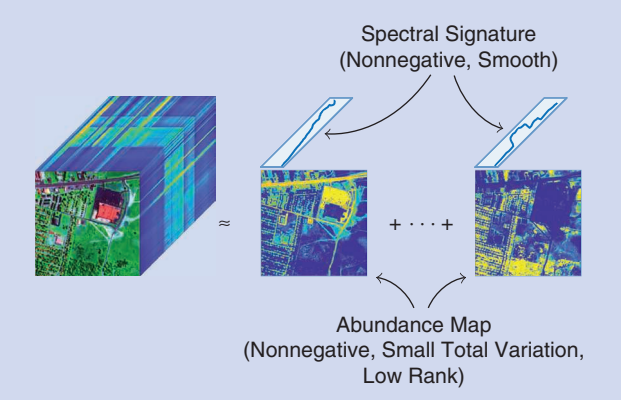


FIGURE 2. The linear mixture model for hyperspectral unmixing (HU). The HU problem can be considered either as an NMF problem [1], [4] or a block-term tensor decomposition problem [7]. Both are SLRDs.

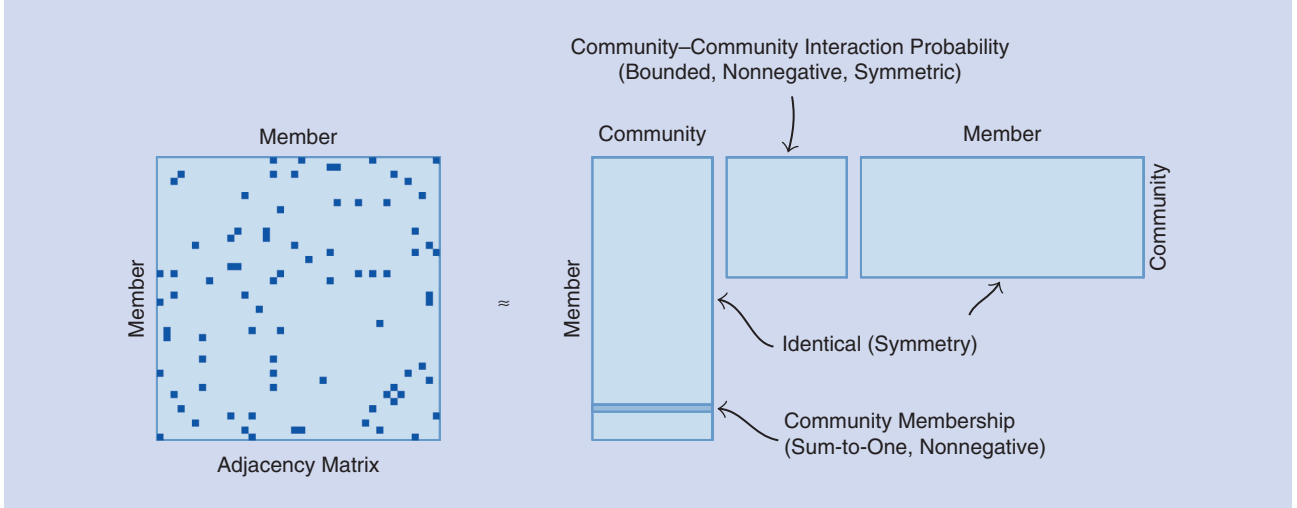


FIGURE 3. An SLRD perspective for community detection under the mixed-membership stochastic block model [8]. The binary adjacency matrix can be considered a noisy, structured, low-rank model. Again, a series of model priors can be used as structural constraints on the latent factor matrices on the right-hand side.

$$\boldsymbol{X} = \boldsymbol{A}_1\boldsymbol{A}_2^{\top} = \sum_{r=1}^{R} \boldsymbol{A}_1(:,r)\circ\boldsymbol{A}_2(:,r), \tag{1}$$

where $\boldsymbol{X} \in \mathbb{R}^{I_1\times I_2}$, $\boldsymbol{A}_1 \in \mathbb{R}^{I_1\times R}$, and $\boldsymbol{A}_2 \in \mathbb{R}^{I_2\times R}$. The integer $R \le \min\{I_1, I_2\}$ is the smallest integer such that the preceding equality holds; i.e., R denotes the matrix rank. If the data entries have more than two indices, the data array is called a *tensor*. Unlike matrices whose rank decomposition is defined as in (1), there are a variety of tensor decomposition models involving different high-order generalizations of the matrix rank. One of the most popular models is CPD [10]. For an Nth-order tensor $\mathcal{T} \in \mathbb{R}^{I_1\times\ldots\times I_N}$, its CPD representation is as follows:

$$\mathcal{T} = \sum_{r=1}^{R} \boldsymbol{A}_1(:,r)\circ\ldots\circ\boldsymbol{A}_N(:,r) \coloneqq [\![\boldsymbol{A}_1,\ldots,\boldsymbol{A}_N]\!], \tag{2}$$

where R is again the smallest integer such that the equality holds (i.e., R is the CP rank of $\mathcal{T}$) and $\boldsymbol{A}_n \in \mathbb{R}^{I_n\times R}$ denotes the mode-n latent factor (see the visualization of a third-order case in Figure 1). Besides CPD, there is, for instance, the Tucker decomposition model, i.e., $\mathcal{T} = \mathcal{G}\times_1\boldsymbol{A}_1\times_2\ldots\times_N\boldsymbol{A}_N$, where $\mathcal{G}$ denotes the so-called core tensor and $\times_n$ is the mode-n product. More recently, a series of extensions and hybrid models has also emerged, including the block-term decomposition (BTD), multilinear rank-$(L_r, L_r, 1)$ decomposition (LL1), coupled CPD/BTD models, the tensor train model, and the hierarchical Tucker model; see [2] and the references therein. In this article, we will mainly focus on the models in (1) and (2) and use them to illustrate different algorithm design principles. Generalization to other models will also be briefly discussed at the end.

In their general formulation, most SLRD problems are NP-hard [11]–[13]. Apart from that, the era of big data brings its own challenges. For example, a $2{,}000\times 2{,}000\times 2{,}000$ tensor (i.e., $I_n = I = 2{,}000$ for all n) requires 58 GB of memory (if the double precision is used). When there is no constraint or regularization on the latent factors, using first-order optimization techniques (e.g., gradient descent or BCD) under the "optimization-friendly" Euclidean loss already costs $O(RI^3)$ floating-point operations (flops) per iteration for the rank-R CPD of this tensor. With constraints and regularization terms, the complexity might be even higher. The situation gets worse when one deals with higher-order tensors. Hence, designing effective algorithms requires synergies between sophisticated optimization tools and the algebraic structures embedded in LRDMs.

Structured decomposition as nonconvex optimization

SLRD can be viewed from a model-fitting perspective. That is, we hope to find a tensor/matrix model that best approximates the data tensor or matrix under a certain distance measure, with prior information about the model parameters. This point of view makes a lot of sense. In practice, the data matrix/tensor often consists of low-rank "essential information" and high-rank noise; thus, using a model-fitting formulation instead of seeking an exact decomposition, such as those in (1) and (2), is more meaningful. Conceptually, the SLRD problems can be summarized as follows:

$$\min_{\text{model param.}} \ \text{dist}(\text{data},\text{model}) + \begin{pmatrix}\text{penalty for}\\ \text{structure violation}\end{pmatrix}$$
$$\text{under} \quad \text{structural constraints}, \tag{3}$$

where dist $(\boldsymbol{X},\boldsymbol{Y})$ is a "distance measure" between $\boldsymbol{X}$ and $\boldsymbol{Y}$ in a certain sense. The most commonly used measure is the (squared) Euclidean distance, i.e.,

$$\text{dist}\,(\boldsymbol{X},\boldsymbol{Y}) = \|\boldsymbol{X}-\boldsymbol{Y}\|_{\text{F}}^2.$$

In addition, a number of other measures are of interest in data science. For example, the Kullback–Leibler (KL) divergence is often used for measuring the "distance" between distributions of random variables (RVs), and it is also commonly used in integer data-fitting problems [since it is closely related to the maximum likelihood estimators (MLEs) that are associated with discrete RVs, e.g., those following the Poisson or Bernoulli distributions]. The ℓ_1 norm and the Huber function as well as their nonconvex counterparts (e.g., the ℓ_p function where $0 < p < 1$ [14]) are used for outlier-robust data analytics; see the "More Discussions and Conclusion" section. The "structural constraints" and "structure violation penalty" are imposed on the model parameters [e.g., the $\boldsymbol{A}_n$s in (2)]. For example, consider CPD under sparsity and nonnegativity conditions, which finds applications in many data analytics problems [15]:

$$\min_{\{\boldsymbol{A}_n\}_{n=1}^N} \ \frac{1}{2}\|\mathcal{T}-[\![\boldsymbol{A}_1,\ldots,\boldsymbol{A}_N]\!]\|_{\text{F}}^2 + \lambda\sum_{n=1}^{N}\|\boldsymbol{A}_n\|_1$$
$$\text{s.t. } \boldsymbol{A}_n \ge \boldsymbol{0}, \forall n. \tag{4}$$

From an optimization viewpoint, these SLRD problems can be summarized in a succinct form:

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) + h(\boldsymbol{\theta}), \tag{5}$$

where $\boldsymbol{\theta}$ collects all the latent parameters of the tensor/matrix model of interest, $f(\boldsymbol{\theta})$ represents the data-fitting part, and $h(\boldsymbol{\theta})$ represents regularization terms added on the latent factor. Note that the expression in (5) also includes the case where $\boldsymbol{\theta}$ is subject to hard constraints; i.e., $\boldsymbol{\theta} \in C$ can also be expressed as a penalty term, where $h(\boldsymbol{\theta})$ is the indicator function of the set C. For example, in (4), $\boldsymbol{\theta} = [\boldsymbol{\theta}_1;\ldots;\boldsymbol{\theta}_N]$ where $\boldsymbol{\theta}_n = \text{vec}(\boldsymbol{A}_n)$. Here, $h(\boldsymbol{\theta}) = \Sigma_{n=1}^N h_n(\boldsymbol{\theta}_n)$, and $h_n(\boldsymbol{\theta}_n) = h_n^{(1)}(\boldsymbol{\theta}_n) + h_n^{(2)}(\boldsymbol{\theta}_n)$, in which $h_n^{(1)}$ is the indicator function of the nonnegativity orthant and $h_n^{(2)}(\boldsymbol{\theta}_n)$ is the ℓ_1 regularization.

Several observations can be made from (4) and (5). First, the SLRD problems are usually nonconvex since the model approximation part $f(\boldsymbol{\theta})$ is nonconvex; in some cases, $h(\boldsymbol{\theta})$ is also nonconvex; see, e.g., volume minimization-based nonnegative matrix factorization (NMF) [4]. Second, the objective function in (5) is often nonsmooth, especially when a nondifferentiable regularization term $h(\boldsymbol{\theta})$ is involved (e.g., an indicator function for enforcing "hard constraints" or an ℓ_1-norm regularization term). Like many nonconvex, nonsmooth optimization problems, the SLRD problems are NP-hard in most cases [11], [13]. The analytical tool

for characterizing their global optimality-attaining properties of general nonconvex optimization algorithms has been elusive. The convention from the optimization literature is to characterize the algorithms' stationary-point-approaching properties since $\boldsymbol{\theta}$ must be a stationary point of (5) for $\boldsymbol{\theta}$ to be an optimal solution. Simply speaking, assume that the data-fitting part $f(\boldsymbol{\theta})$ is differentiable, and $\text{dom}\,(f+h) = \mathbb{R}^d$, where d is the number of variables. Denote $F(\boldsymbol{\theta}) = f(\boldsymbol{\theta}) + h(\boldsymbol{\theta})$. Then, any stationary point of (5) satisfies the following:

$$\mathbf{0} \in \partial F(\boldsymbol{\theta}) = \nabla f(\boldsymbol{\theta}) + \partial h(\boldsymbol{\theta}), \tag{6}$$

where $\partial h(\boldsymbol{\theta})$ denotes the limiting Fréchet subdifferential of $h(\cdot)$, which is the subgradient when $h(\cdot)$ is convex [16]–[18].

BCD-based approaches

One of the workhorses for LRDMs is BCD. The rationale behind BCD-based structured factorization is straightforward: the factorization problems with respect to a single-block variable $\boldsymbol{A}_n$ in (4) are convex under various models and different $F(\cdot)$'s. BCD alternatingly updates the parameters $\boldsymbol{\theta}_n$ (the nth block of $\boldsymbol{\theta}$) while fixing the others; $\boldsymbol{\theta}_n^{(t+1)}$ is updated using

$$\underset{\boldsymbol{\theta}_n}{\text{argmin}} f(\boldsymbol{\theta}_1^{(t+1)}, \ldots, \boldsymbol{\theta}_{n-1}^{(t+1)}, \boldsymbol{\theta}_n, \boldsymbol{\theta}_{n+1}^{(t)}, \ldots \boldsymbol{\theta}_N^{(t)}) + h_n(\boldsymbol{\theta}_n), \tag{7}$$

where $h_n(\cdot)$ is the part of $h(\cdot)$ that is imposed onto $\boldsymbol{\theta}_n$ and $\boldsymbol{\theta}^{(t)}$ denotes the optimization variables in iteration t. In the sequel, we will use the shorthand notation $f(\boldsymbol{\theta}_n; \boldsymbol{\theta}_{-n}^{(t)}) = f(\boldsymbol{\theta}_1^{(t+1)}, \ldots, \boldsymbol{\theta}_{n-1}^{(t+1)}, \boldsymbol{\theta}_n, \boldsymbol{\theta}_{n-1}^{(t)}, \ldots \boldsymbol{\theta}_N^{(t)})$.

BCD and LRDMs are linked together through the "matrix-unfolding" operation. Unfolding is a way of rearranging the elements of a tensor to a matrix. The mode-n unfolding (matricization) of $\mathcal{T}$ is as follows [2] for all $i_1, \ldots, i_N$,

$$\boldsymbol{X}_n(j, i_n) = \mathcal{T}(i_1, \ldots, i_N),$$

where $j = 1 + \Sigma_{\ell=1, \ell \neq n}(i_\ell - 1) J_\ell$, $J_\ell = \prod_{m=1, m \neq n}^{\ell-1} I_m$. Note that tensor unfolding admits several forms in the literature. For example, the unfolding expressions in the two tutorial papers [2] and [5] are different. In this article, we follow the convention in [2]. For tensors with CP rank R, the unfolding has the following compact and elegant expression:

$$\boldsymbol{X}_n = \boldsymbol{H}_n \boldsymbol{A}_n^\top, \tag{8}$$

where the matrix $\boldsymbol{H}_n \in \mathbb{R}^{(\prod_{\ell=1, \ell \neq n}^{N} I_n) \times R}$ is defined as

$$\boldsymbol{H}_n = \boldsymbol{A}_N \odot \ldots \odot \boldsymbol{A}_{n+1} \odot \boldsymbol{A}_{n-1} \odot \ldots \odot \boldsymbol{A}_1.$$

Readers are referred to [2] and [5] for details of unfolding. The unfolding operation explicitly "pushes" the latent factors to the rightmost position in the unfolded tensor representation, which helps efficient algorithm design. Note that many tensor factorization models, e.g., Tucker, BTD, and LL1, have similar multilinearity properties in their respective unfolded representations [19]. Representing the tensor using matrix unfolding, the BCD algorithm for structured CPD consists of the following updates in a cyclical manner:

$$\boldsymbol{A}_n \leftarrow \underset{\boldsymbol{A}_n}{\text{argmin}} \ \frac{1}{2} \left\| \boldsymbol{X}_n - \boldsymbol{H}_n^{(t)} \boldsymbol{A}^\top \right\|_{\text{F}}^2 + h_n(\boldsymbol{A}_n), \tag{9}$$

where $\boldsymbol{H}_n^{(t)} = \boldsymbol{A}_N^{(t)} \odot \ldots \odot \boldsymbol{A}_{n+1}^{(t)} \odot \boldsymbol{A}_{n-1}^{(t+1)} \odot \ldots \odot \boldsymbol{A}_1^{(t+1)}$ since $\boldsymbol{A}_\ell$ for $\ell < n$ has been updated.

Classic BCD-based structured decomposition

If $h_n(\boldsymbol{A}_n)$ is absent, (9) admits an analytical solution; i.e., $\boldsymbol{A}_n^{(t+1)} \leftarrow (\boldsymbol{H}_n^{(t)})^\dagger \boldsymbol{X}_n$, which recovers the classic alternating least-squares (ALS) algorithm for the unconstrained LS loss-based CPD [10]. In principle, if $h_n(\boldsymbol{A}_n)$ is convex, then any off-the-shelf convex optimization algorithms can be utilized to solve (9). However, in the context of SLRD, the employed algorithms should strike a good balance between complexity and efficiency. The reason is that (9) can have a very large size because the row size of $\boldsymbol{H}_n^{(t)}$ is $\prod_{\ell=1, \ell \neq n}^{N} I_\ell$, which can reach into the millions, even when I_n is small.

First-order optimization algorithms (i.e., optimization algorithms using only the gradient information) are known to be scalable, and thus they are good candidates for handling (9). Proximal/projected gradient descent (PGD) [20], [21] is perhaps the easiest to implement. PGD solves (9) using the following iterations:

$$\boldsymbol{A}_n^{(k+1)} \leftarrow \text{Prox}_{h_n}(\boldsymbol{A}_n^{(k)} - \alpha \nabla_{\boldsymbol{A}_n} f(\boldsymbol{A}_n^{(k)}; \boldsymbol{A}_{-n}^{(t)})),$$

where k indexes the iterations of the PGD algorithm. The notation $\text{Prox}_h(\boldsymbol{Z})$ is defined as

$$\text{Prox}_h(\boldsymbol{Z}) = \underset{\boldsymbol{Y}}{\text{argmin}} \ h(\boldsymbol{Y}) + \frac{1}{2} \left\| \boldsymbol{Y} - \boldsymbol{Z} \right\|_{\text{F}}^2,$$

and $\nabla_{\boldsymbol{A}_n} f(\boldsymbol{A}^{(k)}; \boldsymbol{A}_{-n}^{(t)}) = \boldsymbol{A}^{(k)} (\boldsymbol{H}_n^{(t)})^\top \boldsymbol{H}_n^{(t)} - \boldsymbol{X}_n^\top \boldsymbol{H}_n^{(t)}$. For a variety of $h(\cdot)$'s, the proximal operator is easy to compute. For example, if $h(\boldsymbol{Z}) = \lambda \| \boldsymbol{Z} \|_1$, we have $[\text{Prox}_{\lambda \| \cdot \|_1}(\boldsymbol{Z})]_{ij} = \text{sign}(Z_{ij})(|Z_{ij} - \lambda|)_+$, and if $h(\boldsymbol{Z})$ is the indicator function of a closed set $\mathbb{H}$, then the proximal operator becomes a projection operator; i.e., $\text{Prox}_h(\boldsymbol{Z}) = \text{Proj}_{\mathbb{H}}(\boldsymbol{Z}) = \text{argmin}_{\boldsymbol{Y} \in \mathbb{H}} \| \boldsymbol{Y} - \boldsymbol{Z} \|_{\text{F}}^2$. A number of $h(\cdot)$'s that admit simple proximal operations (e.g., the ℓ_1 norm) can be found in [20].

PGD is easy to implement when the proximal operator is simple. When the regularization is complicated, then using algorithms such as the alternating directional method of multipliers (ADMM) to replace PGD may be more effective; see [22] for a collection of examples of ADMM-based constrained LS solving. Beyond PGD and ADMM, many other algorithms have been employed for handling the subproblem in (9) for different structured decomposition problems. For example, accelerated PGD, active set methods, and mirror descent have all been considered in the literature; see, e.g., [23] and [24].

Inexact BCD

Using off-the-shelf solvers to handle the subproblems under the framework of BCD is natural for many LRDMs. However, when the block variable $\boldsymbol{\theta}_{-n}$ is only roughly estimated, it is not necessarily efficient to exactly solve the subproblem with reference to $\boldsymbol{\theta}_n$; after all, $\boldsymbol{\theta}_{-n}$ will change in the next iteration. This argument

leads to a class of algorithms that solves the block subproblems in an inexact manner [16], [18]. Instead of directly minimizing $f(\boldsymbol{A}_n;\boldsymbol{A}_{-n}^{(t)}) + h_n(\boldsymbol{A}_n)$, inexact BCD updates $\boldsymbol{A}_n$ via minimizing a local approximation of $f(\boldsymbol{A}_n;\boldsymbol{A}_{-n}^{(t)}) + h_n(\boldsymbol{A}_n)$ at $\boldsymbol{A}_n = \boldsymbol{A}_n^{(t)}$, which we denote as

$$g(\boldsymbol{A}_n;\mathcal{A}^{(t)}) \approx f(\boldsymbol{A}_n;\boldsymbol{A}_{-n}^{(t)}) + h_n(\boldsymbol{A}_n),$$

where $\mathcal{A}^{(t)} = \{\boldsymbol{A}_1^{(t+1)},\ldots,\boldsymbol{A}_{n-1}^{(t+1)}, \boldsymbol{A}_n^{(t)},\ldots, \boldsymbol{A}_N^{(t)}\}$. That is, inexact BCD updates $\boldsymbol{A}_n$ using

$$\boldsymbol{A}_n^{(t+1)} \leftarrow \operatorname*{argmin}_{\boldsymbol{A}_n} g(\boldsymbol{A}_n;\mathcal{A}^{(t)}).$$

If $g(\boldsymbol{A}_n;\mathcal{A}^{(t)})$ admits a simple minimizer, then the algorithm can quickly update $\boldsymbol{A}_n$ and move to the next block. One of the frequently used $g(\boldsymbol{A}_n;\mathcal{A}^{(t)})$ is as follows:

$$g(\boldsymbol{A}_n;\mathcal{A}^{(t)}) = f(\boldsymbol{A}_n^{(t)};\boldsymbol{A}_{-n}^{(t)}) + h_n(\boldsymbol{A}_n) + \nabla_{\boldsymbol{A}_n} f(\boldsymbol{A}_n^{(t)};\boldsymbol{A}_{-n}^{(t)})^\top (\boldsymbol{A}_n - \boldsymbol{A}_n^{(t)}) + \frac{1}{2\alpha}\left\|\boldsymbol{A}_n - \boldsymbol{A}_n^{(t)}\right\|_{\mathrm{F}}^2, \tag{10}$$

which is obtained via applying Taylor's expansion on the smooth term $f(\boldsymbol{\theta})$. Using this local approximation, the update admits the following form:

$$\boldsymbol{A}_n^{(t+1)} \leftarrow \mathrm{Prox}_{h_n}(\boldsymbol{A}_n^{(t)} - \alpha\nabla_{\boldsymbol{A}_n} f(\boldsymbol{A}_n^{(t)};\boldsymbol{A}_{-n}^{(t)})), \tag{11}$$

which is equivalent to running PGD for one iteration to solve (9)—and this echoes the term *inexact*. A number of frequently used local approximations can be seen in [18]. Note that inexact BCD is not unfamiliar to the SLRD community, especially for nonnegativity constraints. One of the most important early algorithms for NMF, namely, multiplicative updates (MUs) [25], is an inexact BCD algorithm.

Pragmatic acceleration

Compared to exact BCD, inexact BCD normally needs to update all block variables for many more rounds before reaching a "good" solution. Nonetheless, when inexact BCD is combined with the so-called extrapolation technique, the convergence speed can be substantially improved. The procedure of extrapolation is as follows: Consider an extrapolated point

$$\widehat{\boldsymbol{A}}_n^{(t)} = (1+\omega_n^t)\boldsymbol{A}_n^{(t)} + \omega_n^t \boldsymbol{A}_n^{(t-1)}, \tag{12}$$

where $\{\omega_n^t\}$ is a predefined sequence (see practical choices of ω_n^t in [16]). Then, the extrapolation-based inexact BCD replaces (11) by the following:

$$\boldsymbol{A}_n^{(t+1)} \leftarrow \mathrm{Prox}_{h_n}\left(\widehat{\boldsymbol{A}}_n^{(t)} - \alpha\nabla_{\boldsymbol{A}_n} f(\widehat{\boldsymbol{A}}_n^{(t)};\boldsymbol{A}_{-n}^{(t)})\right).$$

In practice, this simple technique often makes a big difference in terms of the convergence speed; see Figure 4. The extrapolation technique was introduced by Nesterov in 1983 to accelerate smooth, single-block convex optimization problems using only first-order derivative information [26]. It was extended to handling nonconvex, multiblock, and nonsmooth problems in the context of tensor decomposition by Xu et al. in 2013 [16]. In this case, no provable acceleration has been shown, which leaves a challenging and interesting research question open. We refer the readers to [73] and the references therein for more information on this topic.

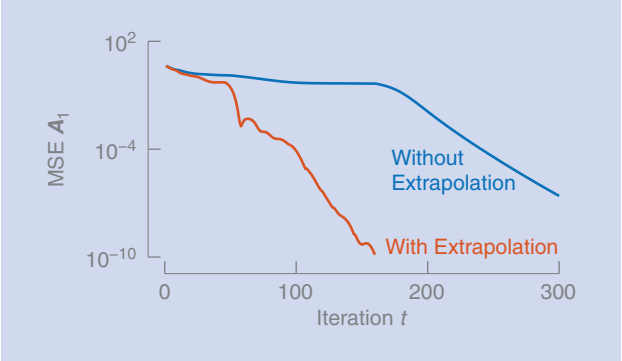


FIGURE 4. The speedup for inexact BCD via extrapolation. The performance is measured by the mean square error (MSE) on the estimated latent factor; see [27]. The tensor size is $30\times30\times30\times30$, and $R = 10$. The inexact BCD and extrapolated version use the updates in (11) and (12) [16], respectively.

Convergence properties and computational complexity

Convergence properties of both exact BCD and inexact BCD are well studied in the literature [18], [28]. An early result from Bertsekas [28] shows that every limit point of $\{\boldsymbol{\theta}^{(t)}\}_t$ is a stationary point of (5) if h is either absent or the indicator function of a convex closed set, and if the subproblems in (7) can be exactly solved with unique minimizers while the objective function is nonincreasing during the interval between two consecutive iterates. This can be achieved if the subproblems in (7) are strictly (quasi)convex. Nonetheless, since $\boldsymbol{H}_n^{(t)}$ may be rank deficient, stationary-point convergence under this framework is not necessarily easy to ensure. In addition, this early result does not cover nonsmooth functions.

For inexact BCD, it was shown in [18] that if $h_n(\boldsymbol{A}_n)$ is convex, and if the local surrogate is strictly (quasi)convex and is "tangent" to $F(\boldsymbol{A}_n;\boldsymbol{A}_{-n}^{(t)})$ at $\boldsymbol{A}_n = \boldsymbol{A}_n^{(t)}$ (i.e., $g(\boldsymbol{A}_n;\mathcal{A}^{(t)})$ is a tight approximation for $F(\boldsymbol{A}_n;\boldsymbol{A}_{-n}^{(t)})$ at $\boldsymbol{A}_n = \boldsymbol{A}_n^{(t)}$, and they share the same directional derivative at this point), then every limit point of the produced solution sequence is a stationary point. This is a somewhat more relaxed condition relative to those for BCD since the upper bound $g(\boldsymbol{A}_n;\mathcal{A}^{(t)})$ can always be constructed as a strictly convex function, e.g., by using (10).

In terms of per-iteration complexity, BCD combined with first-order subproblem solvers for structured tensor decomposition is not a lot more expensive than solving unconstrained ones in many cases, which is the upshot of using algorithms such as PGD, accelerated PGD, and ADMM. The most expensive operation is the so-called matricized tensor times Khatri–Rao product (MTTKRP), i.e., $\boldsymbol{X}_n^\top \boldsymbol{H}_n^{(t)}$. However, even if one uses exact BCD with multiple iterations of PGD and ADMM for solving (9), the MTTKRP needs only to be computed once for every subproblem

with respect to $\boldsymbol{A}_n$, which is the same as in the unconstrained case; see more discussions in [22] and [73].

Block splitting and ordering within BCD

In this article, we focus on the most natural choice of blocks to perform BCD on LRDMs, namely, $\{\boldsymbol{A}_n\}_{n=1}^{N}$. However, in some cases, it might be preferable to optimize across smaller blocks because the subproblems are simpler. For example, with nonnegativity constraints, it has been shown that optimizing across the blocks made of the columns of the $\boldsymbol{A}_n$s is rather efficient (because there is a closed-form solution) and outperforms exact BCD and the MU [1], [9] that are based on $\boldsymbol{A}_n$-block splitting. Another way to modify BCD and possibly improve convergence rates is to update the blocks of variables in a noncyclic way, for example, using random shuffling at each outer iteration or picking the block of variables to update using some criterion that increases our chances to converge faster (e.g., pick the block that was modified the most in the previous iteration, i.e., pick $\operatorname{argmin}_n \| \boldsymbol{A}_n^{(t)} - \boldsymbol{A}_n^{(t-1)} \| / \| \boldsymbol{A}_n^{(t-1)} \|_{\mathrm{F}}$); see, e.g., [29] and [30].

Second-order approaches

Combining SLRD and optimization techniques that exploit (approximate) second-order information has a number of advantages. Empirically, these algorithms converge in much fewer iterations relative to first-order methods, they are less susceptible to the so-called swamps, and they are often more robust to initializations [31], [32]; see, e.g., Figure 5. There are many second-order optimization algorithms, e.g., Newton's method that uses the Hessian and a series of "quasi-Newton" methods that approximate the Hessian. Among these algorithms, the GN framework specialized for handling nonlinear LS (NLS) problems fits Euclidean distance-based tensor/matrix decompositions particularly well. Under the GN framework, the structure inherent to some tensor models (e.g., CPD and LL1) can be exploited to make the per-iteration complexity of the same order as the first-order methods [31].

GN preliminaries

Consider the unconstrained tensor decomposition problem

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \ \text{ with } \ f(\boldsymbol{\theta}) = \frac{1}{2} \| \underbrace{[\![\boldsymbol{A}_1, \ldots, \boldsymbol{A}_N]\!] - \mathcal{T}}_{\mathcal{F}(\boldsymbol{\theta})} \|_{\mathrm{F}}^2. \tag{13}$$

The GN method starts from a linearization of the residual $\mathcal{F}$,

$$\operatorname{vec}(\mathcal{F}(\boldsymbol{\theta})) \approx \operatorname{vec}(\mathcal{F}(\boldsymbol{\theta}^{(t)})) + \left. \frac{\mathrm{d} \operatorname{vec}(\mathcal{F})}{\mathrm{d} \operatorname{vec}(\boldsymbol{\theta})} \right|_{\boldsymbol{\theta}^{(t)}} \cdot \boldsymbol{p} \tag{14}$$

$$= \boldsymbol{f}^{(t)} + \boldsymbol{J}^{(t)} \boldsymbol{p}, \tag{15}$$

where $\boldsymbol{J}^{(t)}$ is the Jacobian of $\mathcal{F}$ with respect to the variables $\boldsymbol{\theta}$, $\boldsymbol{p} = \boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}$, and $\boldsymbol{f}^{(t)} = \operatorname{vec}(\mathcal{F}(\boldsymbol{\theta}^{(t)}))$. Substituting (15) in (13) results in a quadratic optimization problem

$$\boldsymbol{p}^{(t)} \leftarrow \operatorname*{argmin}_{\boldsymbol{p}} \frac{1}{2} \left\| \boldsymbol{f}^{(t)} \right\|^2 + \boldsymbol{g}^{(t)\top} \boldsymbol{p} + \frac{1}{2} \boldsymbol{p}^{\top} \boldsymbol{\Phi}^{(t)} \boldsymbol{p}, \tag{16}$$

in which the gradient is given by $\boldsymbol{g}^{(t)} = \boldsymbol{J}^{(t)\top} \boldsymbol{f}^{(t)}$ and the Gramian (of the Jacobian) is given by $\boldsymbol{\Phi}^{(t)} = \boldsymbol{J}^{(t)\top} \boldsymbol{J}^{(t)}$. The variables are updated as $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} + \boldsymbol{p}^{(t)}$. We have

$$\boldsymbol{\Phi}^{(t)} \boldsymbol{p}^{(t)} = -\boldsymbol{g}^{(t)} \tag{17}$$

by the optimality condition of the quadratic problem in (16). In the case of CPD, the Gramian $\boldsymbol{J}^{(t)\top} \boldsymbol{J}^{(t)}$ is a positive semidefinite matrix instead of a positive definite one (the Gramian $\boldsymbol{J}^{(t)\top} \boldsymbol{J}^{(t)}$ has at least $(N-1)R$ zero eigenvalues because of the scaling indeterminacy), which means that $\boldsymbol{p}^{(t)}$ is not an ascent direction, although it may not be a descent direction either. This problem can be avoided by the Levenberg–Marquardt method, i.e., using $\boldsymbol{\Phi}^{(t)} = \boldsymbol{J}^{(t)\top} \boldsymbol{J}^{(t)} + \lambda \boldsymbol{I}$ for some $\lambda \geq 0$ or using a trust region (TR) that implicitly dampens the system. The GN method can exhibit up to a quadratic convergence rate near an optimum if the residual is small [28], [31].

Second-order methods converge fast once $\boldsymbol{\theta}^{(t)}$ is near a stationary point, while there is a risk that $\boldsymbol{\theta}^{(t)}$ may never come close to any stationary point. To ensure global convergence, i.e., that $\boldsymbol{\theta}^{(t)}$ converges to a stationary point from any starting point $\boldsymbol{\theta}^{(0)}$, globalization strategies can be used [28]. Globalization is considered crucial for nonconvex optimization-based tensor decomposition algorithms and makes them robust with respect to the initial guess $\boldsymbol{\theta}^{(0)}$, as illustrated in Figure 6.

The first effective globalization strategy is determining $\alpha^{(t)}$ via solving the following:

$$\alpha^{(t)} = \operatorname*{argmin}_{\alpha} f(\boldsymbol{\theta}^{(t)} + \alpha \boldsymbol{p}^{(t)}), \tag{18}$$

which is often referred to as the *exact line search* in the optimization literature. Solving this equation can be costly, in general, but when the objective is to minimize a multilinear error term in a LS sense, as in (13), the global minimum of this problem can be found exactly, as the optimality conditions boil down to a polynomial root-finding problem; see [33] and the references therein. This exact line search technique ensures that maximal progress is made in every step of GN, which helps to quickly improve the objective function. Similarly, the exact plane search

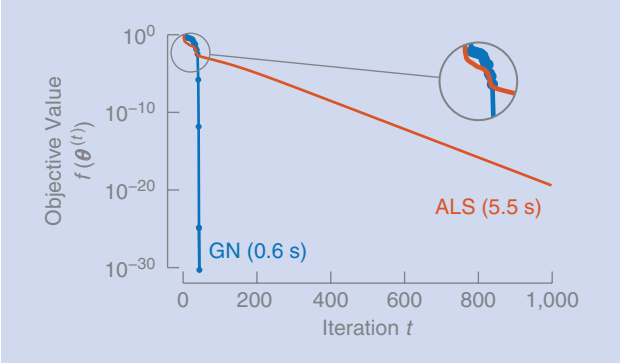


FIGURE 5. While ALS initially improves the function value faster, the GN method converges more quickly, both in terms of time and the number of iterations. Results are shown for a $100 \times 100 \times 100$, rank $R = 10$ tensor with highly correlated rank-1 terms (the average angle is 59°), starting from a random initialization.

can be used to find the best descent direction in the plane spanned by $\boldsymbol{p}^{(t)}$ and $\boldsymbol{g}^{(t)}$ by searching for coefficients α and β that minimize $f(\boldsymbol{\theta}^{(t)} + \alpha\boldsymbol{p}^{(t)} + \beta\boldsymbol{g}^{(t)})$ [33]. Empirically, the steepest descent direction $-\boldsymbol{g}^{(t)}$ decreases the objective function more rapidly during earlier iterations, while the GN step enables fast convergence. Note that plane search can be used to speed up BCD techniques as well [33].

Another effective globalization strategy uses a TR. There, the problems of finding the step direction $\boldsymbol{p}^{(t)}$ and the step-size are combined; i.e., $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \boldsymbol{p}^{(t)}$ with

$$\boldsymbol{p}^{(t)} = \underset{\boldsymbol{p}}{\operatorname{argmin}}\, m(\boldsymbol{p}) \quad \text{s.t.} \quad ||\boldsymbol{p}|| \leq \Delta, \tag{19}$$

where $m(\boldsymbol{p}) = 1/2\,\|\boldsymbol{f}^{(t)}\|^2 + \boldsymbol{g}^{(t)\top}\boldsymbol{p} + 1/2\boldsymbol{p}^{\top}\boldsymbol{\Phi}^{(t)}\boldsymbol{p}$ under the GN framework. Intuitively, the TR is employed to prevent the GN steps from being too aggressive to miss the contraction region of a stationary point. The TR radius Δ is heuristically determined by measuring how well the model predicts the decrease in the function value [28]. Often, the search space is restricted to a 2D subspace spanned by $\boldsymbol{g}^{(t)}$ and $\boldsymbol{p}^{(t)}$. Then, (19) can be approximately solved using the dogleg step or by using plane search [31], [33].

Exploiting tensor structure

The bottleneck operation in the GN approach involves constructing and solving the linear system in (17); i.e.,

$$\boldsymbol{J}^{\top}\boldsymbol{J}\boldsymbol{p} = -\boldsymbol{g}, \tag{20}$$

where the superscripts $(\cdot)^{(t)}$ have been dropped for the simplicity of notation. Note that this system is easily large scale since $\boldsymbol{J} \in \mathbb{R}^{\prod_n I_n \times T}$, where $T = R(\Sigma_{n=1}^{I} I_n)$. Using a general-purpose solver for this system costs $O(T^3)$ flops, which may be prohibitive for big data problems. Fortunately, the Jacobian and the Gramian are both structured under certain decomposition models (e.g., CPD and LL1) that can be exploited to come up with lightweight solutions.

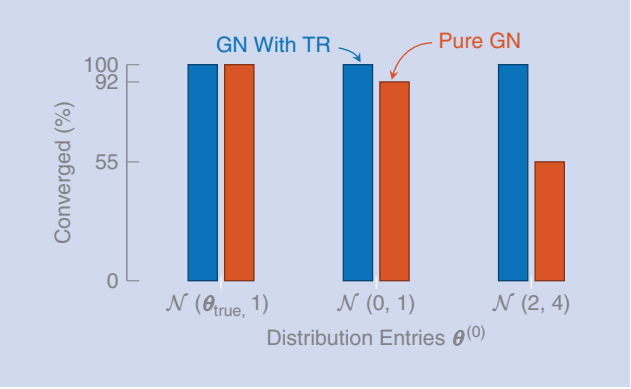


FIGURE 6. Without a globalization strategy, the pure GN does not always converge to a stationary point; by using a dogleg TR-based globalization, GN converges for every initialization $\theta^{(0)}$. Results are shown for a $20 \times 20 \times 20$, rank-10 tensor in which all factor matrix entries θ_{true} are drawn from the normal distribution $\mathcal{N}(0,1)$ and for 100 different initializations for three scenarios ranging from good (left) to bad (right).

The gradient $\boldsymbol{g} = \nabla f(\boldsymbol{\theta})$ of $f(\boldsymbol{\theta})$ can be partitioned as $\boldsymbol{g} = [\text{vec}(\boldsymbol{G}_1); \ldots; \text{vec}(\boldsymbol{G}_N)]$, in which the $\boldsymbol{G}_n$ with respect to factor matrix $\boldsymbol{A}_n$ is given by

$$\boldsymbol{G}_n = \boldsymbol{F}_n^{\top}\boldsymbol{H}_n, \tag{21}$$

in which $\boldsymbol{F}_n$ is the unfolding of the residual $\mathcal{F}$; see (8). The operation $\boldsymbol{F}_n^{\top}\boldsymbol{H}_n$ is the well-known MTTKRP, as we have seen in the BCD approaches. However, the factor matrices $\boldsymbol{A}_n$ $(n = 1, \ldots, N)$ have the same value for every gradient $\boldsymbol{G}_n$, in contrast to BCD algorithms, which use updated variables in every inner iteration. This can be exploited to reduce the computational cost [34].

Similar to the gradient, the Jacobian $\boldsymbol{J}$ can also be partitioned as $\boldsymbol{J} = [\boldsymbol{J}_1, \ldots, \boldsymbol{J}_N]$, in which $\boldsymbol{J}_n = \partial\text{vec}(\mathcal{F})/\partial\text{vec}(\boldsymbol{A}_n)$:

$$\boldsymbol{J}_n = \boldsymbol{\Pi}_n(\boldsymbol{H}_n \otimes \boldsymbol{I}_{I_n}), \tag{22}$$

in which $\boldsymbol{\Pi}_n$ is a matrix corresponding to the permutation of mode 1 to mode n of the vectorized tensors. By exploiting the block and Kronecker structures, constructing $\boldsymbol{\Phi}$ requires only $O(T^2)$ flops, as opposed to $O(T^3)$; for details, see [32] and [35].

Instead of solving (20) exactly, an iterative solver, such as conjugate gradient (CG) methods, can be used. As in power iterations, the key step in a CG iteration is a Gramian-vector product, i.e., given $\boldsymbol{\upsilon}$, compute $\boldsymbol{y}$ as

$$\boldsymbol{y} = \boldsymbol{J}^{\top}\boldsymbol{J}\boldsymbol{\upsilon}. \tag{23}$$

Both $\boldsymbol{y}$ and $\boldsymbol{\upsilon}$ can be partitioned according to the variables; hence, $\boldsymbol{\upsilon} = [\boldsymbol{\upsilon}_1; \ldots; \boldsymbol{\upsilon}_N]$ and $\boldsymbol{y} = [\boldsymbol{y}_1; \ldots; \boldsymbol{y}_N]$. Then, (23) can be written as $\boldsymbol{y}_n = \boldsymbol{J}_n^{\top}\Sigma_{k=1}^{N}\boldsymbol{J}_k\boldsymbol{\upsilon}_k$, which is efficiently computed by exploiting the structure in $\boldsymbol{J}_n$ [see (22)]:

$$\boldsymbol{Y}_n = \boldsymbol{V}_n\boldsymbol{W}_n + \boldsymbol{A}_n \sum_{\substack{m=1 \\ m \neq n}}^{N} \boldsymbol{W}_{mn} \circledast (\boldsymbol{V}_k^{\top}\boldsymbol{A}_k), \tag{24}$$

where $\boldsymbol{Y}_n, \boldsymbol{V}_n \in \mathbb{R}^{I_n \times R}$, $\boldsymbol{y}_n = \text{vec}(\boldsymbol{Y}_n)$, and $\boldsymbol{\upsilon}_n = \text{vec}(\boldsymbol{V}_n)$, respectively. Here, $\boldsymbol{W}_n$ and $\boldsymbol{W}_{mn}$ are defined as follows:

$$\boldsymbol{W}_n = \mathop{\circledast}_{\substack{k=1 \\ k \neq n}}^{N} \boldsymbol{A}_k^{\top}\boldsymbol{A}_k, \quad \boldsymbol{W}_{mn} = \mathop{\circledast}_{\substack{k=1 \\ k \neq m,n}}^{N} \boldsymbol{A}_k^{\top}\boldsymbol{A}_k. \tag{25}$$

Hence, to compute $\boldsymbol{J}^{\top}\boldsymbol{J}\boldsymbol{\upsilon}$ only products of small $I_n \times R$ and $R \times R$ matrices are required. Since CG performs a number of iterations with constant $\boldsymbol{A}_n$, the inner products $\boldsymbol{A}_n^{\top}\boldsymbol{A}_n$ required for $\boldsymbol{W}_n$ and $\boldsymbol{W}_{mn}$ can be precomputed. This way, the complexity per Gramian-vector product is only $O(R^2\Sigma_n I_n)$. Note that for both the GN and the BCD methods, the computation of the gradient, which requires $O(R\Pi_{n=1}^{N} I_n)$ operations, usually dominates the complexity. Therefore, the GN approach is also an excellent candidate for parallel implementations, as it reduces the number of iterations and expensive gradient computations, while the extra CG iterations have a negligible communication overhead.

In practice, it is important to notice that a well-conditioned $\boldsymbol{J}^\top\boldsymbol{J}$ makes solving the system in (20) much faster using CG. In numerical linear algebra, the common practice is to precondition $\boldsymbol{J}^\top\boldsymbol{J}$, leading to the so-called preconditioned CG (PCG) paradigm. Preconditioning can be done rather efficiently under some LRDMs, such as CPD; see "Acceleration via Preconditioning."

Structured decomposition

As mentioned, the GN framework is specialized for NLS problems; i.e., objectives can be written as $\|\mathcal{F}(\boldsymbol{\theta})\|_{\mathrm{F}}^2$. If there are structural constraints on $\boldsymbol{\theta}$, incorporating such structural requirements is often nontrivial. In this section, we introduce a number of ideas for handling structural constraints under the GN framework.

Parametric constraints

One way to handle constraints is to use parametrization to convert the constrained decomposition problem to an unconstrained NLS problem. To see how it works, let us consider the case where $h_n(\boldsymbol{\theta}_n)$ is an indicator function of set $\mathcal{C}_n$, i.e., the constrained decomposition case where $\boldsymbol{\theta}_n \in \mathcal{C}_n$. In addition, we assume that every element in $\mathcal{C}_n$ can be parameterized by an unconstrained variable. Assume $\boldsymbol{\theta}_n = \mathrm{vec}(\boldsymbol{A}_n)$ and that every factor matrix $\boldsymbol{A}_n$ is a function q_n of a disjoint set of parameters $\boldsymbol{\alpha}_n$; i.e., $\boldsymbol{A}_n = q_n(\boldsymbol{\alpha}_n)$, $n = 1, \ldots, N$. For example, if $\mathcal{C}_n = \mathbb{R}_+^{I_n \times R}$, i.e., the nonnegative orthant, one can parameterize $\boldsymbol{A}_n$ using the following:

$$\boldsymbol{A}_n = \boldsymbol{D} \circledast \boldsymbol{D},\ \boldsymbol{D} \in \mathbb{R}^{I_n \times R}.$$

In this case, $\boldsymbol{\alpha}_n = \mathrm{vec}(\boldsymbol{D})$, and $q_n(\cdot): \mathbb{R}^{I_n R} \to \mathbb{R}^{I_n R}$ denotes the elementwise squaring operation. If no constraint is imposed on some $\boldsymbol{A}_n$, $q_n(\boldsymbol{\alpha}_n) = \mathrm{unvec}(\boldsymbol{\alpha}_n)$; see many other examples for different constraints in [35].

By substituting the constraints in the optimization problem (13), we obtain a problem in variables $\boldsymbol{\alpha} = [\boldsymbol{\alpha}_1; \ldots; \boldsymbol{\alpha}_N]$:

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \left\| \mathcal{T} - [\![q_1(\boldsymbol{\alpha}_1), \ldots, q_N(\boldsymbol{\alpha}_N)]\!] \right\|_{\mathrm{F}}^2. \tag{26}$$

Applying GN to (26) follows the same steps as before. Central to this unconstrained problem is the solution of

$$\tilde{\boldsymbol{\Phi}}\tilde{\boldsymbol{p}} = -\tilde{\boldsymbol{g}}, \tag{27}$$

where we denote quantities related to parameters by tildes to distinguish them from quantities related to factor matrices. The structure of the factorization models can still be exploited if we use the chain rule for derivation [36]. This way, (27) can be written as

$$\tilde{\boldsymbol{J}}^\top \boldsymbol{\Phi} \tilde{\boldsymbol{J}} \tilde{\boldsymbol{p}} = -\tilde{\boldsymbol{J}}^\top \boldsymbol{g}, \tag{28}$$

in which $\boldsymbol{\Phi}$ and $\boldsymbol{g}$ are exactly the expressions as derived before in the unconstrained case. The Jacobian $\tilde{\boldsymbol{J}}$ is a block-diagonal matrix containing the Jacobian of each factor matrix with respect to the underlying variables; i.e.,

$$\tilde{\boldsymbol{J}} = \mathrm{blkdiag}(\tilde{\boldsymbol{J}}_1, \ldots, \tilde{\boldsymbol{J}}_N). \tag{29}$$

The Jacobians $\tilde{\boldsymbol{J}}_n$ are often straightforward to derive. For example, if $\boldsymbol{A}_n$ is unconstrained, $\tilde{\boldsymbol{J}}_n = \boldsymbol{I}_{I_n R}$; if nonnegativity is imposed by squaring variables, $\boldsymbol{A}_n = \boldsymbol{D} \circledast \boldsymbol{D}$, and $\tilde{\boldsymbol{J}}_n = \mathrm{diag}(\mathrm{vec}(2\boldsymbol{D}))$; in the case of linear constraints, e.g., $\boldsymbol{A}_n = \boldsymbol{B}\boldsymbol{X}\boldsymbol{C}$ with $\boldsymbol{B}$ and $\boldsymbol{C}$ known, $\tilde{\boldsymbol{J}}_n = \boldsymbol{C}^\top \otimes \boldsymbol{B}$. More complicated constraints can be modeled via composite functions and by repeatedly applying the chain rule [35], [36].

When computing the Gramian or Gramian-vector products in (28), we can exploit the multilinear structure from the CPD as well as the block-diagonal structure of the constraints. Moreover, depending on the constraint, $\tilde{\boldsymbol{J}}_n$ may, for example, also have a diagonal or Kronecker product structure. Therefore, the Gramian-vector products can be computed in three steps:

$$\boldsymbol{\upsilon}_n = \tilde{\boldsymbol{J}}_n \tilde{\boldsymbol{\upsilon}}_n, \quad \boldsymbol{y}_n = \boldsymbol{J}_n^\top \sum_{k=1}^{N} \boldsymbol{J}_k \boldsymbol{\upsilon}_k, \quad \tilde{\boldsymbol{y}}_n = \tilde{\boldsymbol{J}}_n^\top \boldsymbol{y}_n, \tag{30}$$

Acceleration via Preconditioning

As the convergence speed of CG depends on the "clustering of the eigenvalues" of $\boldsymbol{J}^\top\boldsymbol{J}$, a preconditioner is often applied to improve this clustering. More specifically, the system

$$\boldsymbol{M}^{-1}\boldsymbol{J}^\top\boldsymbol{J}\boldsymbol{p} = -\boldsymbol{M}^{-1}\boldsymbol{g} \tag{S1}$$

is solved instead of (20), and the preconditioner $\boldsymbol{M}$ is chosen to reduce the computational cost. In practice, a Jacobi preconditioner, a diagonal matrix with entries $\mathrm{diag}(\boldsymbol{J}^\top\boldsymbol{J})$, or a block-Jacobi preconditioner, i.e., a block-diagonal approximation to $\boldsymbol{J}^\top\boldsymbol{J}$, are often effective for the unconstrained CPD [31]. For example, the latter preconditioner is given by

$$\boldsymbol{M}_{\mathrm{BJ}} = \mathrm{blkdiag}(\boldsymbol{W}_1 \otimes \boldsymbol{I}_{I_1}, \ldots, \boldsymbol{W}_N \otimes \boldsymbol{I}_{I_N}). \tag{S2}$$

Because of the block-diagonal and the Kronecker structure in $\boldsymbol{M}_{\mathrm{BJ}}$, the system $\boldsymbol{\upsilon} = \boldsymbol{M}_{\mathrm{BJ}}^{-1}\boldsymbol{y}$ can be solved in N steps; i.e., $\boldsymbol{V}_n = \boldsymbol{Y}_n\boldsymbol{W}_n^{-1}$ for $n = 1, \ldots, N$. Applying $\boldsymbol{M}_{\mathrm{BJ}}^{-1}$ involves only inverses of small $R \times R$ matrices, which are constant in one GN iteration. Interestingly, $\boldsymbol{M}_{\mathrm{BJ}}$ appears in the ALS algorithm with simultaneous updates, i.e., without updating $\boldsymbol{A}_n$ every inner iteration. The PCG algorithm can therefore be seen as a refinement of the ALS with simultaneous updates by taking the off-diagonal blocks into account [31].

which may all be efficiently computed using similar ideas as in the unconstrained case [compare with (24)]. By leveraging the chain rule and the Gramian-vector-product-based PCG method for handling the unconstrained GN framework, it turns out that many frequently used constraints in signal processing and data analytics can be handled under this framework in an efficient way. Examples include nonnegativity, polynomial constraints, orthogonality, matrix inverses, Vandermonde, and Toeplitz or Hankel structures; see details in [35]. We should mention that the parametrization technique can also handle some special constraints that are considered quite challenging in the context of tensor and matrix factorization, e.g., (partial) symmetry and coupling constraints; see "Handling Special Constraints via Parametrization."

Proximal GN

To handle more constraints and the general cost function $f(\boldsymbol{\theta}) + h(\boldsymbol{\theta})$ in a systematic way, one may also employ the proximal GN (ProxGN) approach. To be specific, in the presence of a nonsmooth $h(\boldsymbol{\theta})$, the ProxGN framework modifies the per-iteration subproblem of GN into

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{2} \left\| \boldsymbol{f}^{(t)} + \boldsymbol{J}^{(t)} \left(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}\right) \right\|_2^2 + h(\boldsymbol{\theta}). \tag{31}$$

This is conceptually similar to the PGD approach: linearizing the smooth part (using the same linearization as in unconstrained GN) while keeping the nonsmooth regularization term untouched. The subproblem in (31) is again a regularized LS problem with respect to $\boldsymbol{\theta}$. Similar to the BCD case [see (9)], there exists no closed-form solution for the subproblem, in general. However, subproblem solvers, such as PGD and the ADMM, can again be employed to handle (31).

A recent theoretical study has shown that incorporating the proximal term does not affect the overall superlinear convergence rate of the GN-type algorithms within the vicinity of the solution. The challenge, however, is to solve (31) in the context of SLRD with lightweight updates. This is possible. The recent paper in [39] has shown that if the ADMM is employed, then the key steps for solving (31) are essentially the same as for the unconstrained GN, namely, computing $(\boldsymbol{J}^{(t)\top}\boldsymbol{J}^{(t)} + \rho \boldsymbol{I})^{-1}$ for a certain $\rho > 0$ once per ProxGN iteration. Note that this step is nothing but inverting the regularized Jacobian Gramian, which, as we have seen, admits a number of economical solutions. In addition, with judiciously designed ADMM steps, this Gramian inversion never needs to be instantiated, and the algorithm is memory efficient, as well; see details in [39] for an implementation for NMF.

Stochastic approaches

Batch algorithms, such as BCD and GN, could have serious memory and computational issues, especially when the data tensor or matrix is large and dense. Recall that the MTTKRP (i.e., $\boldsymbol{H}_n^\top \boldsymbol{X}_n$) costs $O(R\Pi_{n=1}^N I_n)$ operations if no structure of the tensor can be exploited. This is quite expensive for large I_n and high-order tensors. For big data problems, stochastic optimization is a classic workaround for avoiding memory/operation

Handling Special Constraints via Parameterization

Factorizations are often (partially) symmetric, e.g., in blind-source separation and topic modeling (see examples in the tutorial [2]). Symmetry here means that some $\boldsymbol{A}_n$s are identical. The conventional BCD treating each $\boldsymbol{A}_n$ as a block is not straightforward anymore. For example, cyclically updating the factor matrices $\boldsymbol{A}_1 = \boldsymbol{A}_2 = \boldsymbol{A}_3 = \boldsymbol{A}$ in the decomposition $[\![\boldsymbol{A}, \boldsymbol{A}, \boldsymbol{A}]\!]$ breaks symmetry, while the subproblems are no longer convex when enforcing symmetry; see the "Block Splitting and Ordering Within BCD" section. Nevertheless, the GN framework handles such constraints rather naturally.

A (partially) symmetric CPD can be modeled by setting two or more factors to be identical. For example, consider the model $[\![\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{N-2}, \boldsymbol{A}_{N-1}, \boldsymbol{A}_{N-1}]\!]$; i.e., the last two factor matrices are identical. This symmetry constraint leads to a $\tilde{\boldsymbol{J}}$ with the following form:

$$\tilde{\boldsymbol{J}} = \text{blkdiag}\left(\boldsymbol{I}_{I_1 R}, \ldots, \boldsymbol{I}_{I_{N-2} R}, [\boldsymbol{I}_{I_{N-1} R}; \boldsymbol{I}_{I_{N-1} R}]\right),$$

in which $\tilde{\boldsymbol{J}}_n$, $n = 1, \ldots, N-1$ are identity matrices, as no constraints are imposed on $\boldsymbol{A}_n$. Because of the structure in $\tilde{\boldsymbol{J}}$, the extra steps in the Gramian-vector products in (23) only involve summations.

Coupled decomposition often arises in data fusion, e.g., integrating hyperspectral and multispectral images for superresolution purposes [37], spectrum cartography from multiple-sensor-acquired spatiospectral information [38], and jointly analyzing primary data and side information [15]. These problems involve mutually factorizing tensors and/or matrices: the decompositions can share, or are coupled through, one or more factors or underlying variables. For example, consider the coupled matrix tensor factorization problem:

$$\min_{\{\boldsymbol{A}_n\}_{n=1}^4} \frac{\lambda_1}{2} \|[\![\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{A}_3]\!] - \mathcal{T}\|_F^2 + \frac{\lambda_2}{2} \|[\![\boldsymbol{A}_3, \boldsymbol{A}_4]\!] - \boldsymbol{M}\|_F^2,$$

where the two terms are coupled through $\boldsymbol{A}_3$. Coupled decomposition can be handled via BCD. However, in some cases, the key steps of BCD boil down to solving Sylvester equations in each iteration, which can be costly for large-scale problems [37]. Using parametrization and GN, the influence of the coupling constraint and the decomposition can be separated in the CG iterations [35], [36], thus easily putting forth efficient and flexible data fusion algorithms. This serves as the foundation of the structured data fusion toolbox in `Tensorlab`.

explosion. In a nutshell, stochastic algorithms are particularly suitable for handling problems having the following form:

$$\min_{\boldsymbol{\theta}} \frac{1}{L}\sum_{\ell=1}^{L} f_\ell(\boldsymbol{\theta}) + h(\boldsymbol{\theta}), \tag{32}$$

where the first term is often called the *empirical risk function* in the literature. The classic stochastic proximal gradient descent (SPGD) updates the optimization variables via

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \mathsf{Prox}_h\big(\boldsymbol{\theta}^{(t)} - \alpha^{(t)}\boldsymbol{g}(\boldsymbol{\theta}^{(t)})\big), \tag{33}$$

where $\boldsymbol{g}(\boldsymbol{\theta}^{(t)})$ is a random vector (or "stochastic oracle") evaluated at $\boldsymbol{\theta}^{(t)}$, constructed through a RV $\xi^{(t)}$. The idea is to use an easily computable stochastic oracle to approximate the computationally expensive full gradient $\nabla f(\boldsymbol{\theta}) = (1/N)\sum_{\ell=1}^{L}\nabla f_\ell(\boldsymbol{\theta})$, so that (33) serves as an economical version of the PGD algorithm. A popular choice is $\boldsymbol{g}(\boldsymbol{\theta}^{(t)}) = \nabla f_\ell(\boldsymbol{\theta}^{(t)})$, where $\ell \in \{1,\ldots,L\}$ is randomly selected following the PMF $\mathsf{Pr}(\xi^{(t)} = \ell) = 1/L$. This simple construction has a nice property: $\boldsymbol{g}(\boldsymbol{\theta}^{(t)})$ is an unbiased estimate for the full gradient given the history of random sampling; i.e.,

$$\nabla f(\boldsymbol{\theta}^{(t)}) = \frac{1}{L}\sum_{\ell=1}^{L}\nabla f_\ell(\boldsymbol{\theta}^{(t)}) = \mathbb{E}_{\xi^{(t)}}\big[\boldsymbol{g}(\boldsymbol{\theta}^{(t)})\big|\mathcal{H}^{(t)}\big],$$

where $\mathcal{H}^{(t)}$ collects all the RVs appearing before iteration t. The unbiasedness is often instrumental in establishing the convergence of stochastic algorithms. Biased stochastic oracles and their convergence properties are also discussed in the literature; see, e.g., [17]. However, the analysis is more involved. In addition, some conditions (e.g., the bounded bias) are not easy to verify. Another very important aspect is the variance of $\boldsymbol{g}(\boldsymbol{\theta}^{(t)})$. Assume that the variance is bounded; i.e., $\mathbb{V}\big[\boldsymbol{g}(\boldsymbol{\theta}^{(t)})\big|\mathcal{H}^{(t)}\big] \le \tau$. Naturally, one hopes τ to be small so that the average deviation of $\boldsymbol{g}(\boldsymbol{\theta}^{(t)})$ from the full gradient is small, and thus the SPGD algorithm will behave more like the PGD algorithm. A smaller τ can be obtained via using more samples to construct $\boldsymbol{g}(\boldsymbol{\theta}^{(t)})$, e.g., using

$$\boldsymbol{g}(\boldsymbol{\theta}^{(t)}) = \frac{1}{\big|\mathcal{B}^{(t)}\big|}\sum_{\ell\in\mathcal{B}^{(t)}}\nabla f_\ell(\boldsymbol{\theta}^{(t)}),$$

where $\mathcal{B}^{(t)}$ denotes the index set of the $f_\ell(\boldsymbol{\theta}^{(t)})$s sampled at iteration t. This leads to the so-called minibatch scheme. Note that if $\big|\mathcal{B}^{(t)}\big| = L$, then $\tau = 0$ and SPGD becomes PGD. As we have mentioned, a smaller τ would make the convergence properties of SPGD more like PGD, and thus a larger $\big|\mathcal{B}^{(t)}\big|$ is more preferred in terms of variance reduction. However, a larger $\big|\mathcal{B}^{(t)}\big|$ leads to more operations for computing the stochastic oracle. In practice, this is a tradeoff that often requires some tuning to balance.

The randomness of stochastic algorithms makes characterizing the convergence properties of a single sequence of $\{\boldsymbol{\theta}^{(t)}\}_t$ meaningless. Instead, the "expected convergence properties" are often used. For example, when $h(\boldsymbol{\theta})$ is absent, a convergence criterion of interest is expressed as follows:

$$\liminf_{t\to\infty}\mathbb{E}\big[\|\nabla f(\boldsymbol{\theta}^{(t)})\|_2^2\big] = 0, \tag{34}$$

where the expectation is taken across all the RVs that were used for constructing the stochastic oracles for all the iterations (i.e., the "total expectation"). Equation (34) means that every limit point of $\{\boldsymbol{\theta}^{(t)}\}$ is a stationary point in expectation. When $h(\boldsymbol{\theta})$ is present, similar ideas are utilized. Recall that $\mathbf{0} \in \partial F(\boldsymbol{\theta}^{(t)})$ is a necessary condition for attaining a stationary point [see (6)]. In [17], the expected counterpart of (6), i.e.,

$$\liminf_{t\to\infty}\mathbb{E}\big[\mathrm{dist}\big(\mathbf{0},\partial F(\boldsymbol{\theta}^{(t)})\big)\big] = 0, \tag{35}$$

is employed for establishing the notion of stationary-point convergence for nonconvex, nonsmooth problems under the stochastic settings. For both (34) and (35), when some more assumptions hold (e.g., the solution sequence is bounded), the "inf" notation can be removed, meaning that the whole sequence converges to a stationary point on average.

Entry sampling

Many SLRD problems can be re-expressed in a similar form as that in (32). One can rewrite the constrained CPD problem under the LS fitting loss as follows:

$$\min_{\{\boldsymbol{A}_n\}_{n=1}^N}\frac{1}{L}\sum_{i_1=1}^{I_1}\cdots\sum_{i_N=1}^{I_N} f_{i_1,\ldots,i_N}(\boldsymbol{\theta}) + h(\boldsymbol{\theta}), \tag{36}$$

where $L = \prod_{n=1}^{N} I_n$, $h(\boldsymbol{\theta}) = \sum_{n=1}^{N} h_n(\boldsymbol{A}_n)$ and $f_{i_1,\ldots,i_N}(\boldsymbol{\theta}) = \big(\mathcal{T}(i_1,\ldots,i_N) - \sum_{r=1}^{R}\prod_{n=1}^{N}\boldsymbol{A}_n(i_n,r)\big)^2$. Assume $\mathcal{B}^{(t)}$ is a set of indices of the tensor entries that are randomly sampled [see Figure 7(a)]. The corresponding SPGD update is as follows: $\boldsymbol{\theta}^{(t+1)}$ is given by

$$\mathsf{Prox}_h\Bigg(\boldsymbol{\theta}^{(t)} - \frac{\alpha^{(t)}}{\big|\mathcal{B}^{(t)}\big|}\sum_{(i_1,\ldots,i_N)\in\mathcal{B}^{(t)}}\nabla f_{i_1,\ldots,i_N}(\boldsymbol{\theta}^{(t)})\Bigg). \tag{37}$$

It is not difficult to see that many entries of $\nabla f_{i_1,\ldots,i_N}(\boldsymbol{\theta}^{(t)})$ are zero since $\nabla f_{i_1,\ldots,i_N}(\boldsymbol{\theta}^{(t)})$ contains only the information of $\boldsymbol{A}_n(i_n,:)$; we have $\big[\nabla f_{i_1,\ldots,i_N}(\boldsymbol{\theta}^{(t)})\big]_g = 0$ for all $\theta_g \notin \{\boldsymbol{A}_n(i_n,r)\,|\,(i_1,\ldots,i_N)\in\mathcal{B}^{(t)}\}$. The derivative with respect to $\boldsymbol{A}_n(i_n,:)$ for the sampled indices is easy to compute; see [2]. This is essentially the idea in [15] for coupled tensor and matrix decompositions. This kind of sampling strategy ensures that the constructed stochastic oracle is an unbiased estimation for the full gradient, and it features very lightweight updates. Computing the term $\Sigma_{\mathcal{B}^{(t)}}\nabla f_{i_1,\ldots,i_N}(\boldsymbol{\theta}^{(t)})$ requires only $O(R\big|\mathcal{B}^{(t)}\big|)$ operations, instead of $O(R\prod_{n=1}^{N} I_n)$ operations for computing the full gradient.

Subtensor sampling

Entry sampling-based approaches are direct applications of conventional SPGD for tensor decomposition. However, these methods do not leverage existing tensor decomposition tools. One way to take advantage of existing tensor decomposition algorithms is sampling subtensors, instead of entries. The

randomized block sampling (RBS) algorithm [40] considers the unconstrained CPD problem. The algorithm samples a subtensor

$$\mathcal{T}_{\text{sub}}^{(t)} = \mathcal{T}(\mathcal{S}_1, \ldots, \mathcal{S}_N) \approx \sum_{r=1}^{R} \boldsymbol{A}_1(\mathcal{S}_1, r) \circ \ldots \circ \boldsymbol{A}_N(\mathcal{S}_N, r)$$

at every iteration t and updates the latent variables by computing one optimization step using

$$\begin{aligned} \boldsymbol{\theta}_{\text{sub}}^{(t+1)} &\leftarrow \underset{\boldsymbol{\theta}_{\text{sub}}}{\operatorname{argmin}} \| \mathcal{T}_{\text{sub}}^{(t)} - [\![\boldsymbol{A}_1^{\text{sub}}, \ldots, \boldsymbol{A}_N^{\text{sub}}]\!] \|_{\text{F}}^2, \\ \boldsymbol{\theta}_{-\text{sub}}^{(t+1)} &\leftarrow \boldsymbol{\theta}_{-\text{sub}}^{(t)}, \end{aligned} \tag{38}$$

where all the variables affected by $\mathcal{T}_{\text{sub}}^{(t)}$ are collected in $\boldsymbol{\theta}_{\text{sub}} = [\text{vec}(\boldsymbol{A}_1^{\text{sub}}); \ldots, \text{vec}(\boldsymbol{A}_N^{\text{sub}})]$, $\boldsymbol{A}_n^{\text{sub}} = \boldsymbol{A}_n(\mathcal{S}_n, :)$ and $\boldsymbol{\theta}_{-\text{sub}}$ contains all the other optimization variables. Since each update in (40) involves one step in a common tensor decomposition problem, many off-the-shelf algorithms, such as ALS or GN, can be leveraged [40].

The algorithm in (38) works well, especially when the tensor rank is low and the sampled subtensors already have identifiable latent factors; under such cases, the estimated $\boldsymbol{A}_n^{\text{sub}}$ from subtensors can serve as a good estimate for the corresponding part of $\boldsymbol{A}_n$ after one or two updates. In practice, one does not need to exactly solve the subproblems in (38). Combining some TR considerations, the work in [40] suggested using a one-step GN or a one-step regularized ALS to update $\boldsymbol{\theta}_{\text{sub}}$. Note that the sampled subtensors are typically not independent under this framework since one wishes to update every unknown parameter in an equally frequent way; see [40]. This is quite different from established conventions in stochastic optimization, which makes convergence analysis for RBS more challenging than the entry sampling-based methods.

Fiber sampling

In principle, the entry sampling and SPGD idea in (37) can handle any $h(\cdot)$ that admits simple proximal operators. In addition, the RBS algorithm can be applied with any constraint compatible with the GN framework. However, such sampling strategies are no longer viable when it comes to constraints/regularization terms that are imposed on the columns of the latent factors, e.g., the probability simplex constraint that is often used in statistical learning ($\mathbf{1}^\top \boldsymbol{A}_n = \mathbf{1}^\top$, $\boldsymbol{A}_n \geq 0$), the constraint $\| \boldsymbol{A}_n \|_{2,1} = \sum_{i_n=1}^{I_N} \| \boldsymbol{A}_n(i_n, :) \|_2$ used for promoting row sparsity, and the total variation/smoothness regularization terms on the columns of $\boldsymbol{A}_n$. The reason is that $\mathcal{T}_{\text{sub}}$ contains only information about $\boldsymbol{A}_n(\mathcal{S}_n, :)$, which means that enforcing column constraints on $\boldsymbol{A}_n$ is not possible if updates in (37) or (38) are employed.

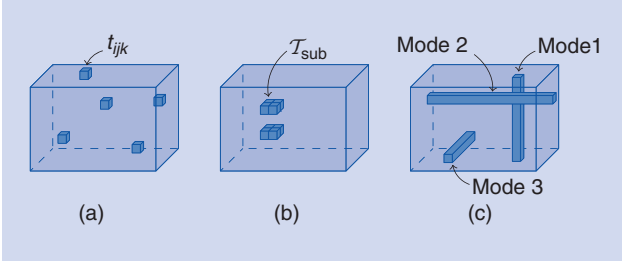


FIGURE 7. Various sampling strategies used in stochastic optimization algorithms. (a) Entry. (b) Subtensor. (c) Fiber.

Recently, the work in [27] and [41] advocated sampling (a set of) mode-n "fibers" for updating $\boldsymbol{A}_n$. A mode-n fiber of the tensor $\mathcal{T}$ is an I_n-dimensional vector that is obtained by varying the mode-n index while fixing others of $\mathcal{T}$ [see Figure 7(c)]. The interesting connection here is that

$$\underbrace{\mathcal{T}(i_1, \ldots, i_{n-1}, :, i_{n+1}, \ldots, i_N)}_{\text{a mode}-n \text{ fiber}} = \boldsymbol{X}_n(j_n, :),$$

where $j_n = 1 + \Sigma_{\ell=1, \ell \neq n}(i_\ell - 1) J_\ell$ and $J_\ell = \Pi_{m=1, m \neq n}^{\ell-1} I_m$. Under this sampling strategy, the whole $\boldsymbol{A}_n$ can be updated in one iteration. Specifically, in iteration t, the work in [41] sequentially updates $\boldsymbol{A}_n$ for $n = 1, \ldots, N$, as in the BCD case. To update $\boldsymbol{A}_n$, it samples a set of mode-n fibers, indexed by $\mathcal{Q}_n^{(t)}$, and solves a "sketched LS" problem,

$$\min_{\boldsymbol{A}_n} \left\| \boldsymbol{X}_n(\mathcal{Q}_n^{(t)}, :) - \boldsymbol{H}_n^{(t)}(\mathcal{Q}_n^{(t)}, :) \boldsymbol{A}_n^\top \right\|_{\text{F}}^2, \tag{39}$$

whose solution is

$$\boldsymbol{A}_n^{(t+1)} \leftarrow (\boldsymbol{H}_n^{(t)}(\mathcal{Q}_n^{(t)}, :)^\dagger \boldsymbol{X}_n(\mathcal{Q}_n^{(t)}, :))^\top.$$

This simple sampling strategy makes sure that every entry of $\boldsymbol{A}_n$ can be updated in iteration t. The rationale behind it is also reasonable: if the tensor is low-rank, then one does not need to use all the data to solve the LS subproblems; using randomly sketched data is enough. If the system of linear equations $\boldsymbol{X}_n = \boldsymbol{H}^{(t)}(\mathcal{Q}_n^{(t)}, :) \boldsymbol{A}_n^\top$ is overdetermined, it returns the same solution as solving $\boldsymbol{X}_n = \boldsymbol{H}^{(t)} \boldsymbol{A}_n^\top$.

The work in [41] did not explicitly consider structural information on $\boldsymbol{A}_n$s, and the convergence properties of the approach are unclear. To incorporate structural information and to establish convergence, the recent work in [27] offered a remedy. There, a block-randomized sampling strategy was proposed to help establish the unbiasedness of the gradient estimation. Then, PGD is combined with fiber sampling for handling structural constraints. The procedure consists of two sampling stages: first, randomly sample a mode $n \in \{1, \ldots, N\}$ with random seed $\zeta^{(t)}$ such that $\Pr(\zeta^{(t)} = n) = 1/N$: then, sample a set of mode-n fibers indexed by $\mathcal{Q}_n$ uniformly at random (with another random seed $\xi^{(t)}$). Using the sampled data, construct

$$\boldsymbol{G}^{(t)} = [\boldsymbol{G}_1^{(t)}; \ldots; \boldsymbol{G}_N^{(t)}], \tag{40}$$

where $\boldsymbol{G}_n^{(t)} = \boldsymbol{A}_n \boldsymbol{B}^\top \boldsymbol{B} - \boldsymbol{X}_n(\mathcal{Q}_n, :)^\top \boldsymbol{B}$, with $\boldsymbol{B} = \boldsymbol{H}^{(t)}(\mathcal{Q}_n^{(t)}, :)$, and $\boldsymbol{G}_k^{(t)} = 0$ for $k \neq n$. This block-randomization technique entails the following equality:

$$\text{vec}\left(\mathbb{E}_{\xi^{(t)}} \left[\boldsymbol{G}^{(t)} \middle| \mathcal{H}^{(t)}, \zeta^{(t)} \right] \right) = c \nabla f(\boldsymbol{\theta}^{(t)}), \tag{41}$$

where $c > 0$ is a constant; i.e., the constructed stochastic vector is an unbiased estimation (up to a constant scaling factor) for the full gradient, conditioned on the filtration. Then, the algorithm updates the latent factors via

$$\boldsymbol{A}_n^{(t+1)} \leftarrow \text{Prox}_{h_n}(\boldsymbol{A}_n^{(t)} - \alpha^{(t)} \boldsymbol{G}_n^{(t)}). \tag{42}$$

Because of (41), the algorithm is almost identical to single-block SPGD and thus enjoys similar convergence guarantees [27].

Fiber-sampling approaches, as in [41] and [27], are economical since they never need to instantiate the large matrix $\boldsymbol{H}_n$ or compute the full MTTKRP. One remark is that fiber sampling is also of interest in partially observed tensor recovery [38], [42]; in the "Tractable SLRD Problems and Algorithms" section, it will actually be argued that under mild conditions, the exact completion of a fiber-sampled tensor is possible via a matrix eigenvalue decomposition [43].

Adaptive step-size scheduling

Implementing stochastic algorithms often requires somewhat intensive hands-on tuning for selecting hyperparameters, in particular, the step-size $\alpha^{(t)}$. Generic stochastic gradient descent and SPGD analyses suggest setting the step-size sequence by following Robbins and Monro's rule; i.e., $\Sigma_{t=0}^{\infty}\alpha^{(t)}=\infty$, $\Sigma_{t=0}^{\infty}(\alpha^{(t)})^2<\infty$. The common practice is to set $\alpha^{(t)}=\alpha/t^{\beta}$ with $\beta>1$, but the "best" α and β for different problem instances can be quite different. To resolve this issue, adaptive step-size strategies that can automatically determine $\alpha^{(t)}$ are considered in the literature. The RBS method in [40] and the fiber-sampling method in [27] both consider adaptive step-size selection for tensor decomposition. In particular, the latter combines the insight of `adagrad`, which has been popular in deep neural network training, with block-randomized tensor decomposition to come up with an adaptive step-size scheme (see "`adagrad` for Stochastic SLRD").

In Figure 8, we show the MSE on the estimated $\boldsymbol{A}_n$'s obtained by different algorithms after using a certain number of the full MTTKRP (which serves as a unified complexity measure). Here, the tensor has the size $100\times100\times100$, and its CP rank is $R=10$. One can see that stochastic algorithms (BrasCPD and AdaCPD) work remarkably well in this simulation. In particular, the adaptive step-size algorithm exhibits promising performance without tuning the step-size parameters. We also would like to mention that the stochastic algorithms naturally work with incomplete data (e.g., data with missing entries or fibers) since the updates rely only on partial information.

Table 1 presents an incomplete summary of structural constraints/regularization terms (together with the Euclidean data fitting-based CPD cost function) that can be handled by the introduced nonconvex optimization frameworks. One can see that different frameworks may be specialized for various types of structural constraints and regularization terms. In terms of accommodating structural requirements, the AO-ADMM algorithm [22] and the GN framework offered in Tensorlab [44] may be the most flexible ones since they can handle multiple structural constraints simultaneously.

More discussions and conclusion

Exploiting structure at data level

Until now, the focus has been on exploiting the multilinear structure of the decomposition to come up with scalable SLRD algorithms. In many cases, the tensor itself has additional structure that can be exploited to reduce the complexity of some "bottleneck operations," such as MTTKRP (which is used in both GN and BCD) and computing the fitting residual (needed in GN). Note that for batch algorithms, both the computational and the memory complexities of these operations scale as $\mathcal{O}(\text{tensor entries})$. For classic methods, such as BCD, there is

`adagrad` for Stochastic SLRD

In [27], the following term is updated for each block n under the block-randomized fiber-sampling framework:

$$[\boldsymbol{\eta}_n^{(t)}]_{i,r}\leftarrow\frac{1}{\left(b+\sum_{q=1}^{t}[\boldsymbol{G}_{(n)}^{(q)}]_{i,r}^2\right)^{1/2+\epsilon}},$$

where b and ϵ are inconsequential small positive quantities for regularization purposes. Then, the selected block is updated via

$$\boldsymbol{A}_n^{(t+1)}\leftarrow\text{Prox}_{h_n}(\boldsymbol{A}_n^{(t)}-\boldsymbol{\eta}_n^{(t)}\circledast\boldsymbol{G}_n^{(t)}). \tag{S3}$$

This algorithm can be understood as data-adaptive preconditioning for the stochastic oracle $\boldsymbol{G}_n^{(t)}$. Implementing `adagrad`-based stochastic CPD (AdaCPD) is fairly easy, and it often saves a lot of effort for fine-tuning $\alpha^{(t)}$ while attaining a competitive convergence speed; see Figure 8. This also shows the potential of adapting the well-developed stochastic optimization tools in deep neural network training to serve the purpose of SLRD. It is shown in [27] that, using the `adagrad` version of the fiber-sampling algorithm, every limit point of $\{\boldsymbol{\theta}^{(t)}\}$ is a stationary point in expectation if $h(\boldsymbol{\theta})$ is absent. However, convergence in the presence of nonsmooth $h(\boldsymbol{\theta})$ is still an open challenge.

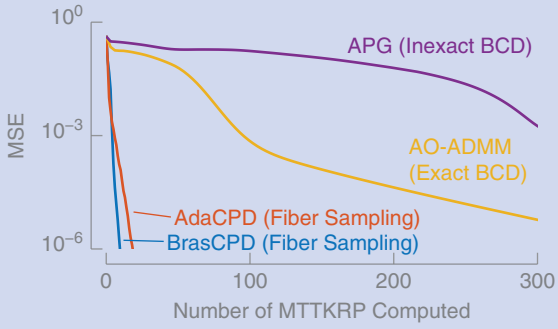


FIGURE 8. Stochastic algorithms [block-randomized stochastic CPD (BrasCPD) (manually fine-tuned step-size) and AdaCPD (adaptive step-size)] use significantly fewer operations to reach a good estimation accuracy for the latent factors, compared to batch algorithms. The MSE for estimating the $\boldsymbol{A}_n$'s against the number of the full MTTKRP used. The CP rank is 10, and $I_n=100$ for all $n=1,2,3$. (Reproduced from [27].) AO-ADMM: alternating optimization-ADMM; APG: alternating proximal gradient.

Table 1. An incomplete summary of structural constraints that can be handled by some representative algorithms.

Structural Constraint or Regularization	AO-ADMM [22] (Exact BCD)	APG [16] (Inexac BCD)	Tensorlab [44] (GN)	AdaCPD [27] (Stochastic)
Nonnegativity ($\boldsymbol{A}_n \geq \boldsymbol{0}$)	✓	✓	✓	✓
Sparsity $\left(\|\boldsymbol{A}_n\|_1 = \sum_{i=1}^{I_n}\sum_{r=1}^{R}|\boldsymbol{A}_n(i,r)|\right)$	✓	✓	✓+	✓
Column group sparsity $\left(\|\boldsymbol{A}_n\|_{2,1} = \sum_{r=1}^{R}\|\boldsymbol{A}_n(:,r)\|_2\right)$	✓	✓	✓+	✓
Row group sparsity $\left(\|\boldsymbol{A}_n^\top\|_{2,1} = \sum_{i=1}^{I_n}\|\boldsymbol{A}_n(i,:)\|_2\right)$	✓	✓	✓+	✓
Total variation ($\|\boldsymbol{T}_1\boldsymbol{A}_n\|_1$)*	✓	✓	✓+	✓
Row probability simplex ($\boldsymbol{A}_n\boldsymbol{1}=\boldsymbol{1},\ \boldsymbol{A}_n \geq \boldsymbol{0}$)	✓	✓	✓	✓
Column probability simplex ($\boldsymbol{1}^\top\boldsymbol{A}_n=\boldsymbol{1}^\top,\ \boldsymbol{A}_n \geq \boldsymbol{0}$)	✓	✓	✓	✓
Tikhonov smoothness ($\|\boldsymbol{T}_2\boldsymbol{A}_n\|_F^2$)*	✓	✓	✓	✓
Decomposition symmetry ($\boldsymbol{A}_n = \boldsymbol{A}_m$)			✓	
Boundedness ($a \leq \boldsymbol{A}_n(i,r) \leq b$)	✓	✓	✓	✓
Coupled factorization (see [42], [45], [46], and [47])	✓	✓	✓	✓
Multiple structures combined (e.g., $\|\boldsymbol{T}_2\boldsymbol{A}_n\|_1 + \|\boldsymbol{A}_n\|_{2,1}$)	✓		✓	

*The operators $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$ are sparse circulant matrices whose expressions can be found in the literature, e.g., [48].
+GN-based methods (except for ProxGN in [39]) work with differentiable functions. In Tensorlab, the ℓ_1 norm-related nondifferentiable terms are handled using function-smoothing techniques as approximations; see details in [35].

rich literature on exploiting the data structure (in particular, sparsity) to avoid memory and flops explosions; see [2] and [5] and the references therein. For all the batch methods, it is crucial to exploit the data structure to reduce the complexity of computing $\boldsymbol{f}$ and $\boldsymbol{g}$ to $O(\text{parameters in representation})$. The key is avoiding the explicit construction of the residual $\mathcal{F}$. The techniques for second-order methods and constraints outlined in the section "Second-Order Approaches" can be used without changes, as the computation of the Gramian as well as the Jacobians $\bar{\boldsymbol{J}}$ resulting from parametric, symmetry, or coupling constraints are independent of the tensor [49], which can be verified from (20). This way, the nonnegative CPD of gigabyte-size tensors, and deterministic blind source separation (BSS) problems with up to millions of samples, can be handled easily on simple laptops and desktops; see [49] for examples.

Other loss functions

In the previous sections, we focused on the standard Euclidean distance to measure the error of the data-fitting term. This is by no means the best choice in all scenarios. It corresponds to the MLE, assuming the input tensor is a low-rank tensor to which additive i.i.d. Gaussian noise is added. It may be crucial in some cases to adopt other data-fitting terms. Let us mention an array of important examples, including the following:

- For count data, such as documents represented as vectors of word counts (this is the so-called bag-of-words model), the matrix/tensor is nonnegative and typically sparse (most documents do not use most words from the dictionary), for which Gaussian noise is clearly not appropriate. Let us focus on the matrix case for simplicity. If we assume the noise added to the entry (i, j) of the input matrix $\boldsymbol{X}$ is a Poissonian of parameter $\lambda = [\boldsymbol{A}_1\boldsymbol{A}_2]_{i,j}$, we have $\Pr(X_{i,j} = k) = e^{-\lambda}\lambda^k/k!$ with $k \in \mathbb{Z}_+$. The MLE leads to minimizing the KL divergence between $\boldsymbol{X}$ and $\boldsymbol{A}_1\boldsymbol{A}_2$:

$$\min_{\boldsymbol{A}_1,\boldsymbol{A}_2} \sum_{i,j} \boldsymbol{X}_{i,j} \log \frac{\boldsymbol{X}_{i,j}}{[\boldsymbol{A}_1\boldsymbol{A}_2^\top]_{i,j}} - \boldsymbol{X}_{i,j} + [\boldsymbol{A}_1\boldsymbol{A}_2^\top]_{i,j}. \tag{43}$$

  The KL divergence is also widely used in imaging because the acquisition can be seen as a photon-counting process (note that, in this case, the input matrix is not necessarily sparse).
- Multiplicative noise, for which each entry of the low-rank tensor is multiplied with some noise, has been shown to be particularly well adapted to audio signals. For example, if the multiplicative noise follows a Gamma distribution, the MLE minimizes the Itakura–Saito (IS) divergence between the observed tensor and its low-rank approximation [50]; in the matrix case with $\boldsymbol{X} \approx \boldsymbol{A}_1\boldsymbol{A}_2^\top$, it is given by

$$\min_{\boldsymbol{A}_1,\boldsymbol{A}_2} \sum_{i,j} \frac{\boldsymbol{X}_{i,j}}{[\boldsymbol{A}_1\boldsymbol{A}_2^\top]_{i,j}} - \log \frac{\boldsymbol{X}_{i,j}}{[\boldsymbol{A}_1\boldsymbol{A}_2^\top]_{i,j}} - 1. \tag{44}$$

- In the presence of outliers (that is, the noise has some entries with a large magnitude), using the component-wise ℓ_1 norm is more appropriate:

$$\sum_{i_1,i_2,\ldots,i_N} \left| \mathcal{T}(i_1,\ldots,i_N) - \sum_{r=1}^{R}\prod_{n=1}^{N} \boldsymbol{A}_n(i_n,r) \right| \tag{45}$$

and corresponds to the MLE for Laplace noise [51]. This is closely related to robust principal component analysis and can be used, for example, to extract the low-rank background from moving objects (treated as outliers) in a video sequence [12]. When "gross outliers" heavily corrupt a number of slabs of the tensor data (or columns/rows of the matrix data), optimization objectives involving nonconvex mixed ℓ_2/ℓ_p functions (where $0 < p \leq 1$)) may also be used [14], [52]. For example, the following fitting cost may be used when one believes that some columns of $\boldsymbol{X}$ are outliers [14]:

$$\sum_{i_2=1}^{I_2} \left\| \boldsymbol{X}(:,i_2) - \boldsymbol{A}_1\boldsymbol{A}_2(i_2,:)^\top \right\|_2^p,$$

where $0 < p \leq 1$ is used to downweight the impact of the outlying columns.

- For quantized signals, that is, signals whose entries have been rounded to some accuracy, an appropriate noise model is the uniform distribution [to be more precise, for the ℓ_∞ norm to correspond to the MLE, all entries must be rounded with the same absolute accuracy (e.g., the nearest integer), which is typically not the case in most programming languages]. For example, if each entry of a low-rank matrix is rounded to the nearest integer, then each entry of the noise can be modeled with the uniform distribution in the interval $[-0.5, 0.5]$. The corresponding MLE minimizes the component-wise ℓ_∞ norm, replacing $\Sigma_{i_1,i_2,\ldots,i_N}$ by $\max_{i_1,i_2,\ldots,i_N}$ in (45).
- If the noise is not identically distributed among the entries of the tensor, a weight should be assigned to each entry. For example, for independently distributed Gaussian noise, the MLE minimizes

$$\sum_{i_1=1}^{I_1} \cdots \sum_{i_N=1}^{I_N} \frac{\left(\mathcal{T}(i_1,\ldots,i_N) - \sum_{r=1}^{R}\prod_{n=1}^{N} \boldsymbol{A}_n(i_n,r)\right)^2}{\sigma^2(i_1,\ldots,i_N)},$$

where $\sigma^2(i_1,\ldots,i_N)$ is the variance of the noise for the entry at position $(i_1,\ldots,i_N)$. Interestingly, for missing entries, $\sigma(i_1,\ldots,i_N) = +\infty$ corresponds to a weight of zero while, if there is no noise [that is, $\sigma(i_1,\ldots,i_N) = 0$], the weight is infinite so that the entry must be exactly reconstructed.

In all these cases, we end up with more complicated optimization problems because the nice properties of the Euclidean distance are lost, in particular, the Lipschitz continuity of the gradient (the ℓ_1 and ℓ_∞ norms are even nonsmooth). For the weighted norm, the problem might become ill-posed (the optimal solution might not exist, even with nonnegativy constraints) in the presence of missing entries because some weights are zero so that the weighted "norm" is actually not a norm. For the KL and IS divergences, the gradient of the objective is not Lipschitz continuous, and the objective is not defined everywhere: $\boldsymbol{X}_{i,j} > 0$ requires $[\boldsymbol{A}_1\boldsymbol{A}_2^\top]_{i,j} > 0$ in (43) and (44).

The most popular optimization method for these divergences is multiplicative updates, which constitutes an inexact BCD approach; see the "Inexact BCD" section. For the component-wise ℓ_1, ℓ_∞ norms and nonconvex ℓ_2/ℓ_p functions, subgradient descent (which is similar to PGD), iteratively reweighed LS, and exact BCD are popular approaches; see, e.g., [14], [51], and [52]. Some of these objectives (e.g., the KL divergence and the component-wise ℓ_1 norm) can also be handled under a variant of the AO-ADMM framework with simple updates but possibly high memory complexities [22]. In all cases, convergence will be typically slower than for the Euclidean distance.

Tractable SLRD problems and algorithms

We have introduced a series of nonconvex optimization tools for SLRD that are all supported by stationary-point convergence guarantees. However, it is, in general, unknown if these algorithms will reach a globally optimal solution (or if the LRDMs can be exactly found). While convergence to the global optimum can be observed in practical applications, establishing pertinent theoretical guarantees is challenging given the NP-hardness of the problem [11]–[13], [53]. Nevertheless, in certain settings, the computation of LRDMs is known to be tractable. We mention the following:

- In the case where a fully symmetric tensor admits a CPD with all latent factors identical and orthogonal (i.e., all the $\boldsymbol{A}_n$s are identical, and $\boldsymbol{A}_n^\top\boldsymbol{A}_n = \boldsymbol{I}$), the latent factors can be computed using a power iteration/deflation-type algorithm [53]. This is analogous to the computation of the eigendecomposition of a symmetric matrix through successive power iteration and deflation. One difference is that a symmetric matrix can be exactly diagonalized by an orthogonal eigentransformation, while a generic higher-order tensor can only be approximately diagonalized; the degree of diagonalizability affects the convergence [54]. By itself, CPD with identical and orthogonal $\boldsymbol{A}_n$s is a special model that is not readily encountered in many applications. However, in an array of BSS and machine learning problems (e.g., independent component analysis, topic modeling, and community detection), it is possible under some conditions to transform higher-order statistics so that they satisfy this special model up to estimation errors. In particular, second-order statistics can be used for a prewhitening that is guaranteed to orthogonalize the latent factors when the decomposition is exact. For deflation-based techniques that do not require orthogonality or symmetry, see [55] and [56].
- Beyond CPD with identical and orthogonal latent factors, eigendecomposition-based algorithms have a long history for finding the exact CPD under various conditions. The simplest scenario is where two factor matrices have a full column rank and the third factor matrix does not have proportional columns. In this scenario, the exact CPD can be found from the generalized eigenvalue decomposition of a pencil formed by two tensor slices (or linear combinations of slices) [57]. The fact that, in the first steps of the algorithm, the tensor is reduced to just a pair of its slices implies some bounds on the accuracy, especially in cases where the rank is high compared to the tensor dimensions, i.e., when a lot of information is extracted from the two slices [58]. To mitigate this, [56] presents an algebraic approach in which multiple pencils are each partially used in a way that takes into account their numerical properties.

Moreover, the working conditions of the basic eigendecomposition approach have been relaxed to situations in which only one factor is required to have a full column rank [59]. The method utilizes a bilinear mapping to convert the more general CPD problem to the "simplest scenario" in the preceding. This line of work has been further extended to handle cases where the latent factors are all allowed to be rank deficient, enabling exact algebraic computation up to the famous Kruskal bound and beyond [60], [61]. Algorithms of this type have been proposed for other tensor decomposition models, as well, e.g., BTD and LL1 decomposition [19], [62], coupled CPD [63], and CPD of incomplete fiber-sampled tensors [43]. While the accuracy of these methods is sometimes limited in practical noisy settings, the computed results often provide good initialization points for the introduced iterative, nonconvex, optimization-based methods.

- In [64] noise bounds are derived under which the CPD minimization problem is well posed and the cost function has only one local minimum, which is hence global.
- Many unconstrained low-rank matrix estimation problems (e.g., compressed matrix recovery and matrix completion) are known to be solvable via nonconvex optimization methods under certain conditions [65]. Structure-constrained matrix decomposition problems are, in general, more challenging, but solvable cases also exist under some model assumptions. For example, separable NMF tackles the NMF problem through the assumption that a latent factor contains a column-scaled version of the identity matrix as its submatrix. This assumption facilitates a number of algorithms that provably output the target latent factors, even in the noisy cases; see tutorials in [1] and [4]. Solvable cases also exist in dictionary learning that identifies a sparse factor in an "overcomplete" basis. If the sparse latent factor is generated following a Gaussian-Bernoulli model, it has been shown that the optimization landscape under an "inverse-filtering" formulation is "benign"; i.e., all local minima are also global minima. Consequently, a globally optimal solution can be attained via nonconvex optimization methods [66].

Other models

The algorithm design principles can be generalized to cover other models, e.g., BTD, LL1, Tucker, and Tensor Train (TT)/hierarchical Tucker (hT), to name a few [67]–[72]. Note that, in their basic form, BTD, LL1, Tucker, and TT/hT involve subspaces rather than vectors so that optimization on manifolds is a natural framework. Some extensions of SLRD are straightforward. For instance, both BCD and second-order algorithms for structured Tucker, BTD, and LL1 decompositions exist [16], [19], [31], [36]. GN-based methods were also considered for nonnegativity-constrained Tucker decomposition. LL1 can be regarded as CPD with repeated columns in some latent factor matrices, and thus the parametrization techniques can be used to come up with GN algorithms for LL1 as constrained CPD [31], [35]. However, some extensions may require more effort. For example, in stochastic algorithm design, different tensor models and structural constraints may require the custom design of sampling strategies, as we have seen in the CPD case. This also entails many research opportunities ahead.

Concluding remarks

In this article, we introduced three types of nonconvex optimization tools that are effective for SLRD. Several remarks are in order.

- The BCD-based approaches are easy to understand and implement. Inexact BCD and extrapolation techniques are particularly useful in practice. This line of work can potentially handle a large variety of constraints and regularization terms if the subproblem solver is properly chosen. The downside is that BCD is a first-order optimization approach at a high level. Hence, the speed of convergence (in terms of the number of iterations needed) is usually not fast. Designing effective and lightweight acceleration strategies may help advance BCD-based SLRD algorithms.
- The GN-based approaches are powerful in terms of their convergence speed and per-iteration computational complexity. They are also the foundation of the tensor computation infrastructure Tensorlab. On the other hand, the GN approaches are specialized for NLS and smoothed objective functions. In other words, they may not be as flexible as BCD-based approaches in terms of incorporating structural information. Using ProxGN may improve the flexibility, but the subproblems arising in the ProxGN framework are not necessarily easy to solve. Extending the second-order approaches to accommodate more structural requirements and objective functions other than the LS loss promises a fertile research ground.
- The stochastic approaches strike a balance between per-iteration computational/memory complexity and the overall decomposition algorithm effectiveness. Different sampling strategies may be able to handle various types of structural information. Stochastic optimization may involve more hyperparameters to tune (in particular, the minibatch size and the step-size) and thus may require more attentive software engineering for implementation. Convergence properties of stochastic tensor/matrix decomposition algorithms are not as clear, which also poses many exciting research questions for the tensor/matrix and optimization communities to explore.

Acknowledgments

Xiao Fu is supported by the National Science Foundation, under projects ECCS-1608961, ECCS-1808159, and III-1910118, and the Army Research Office, under projects ARO W911NF-19-1-0247 and ARO W911NF-19-1-0407. Nico Vervliet is supported by a junior postdoctoral fellowship (12ZM220N) from the Research Foundation–Flanders. The work of the Belgian team is also supported by the Fonds de la Recherche Scientifique–Fonds National de la Recherche Scientifique and the Fonds Wetenschappelijk Onderzoek–Vlaanderen, under Excellence Of Science project 30468160 (SeLMA); KU Leuven Internal Funds

C16/15/059 and ID-N project 3E190402; and the Flemish Government (Artificial Intelligence Research Program). Nicolas Gillis acknowledges the support of the European Research Council (starting grant 679515).

Authors

Xiao Fu (xiao.fu@oregonstate.edu) received his Ph.D. degree in electronic engineering from the Chinese University of Hong Kong in 2014. He is an assistant professor in the School of Electrical Engineering and Computer Sciences at Oregon State University, Corvallis. He was a postdoctoral associate at the University of Minnesota from 2014 to 2017. His research interests include the theory and methods of matrix and tensor factorization and their applications in signal processing and machine learning. He received a Best Student Paper Award at ICASSP 2014 and the Outstanding Postdoctoral Scholar Award at the University of Minnesota in 2016.

Nico Vervliet (nico.vervliet@esat.kuleuven.be) received his Ph.D. degree from the Faculty of Engineering, KU Leuven, Belgium, in 2018. He is currently a postdoctoral researcher at the STADIUS Center for Dynamical Systems, Signal Processing, and Data Analytics, Department of Electrical Engineering, KU Leuven. His research interests include numerical multilinear algebra, the optimization and application of tensor decompositions in signal processing, and data analysis, with a focus on big data and data fusion. He is the lead developer of Tensorlab.

Lieven De Lathauwer (lieven.delathauwer@kuleuven.be) received his Ph.D. degree from the Faculty of Engineering, KU Leuven, Belgium, in 1997, where he is currently a full professor. From 2000 to 2007, he was a research associate at the CNRS-ETIS. He is an associate editor of *SIAM Journal on Matrix Analysis and Applications* and has served as an associate editor of *IEEE Transactions on Signal Processing*. He was a corecipient of the IEEE Signal Processing Society Signal Processing Magazine best paper award in 2018. His research concerns the development of tensor tools for engineering applications. He is a Fellow of IEEE, the Society for Industrial and Applied Mathematics, and European Association for Signal Processing.

Kejun Huang (kejun.huang@ufl.edu) received his B.Eng. degree in communications engineering from the Nanjing University of Information Science and Technology, China, in 2010 and his Ph.D. degree in electrical engineering from the University of Minnesota, Minneapolis, in 2016. He is an assistant professor in the Department of Computer and Information Science and Engineering at the University of Florida, Gainesville. He was a postdoctoral associate in the Department of Electrical and Computer Engineering at the University of Minnesota from 2016 to 2018. His research interests include machine learning, signal processing, optimization, and statistics. He is a Member of IEEE.

Nicolas Gillis (nicolas.gillis@umons.ac.be) received his M.S. and Ph.D. degrees in applied mathematics from UCLouvain, Belgium, in 2007 and 2011, respectively. He is currently an associate professor in the Department of Mathematics and Operational Research, University of Mons, Belgium. His research interests include optimization, numerical linear algebra, signal processing, machine learning, and data mining. He received the Householder Award in 2014 and a European Research Council starting grant in 2015. He currently serves as an associate editor of *IEEE Transactions on Signal Processing* and *SIAM Journal on Matrix Analysis and Applications*.

References

[1] N. Gillis, "The why and how of nonnegative matrix factorization," in *Regularization, Optimization, Kernels, and Support Vector Machines* (Machine Learning and Pattern Recognition), J. A. K. Suykens, M. Signoretto, and A. Argyriou, Eds. Boca Raton, FL: Chapman & Hall/CRC, 2014, ch. 12, pp. 257–291.

[2] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Trans. Signal Process.*, vol. 65, no. 13, pp. 3551–3582, 2017. doi: 10.1109/TSP.2017.2690524.

[3] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H.-A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Process. Mag.*, vol. 32, no. 2, pp. 145–163, 2015. doi: 10.1109/MSP.2013.2297439.

[4] X. Fu, K. Huang, N. D. Sidiropoulos, and W.-K. Ma, "Nonnegative matrix factorization for signal and data analytics: Identifiability, algorithms, and applications," *IEEE Signal Process. Mag.*, vol. 36, no. 2, pp. 59–80, Mar. 2019. doi: 10.1109/MSP.2018.2877582.

[5] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, 2009. doi: 10.1137/07070111X.

[6] L.-H. Lim and P. Comon, "Nonnegative approximations of nonnegative tensors," *J. Chemometr.*, vol. 23, no. 7–8, pp. 432–441, 2009. doi: 10.1002/cem.1244.

[7] Y. Qian, F. Xiong, S. Zeng, J. Zhou, and Y. Y. Tang, "Matrix-vector nonnegative tensor factorization for blind unmixing of hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 3, pp. 1776–1792, 2017. doi: 10.1109/TGRS.2016.2633279.

[8] K. Huang and X. Fu, "Detecting overlapping and correlated communities without pure nodes: Identifiability and algorithm," in *Proc. 36th Int. Conf. Machine Learning*, Long Beach, CA, June 9–15, 2019, vol. 97, pp. 2859–2868.

[9] G. Zhou, A. Cichocki, Q. Zhao, and S. Xie, "Nonnegative matrix and tensor factorizations: An algorithmic perspective," *IEEE Signal Process. Mag.*, vol. 31, no. 3, pp. 54–65, 2014. doi: 10.1109/MSP.2014.2298891.

[10] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an 'explanatory' multimodal factor analysis," *UCLA Work. Papers Phonetics*, vol. 16, pp. 1–84, 1970.

[11] C. J. Hillar and L.-H. Lim, "Most tensor problems are NP-hard," *J. ACM*, vol. 60, no. 6, pp. 45:1–45:39, 2013. doi: 10.1145/2512329.

[12] N. Gillis and S. A. Vavasis, "On the complexity of robust PCA and 1-norm low-rank matrix approximation," *Math. Oper. Res.*, vol. 43, no. 4, pp. 1072–1084, 2018. doi: 10.1287/moor.2017.0895.

[13] S. A. Vavasis, "On the complexity of nonnegative matrix factorization," *SIAM J. Optim.*, vol. 20, no. 3, pp. 1364–1377, 2009. doi: 10.1137/070709967.

[14] X. Fu, K. Huang, B. Yang, W. Ma, and N. D. Sidiropoulos, "Robust volume minimization-based matrix factorization for remote sensing and document clustering," *IEEE Trans. Signal Process.*, vol. 64, no. 23, pp. 6254–6268, Dec. 2016. doi: 10.1109/TSP.2016.2602800.

[15] A. Beutel, P. P. Talukdar, A. Kumar, C. Faloutsos, E. E. Papalexakis, and E. P. Xing, "Flexifact: Scalable flexible factorization of coupled tensors on Hadoop," in *Proc. SIAM Int. Conf. Data Mining (SDM 2014)*, 2014, pp. 109–117.

[16] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM J. Imaging Sci.*, vol. 6, no. 3, pp. 1758–1789, 2013. doi: 10.1137/120887795.

[17] Y. Xu and W. Yin, "Block stochastic gradient iteration for convex and nonconvex optimization," *SIAM J. Optim.*, vol. 25, no. 3, pp. 1686–1716, 2015.

[18] 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. doi: 10.1137/120891009.

[19] L. De Lathauwer and D. Nion, "Decompositions of a higher-order tensor in block terms—Part III: Alternating least squares algorithms," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1067–1083, 2008. doi: 10.1137/070690730.

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

[21] C.-J. Lin, "Projected gradient methods for nonnegative matrix factorization," *Neural Comput.*, vol. 19, no. 10, pp. 2756–2779, 2007. doi: 10.1162/neco.2007.19.10.2756.

[22] K. Huang, N. D. Sidiropoulos, and A. P. Liavas, "A flexible and efficient algorithmic framework for constrained matrix and tensor factorization," *IEEE Trans. Signal Process.*, vol. 64, no. 19, pp. 5052–5065, 2016. doi: 10.1109/TSP.2016.2576427.

[23] N. Guan, D. Tao, Z. Luo, and B. Yuan, "NeNMF: An optimal gradient method for nonnegative matrix factorization," *IEEE Trans. Signal Process.*, vol. 60, no. 6, pp. 2882–2898, 2012. doi: 10.1109/TSP.2012.2190406.

[24] 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. doi: 10.1137/07069239X.

[25] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Proc. Advances Neural Information Processing Systems*, 2001, vol. 13, pp. 556–562.

[26] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$," *Doklady an USSR*, vol. 269, pp. 543–547, 1983.

[27] X. Fu, S. Ibrahim, H.-T. Wai, C. Gao, and K. Huang, "Block-randomized stochastic proximal gradient for low-rank tensor factorization," *IEEE Trans. Signal Process.*, vol. 68, pp. 2170–2185, Mar. 20, 2020. doi: 10.1109/TSP.2020.2982321.

[28] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, Nashua, NH, 1999.

[29] C.-J. Hsieh and I. S. Dhillon, "Fast coordinate descent methods with variable selection for non-negative matrix factorization," in *Proc. 17th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2011, pp. 1064–1072. doi: 10.1145/2020408.2020577.

[30] Z. Li, A. Uschmajew, and S. Zhang, "On convergence of the maximum block improvement method," *SIAM J. Optim.*, vol. 25, no. 1, pp. 210–233, 2015. doi: 10.1137/130939110.

[31] L. Sorber, M. Van Barel, and L. De Lathauwer, "Optimization-based algorithms for tensor decompositions: Canonical polyadic decomposition, decomposition in rank-(L_r, L_r, 1) terms, and a new generalization," *SIAM J. Optim.*, vol. 23, no. 2, pp. 695–720, 2013. doi: 10.1137/120868323.

[32] A.-H. Phan, P. Tichavský, and A. Cichocki, "Low complexity damped Gauss–Newton algorithms for CANDECOMP/PARAFAC," *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 1, pp. 126–147, 2013. doi: 10.1137/100808034.

[33] L. Sorber, I. Domanov, M. Van Barel, and L. De Lathauwer, "Exact line and plane search for tensor optimization," *Comput. Optim. Appl.*, vol. 63, no. 1, pp. 121–142, 2015. doi: 10.1007/s10589-015-9761-5.

[34] A.-H. Phan, P. Tichavský, 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. doi: 10.1109/TSP.2013.2269903.

[35] N. Vervliet and L. D. Lathauwer, "Numerical optimization based algorithms for data fusion," in *Data Fusion Methodology and Applications* (Data Handling in Science and Technology), vol. 31, 1st ed., M. Cocchi, Ed. New York: Elsevier, 2019, ch. 4, pp. 81–128.

[36] 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. doi: 10.1109/JSTSP.2015.2400415.

[37] Q. Wei, N. Dobigeon, and J.-Y. Tourneret, "Fast fusion of multi-band images based on solving a Sylvester equation," *IEEE Trans. Image Process.*, vol. 24, no. 11, pp. 4109–4121, 2015. doi: 10.1109/TIP.2015.2458572.

[38] G. Zhang, X. Fu, J. Wang, X.-L. Zhao, and M. Hong, "Spectrum cartography via coupled block-term tensor decomposition," *IEEE Trans. Signal Process.*, vol. 68, pp. 3660–3675, May 14, 2020. doi: 10.1109/TSP.2020.2993530.

[39] K. Huang and X. Fu, "Low-complexity proximal Gauss–Newton algorithm for nonnegative matrix factorization," in *Proc. IEEE Global Conf. Signal and Information Processing (GlobalSIP 2019)*, 2019, pp. 1–5. doi: 10.1109/GlobalSIP45357.2019.8969492.

[40] N. Vervliet and L. D. Lathauwer, "A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors," *IEEE J. Sel. Topics Signal Process.*, vol. 10, no. 2, pp. 284–295, 2016. doi: 10.1109/JSTSP.2015.2503260.

[41] C. Battaglino, G. Ballard, and T. G. Kolda, "A practical randomized CP tensor decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 39, no. 2, pp. 876–901, 2018. doi: 10.1137/17M1112303.

[42] C. I. Kanatsoulis, X. Fu, N. D. Sidiropoulos, and M. Akcakaya, "Tensor completion from regular sub-Nyquist samples," *IEEE Trans. Signal Process.*, vol. 68, pp. 1–16, Nov. 6, 2019. doi: 10.1109/TSP.2019.2952044.

[43] M. Sørensen and L. De Lathauwer, "Fiber sampling approach to canonical polyadic decomposition and application to tensor completion," *SIAM J. Matrix Anal. Appl.*, vol. 40, no. 3, pp. 888–917, 2019. doi: 10.1137/17M1140790.

[44] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer. *Tensorlab 3.0* (2016). [Online]. Available: https://www.tensorlab.net

[45] S. Ibrahim, X. Fu, N. Kargas, and K. Huang, "Crowdsourcing via pairwise co-occurrences: Identifiability and algorithms," in *Proc. Advances Neural Information Processing Systems*, 2019, vol. 32, pp. 7845–7855.

[46] S. Ibrahim and X. Fu, "Stochastic optimization for coupled tensor decomposition with applications in statistical learning," in *Proc. IEEE DSW*, 2019, pp. 300–304. doi: 10.1109/DSW.2019.8755797.

[47] M. Sørensen and L. De Lathauwer, "Coupled canonical polyadic decompositions and (coupled) decompositions in multilinear rank-($L_{r,n}$, $L_{r,n}$, 1) terms—Part I: Uniqueness," *SIAM J. Matrix Anal. Appl.*, vol. 36, no. 2, pp. 496–522, 2015.

[48] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[49] N. Vervliet, O. Debals, and L. De Lathauwer, "Exploiting efficient representations in tensor decompositions," *SIAM J. Sci. Comput.*, vol. 41, no. 2, pp. A789–A815, 2019. doi: 10.1137/17M1152371.

[50] C. Févotte, N. Bertin, and J.-L. Durrieu, "Nonnegative matrix factorization with the Itakura–Saito divergence: With application to music analysis," *Neural Comput.*, vol. 21, no. 3, pp. 793–830, 2009. doi: 10.1162/neco.2008.04-08-771.

[51] S. A. Vorobyov, Y. Rong, N. D. Sidiropoulos, and A. B. Gershman, "Robust iterative fitting of multilinear models," *IEEE Trans. Signal Process.*, vol. 53, no. 8, pp. 2678–2689, 2005. doi: 10.1109/TSP.2005.850343.

[52] X. Fu, K. Huang, W.-K. Ma, N. 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. doi: 10.1109/TSP.2015.2469642.

[53] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky, "Tensor decompositions for learning latent variable models," *J. Mach. Learn. Res.*, vol. 15, no. 80, pp. 2773–2832, 2014.

[54] M. Espig, W. Hackbusch, and A. Khachatryan, On the convergence of alternating least squares optimisation in tensor format representations. 2015. [Online]. Available: arXiv:1506.00062

[55] A.-H. Phan, P. Tichavsky`, and A. Cichocki, "Tensor deflation for CANDECOMP/PARAFAC-Part I: Alternating subspace update algorithm," *IEEE Trans. Signal Process.*, vol. 63, no. 22, pp. 5924–5938, 2015. doi: 10.1109/TSP.2015.2458785.

[56] E. Evert, M. Vandecappelle, and L. De Lathauwer, "The generalized eigenspace decomposition," ESAT-STADIUS, KU Leuven, Leuven, Belgium, Tech. Rep. 20-80, 2020.

[57] S. E. Leurgans, R. T. Ross, and R. B. Abel, "A decomposition for three-way arrays," *SIAM J. Matrix Anal. Appl.*, vol. 14, no. 4, pp. 1064–1083, 1993. doi: 10.1137/0614071.

[58] C. Beltrán Álvarez, P. Breiding, and N. Vannieuwenhoven, "Pencil-based algorithms for tensor rank decomposition are not stable," *SIAM J. Matrix Anal. Appl.*, vol. 40, no. 2, pp. 739–773, 2019. doi: 10.1137/18M1200531.

[59] L. De Lathauwer, "A link between the canonical decomposition in multilinear algebra and simultaneous matrix diagonalization," *SIAM J. Matrix Anal. Appl.*, vol. 28, no. 3, pp. 642–666, 2006. doi: 10.1137/040608830.

[60] I. Domanov and L. De Lathauwer, "Canonical polyadic decomposition of third-order tensors: Reduction to generalized eigenvalue decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 35, no. 2, pp. 636–660, 2014. doi: 10.1137/130916084.

[61] I. Domanov and L. De Lathauwer, "Canonical polyadic decomposition of third-order tensors: Relaxed uniqueness conditions and algebraic algorithm," *Linear Algebra Appl.*, vol. 513, pp. 342–375, Jan. 2017. doi: 10.1016/j.laa.2016.10.019.

[62] I. Domanov and L. D. Lathauwer, "On uniqueness and computation of the decomposition of a tensor into multilinear rank-($1,L_r, L_r$) terms," *SIAM J. Matrix Anal. Appl.*, vol. 41, no. 2, pp. 747–803, 2020. doi: 10.1137/18M1206849.

[63] M. Sørensen, I. Domanov, and L. De Lathauwer, "Coupled canonical polyadic decompositions and (coupled) decompositions in multilinear rank-($L_{r,n}$, $L_{r,n}$ 1) terms—Part II: Algorithms," *SIAM J. Matrix Anal. Appl.*, vol. 36, no. 3, pp. 1015–1045, 2015. doi: 10.1137/140956865.

[64] E. Evert and L. De Lathauwer, "Perturbation theory for CPD and for joint generalized eigenvalues," ESAT-STADIUS, KU Leuven, Leuven, Belgium, Tech. Rep. 19-71, 2019.

[65] Y. Chi, Y. M. Lu, and Y. Chen, "Nonconvex optimization meets low-rank matrix factorization: An overview," *IEEE Trans. Signal Process.*, vol. 67, no. 20, pp. 5239–5269, 2019. doi: 10.1109/TSP.2019.2937282.

[66] J. Sun, Q. Qu, and J. Wright, "Complete dictionary recovery over the sphere I: Overview and the geometric picture," *IEEE Trans. Inf. Theory*, vol. 63, no. 2, pp. 853–884, Feb. 2017. doi: 10.1109/TIT.2016.2632162.

[67] I. V. Oseledets, D. Savostianov, and E. E. Tyrtyshnikov, "Tucker dimensionality reduction of three-dimensional arrays in linear time," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 939–956, 2008. doi: 10.1137/060655894.

[68] B. Savas and L.-H. Lim, "Quasi-Newton methods on Grassmannians and multilinear approximations of tensors," *SIAM J. Sci. Comput.*, vol. 32, no. 6, pp. 3352–3393, 2010. doi: 10.1137/090763172.

[69] M. Ishteva, P.-A. Absil, S. Van Huffel, and L. De Lathauwer, "Best low multilinear rank approximation of higher-order tensors, based on the Riemannian trust-region scheme," *SIAM J. Matrix Anal. Appl.*, vol. 32, no. 1, pp. 115–135, 2011. doi: 10.1137/090764827.

[70] W. Hackbusch, *Tensor Spaces and Numerical Tensor Calculus*, 2nd ed. (Springer Series in Computational Mathematics 56). Berlin: Springer-Verlag, 2012.

[71] L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques," *GAMM-Mitteilungen*, vol. 36, no. 1, pp. 53–78, 2013. doi: 10.1002/gamm.201310004.

[72] B. N. Khoromskij, *Tensor Numerical Methods in Scientific Computing*, vol. 19. Berlin: Walter de Gruyter GmbH & Co. KG, 2018.

[73] A. M. S. Ang, J. E. Cohen, N. Gillis, and L. T. K. Hien, Accelerating block coordinate descent for nonnegative tensor factorization. 2020. arXiv:2001.04321.

SP