AN ABSTRACT OF THE THESIS OF

Cheng Gao for the degree of Master of Science in Electrical and Computer Engineering presented on May 31, 2019.

Title: Large Scale Tensor Decomposition Algorithms Using Block-randomized Stochastic Proximal Gradient

Abstract approved: _____

Xiao Fu

We consider the problem of computing the cannonical polyadic decomposition (CPD) for large-scale dense tensors. This work is a combination of alternating least squares and fiber sampling. Data sparsity can be leveraged to handle large tensor CPD, but this route is not feasible for dense data. Inspired by stochastic optimization's low memory consuming and per iteration complexity, we propose a stochastic optimization framework for CPD which combines the insights of block coordinate descent and stochastic proximal gradient. At the same time, we also provide an analysis for the convergence property which are unclear to many state-of-art. In addition, our algorithm can handle many frequently used regularizers and constraints, and thus is more flexible compared to many existing algorithms. Applications to hyperspectral images are considered in the experiment analysis part.

Large Scale Tensor Decomposition Algorithms Using Block-randomized Stochastic Proximal Gradient

by

Cheng Gao

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented May 31, 2019
Commencement June 2019

Master of Science thesis of Cheng Gao presented on May 31, 2019.

APPROVED:

Major Professor, representing Electrical and Computer Engineering

---

Head of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Cheng Gao, Author

ACKNOWLEDGEMENTS

I would like to acknowledge my major advisor Dr. Xiao Fu and my family for their support.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

Chapter 1 Introduction

Canonical polyadic decomposition (CPD) [previously known as parallel factor analysis (PARAFAC)] [11, 29, 44] is arguably the most popular low-rank tensor decomposition model. CPD has successfully found many applications in various fields, such as analytical chemistry [38], social network mining [48], hyperspectral imaging [26], topic modeling [2], and time series analysis [3]; also see [21, 43, 45] for more classic applications in communications.

Computing the CPD of a tensor, however, is a quite challenging optimization problem [24]. Many algorithms have been proposed through the years [11, 25, 32, 52]. To keep pace with the ever growing volume of available data, one pressing challenge is to compute CPD at scale. The classic alternating least squares (ALS) algorithm [11] has an elegant algorithmic structure, but also suffers from a number of numerical issues [13, 33] and is hardly scalable. In recent years, many new CPD algorithms have appeared, triggered by the advances in big data analytics and first-order optimization [25, 27, 35, 52]. Many of these algorithms leverage data sparsity to scale up CPD—by cleverly using the zero elements in huge tensors, computationally costly key operations in ALS (e.g., the *matricized tensor times Khatri-Rao product* (MTTKRP) operation) can be significantly simplified. Consequently, the classic ALS algorithm can be modified to handle CPD of huge and sparse tensors.

However, when the tensor to be factored is *dense*—i.e., when most entries of the tensor are nonzero—the sparsity-enabled efficient algorithms [25–27, 35, 52] are no longer applicable. Note that large and dense tensors arise in many timely and important applications such as medical imaging [1], hyperspectral imaging [26], and computer vision [42]. In fact, since big dense tensors typically cost a lot of memory (e.g., a dense tensor with a size of $2,000 \times 2,000 \times 2,000$ occupies 57.52GB memory if saved as double-precision numbers), it is even hard to load them into the RAM of laptops, desktops, or servers. This also raises serious challenges in the era of Internet of Things (IoT)—where edge computing on small devices is usually preferable.

Stochastic approximation is a powerful tool for handling optimization problems in-

volving dense data, which is known for its low per-iteration memory and computational complexities [8]. A number of stochastic optimization based CPD algorithms have been proposed in the literature [4, 7, 49]. Specifically, The works in [7, 49] work in an iterative manner. In each iteration, the algorithm samples a random subset of the tensor entries and update the corresponding parts of the latent factors using the sampled data. The algorithms have proven quite effective in practice, and features distributed implementation [7]. The challenge here is that every tensor entry only contains information of a certain *row* of the latent factors, and updating the entire latent factors may need a lot of iterations. This may lead to slow improvement of the latent factor estimation accuracy. More importantly, this update strategy loses the opportunity to incorporate constraints/regularizations on the whole latent factors, since the sampled entries only contain partial information of them. This is undesired in practice, since prior information on the latent factors are critical for enhancing performance, especially in noisy cases.

Recently, a stochastic algorithm that ensures updating one entire latent factor in every iteration was proposed in [4]. Instead of sampling tensor entries, the algorithm works via sampling *tensor fibers* that contain information of the whole latent factors. However, this algorithm works with at least as many fibers as the tensor rank, which in some cases gives rise to much higher per-iteration complexity relative to the algorithms in [7,49]. In addition, like those in [7,49], the algorithm in [4] cannot handle constraints or regularizations on the latent factors, either. Furthermore, although empirically working well, convergence properties of many stochastic CPD algorithms such as those in [4, 49] are unclear.

Contributions In this thesis, we propose a new stochastic algorithmic framework for computing the CPD of large-scale dense tensors. Specifically, our contributions include:

• **A Doubly Randomized Computational Framework for Large-Scale CPD**. Our first contribution lies in proposing an efficient and flexible computational framework for CPD of large dense tensors. Our method is a judicious combination of randomized block coordinate descent (BCD) [5, 34] and stochastic proximal gradient (SPG) [22, 23]. Specifically, in each iteration, our method first samples a mode from all modes of the tensor. Then, the algorithm samples some fibers of this mode and updates the corresponding latent factor via stochastic proximal operations. Such a combination exhibits an array of attractive features: It admits much smaller per-iteration memory and computational complexities relative to the existing fiber sampling based method

in [4]. More importantly, it is very flexible in terms of incorporating regualrizations and constraints on the latent factors.

• **Rigorous Convergence Analysis**. Both BCD and SPG are well studied topics in the optimization literature [5, 34, 39]. However, convergence properties of the proposed framework is not immediately clear, due to the nonconvex nature of CPD. The existing block-randomized SGD (BR-SGD) framework [50] only considers convex optimization. A related work in [53] deals with nonconvex problems via block stochastic gradient, but their *Gauss-Seidel* type BCD strategy (without block randomization) makes the convergence analysis inapplicable to our case. Hence, we offer tailored convergence analyses for our proposed CPD algorithms.

• **Implementation-friendly Adaptive Stepsize Scheduling**. In practice, one of the most challenging aspects in stochastic optimization is selecting a proper stepsize schedule. To make the proposed algorithms friendly to use by practitioners, we propose a practical and adaptive stepsize schedule that is based on the celebrated `Adagrad` algorithm [16]. `Adagrad` is an adaptive stepsize selection method that was devised for single-block gradient descent. Nonetheless, we find through extensive simulations that it largely helps reduce the agonizing pain of tuning stepsize when implementing our multi-block algorithm for CPD. In addition, we also show that the adaptive stepsize-based algorithm converges to a stationary problem almost surely under some conditions.

A quick demonstration of the effectiveness of the proposed algorithms is shown in Fig. 1.1, where the average mean squared error (MSE) of the estimated latent factors against the number of MTTKRP computed (which dominates the complexity) is plotted. One can see that the proposed algorithm largely outperforms a couple of state-of–the-art algorithms for constrained CPD.

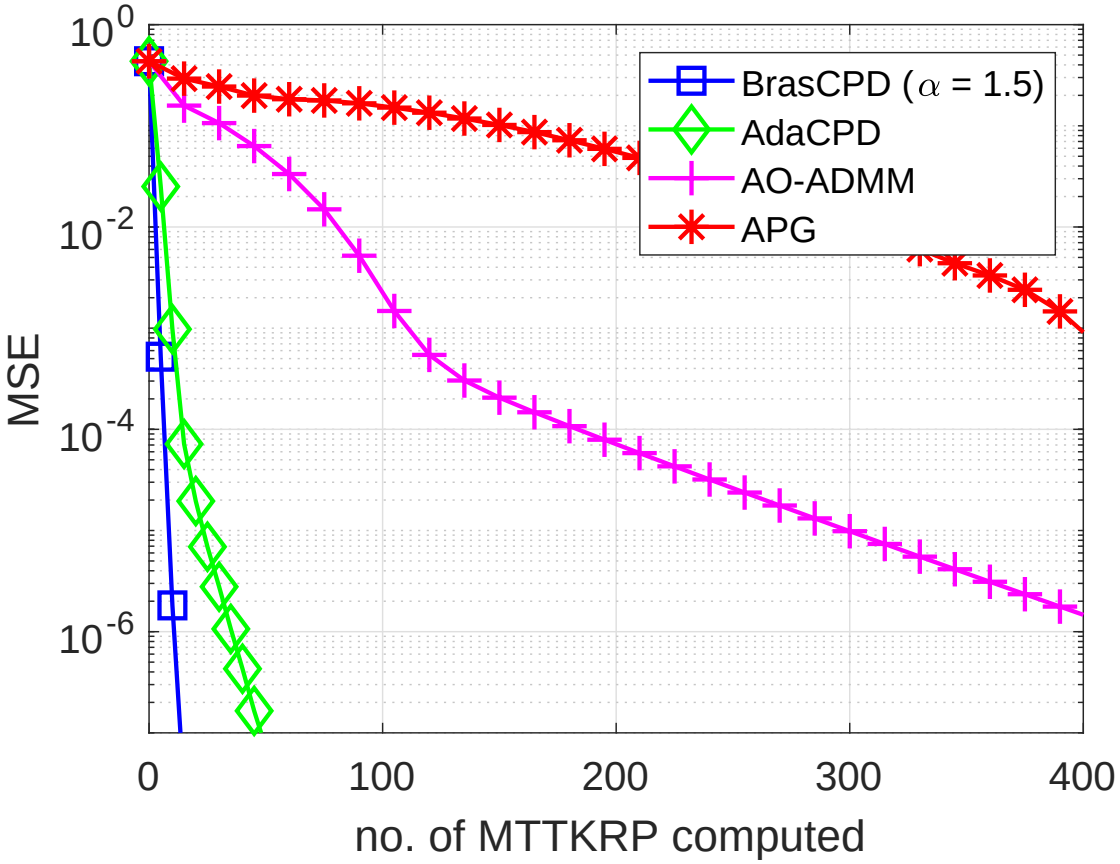


Figure 1.1: The proposed algorithms (`AdaCPD` and `BrasCPD`) exhibit low complexity for achieving high accuracy of the estimated latent factors. The tensor under test has a size of $100 \times 100 \times 100$ and the rank is 10. The latent factors are constrained to be nonnegative. The baselines are two state-of-art constrained CPD algorithms `AO-ADMM` [25] and `APG` [52].

Chapter 2 Background

2.1 Preliminary

In this chapter, we will introduce some basic knowledge of tensors used in state-of-art and our proposed approach. Specifically, we will focus on tensor fibers, mactricization, and Canonical Polyadic Decomposition (CPD). Before that, we will also talk about some notations and useful product operations for tensors.

2.1.1 Notation

In this thesis, we denote scalars by lower case letters $x \in \mathbb{R}$, vectors by lower case bold letters $\boldsymbol{x} \in \mathbb{R}^{I_1}$, matrices by upper case bold letters $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2}$, and higher order tensors by upper case bold calligraphic letters $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$.

2.1.2 Useful Product Operations

Outer product: The *vector outer product* is defined as the product of the vector's elements. This operation is denoted by the $\circ$ symbol. The vector outer product of two n-dimensional vectors $\boldsymbol{a}$, $\boldsymbol{b}$ is defined as follows and produces a matrix $\boldsymbol{X}$:

$$\boldsymbol{X} = \boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}^T \tag{2.1}$$

By the definition of tensors, we can define a tensor $\boldsymbol{\mathcal{X}}$ as the outer product of N vectors:

$$\boldsymbol{\mathcal{X}} = \boldsymbol{a}_{(1)} \circ \boldsymbol{a}_{(2)} \circ \cdots \circ \boldsymbol{a}_{(N)} \tag{2.2}$$

Inner product: The *inner product* of two n-dimensional vectors $\boldsymbol{a}$, $\boldsymbol{b}$ is defined as

$$x = \langle \boldsymbol{a}, \boldsymbol{b} \rangle = \boldsymbol{a}^T \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \tag{2.3}$$

and generates a scalar x.

Kronecker product: The Kronecker product of $\boldsymbol{A}$ $(I \times K)$ and $\boldsymbol{B}$ $(J \times L)$ produces a $IJ \times KL$ matrix

$$\boldsymbol{A} \otimes \boldsymbol{B} := \begin{bmatrix} \boldsymbol{BA}(1,1) & \boldsymbol{BA}(1,2) & \cdots & \boldsymbol{BA}(1,K) \\ \boldsymbol{BA}(2,1) & \boldsymbol{BA}(2,2) & \cdots & \boldsymbol{BA}(2,K) \\ \vdots & \vdots & \cdots & \vdots \\ \boldsymbol{BA}(I,1) & \boldsymbol{BA}(I,2) & \cdots & \boldsymbol{BA}(I,K) \end{bmatrix}$$

Khatri–Rao product: One important product operation is the Khatri–Rao (column-wise Kronecker) product of two matrices *with the same number of columns*. Assuming we have two matrices which are $\boldsymbol{A} = [\boldsymbol{a}_1, \cdots, \boldsymbol{a}_R]$($\boldsymbol{a}_i$ denotes the corresponding columns of matrix $\boldsymbol{A}$) and $\boldsymbol{B} = [\boldsymbol{b}_1, \cdots, \boldsymbol{b}_R]$($\boldsymbol{b}_i$ denotes the corresponding columns of matrix $\boldsymbol{B}$), the Khatri–Rao product of $\boldsymbol{A}$ and $\boldsymbol{B}$ is

$$\boldsymbol{A} \odot \boldsymbol{B} := [\boldsymbol{a}_1 \otimes \boldsymbol{b}_1, \cdots \boldsymbol{a}_R \otimes \boldsymbol{b}_R]\,. \tag{2.4}$$

Hadamard product: The *Hadamard product* of two same-dimensional matrices $\boldsymbol{A} \in \mathbb{R}^{I \times J}$ and $\boldsymbol{B} \in \mathbb{R}^{I \times J}$, $\boldsymbol{A} * \boldsymbol{B} \in \mathbb{R}^{I \times J}$, is defined as the element-wise matrix product(a_{ij} and b_{ij} are the corresponding elements in matrix $\boldsymbol{A}$ and $\boldsymbol{B}$).

$$\boldsymbol{A} * \boldsymbol{B} := \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1J}b_{1J} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2J}b_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}b_{I1} & a_{I2}b_{I2} & \cdots & a_{IJ}b_{IJ} \end{bmatrix} \tag{2.5}$$

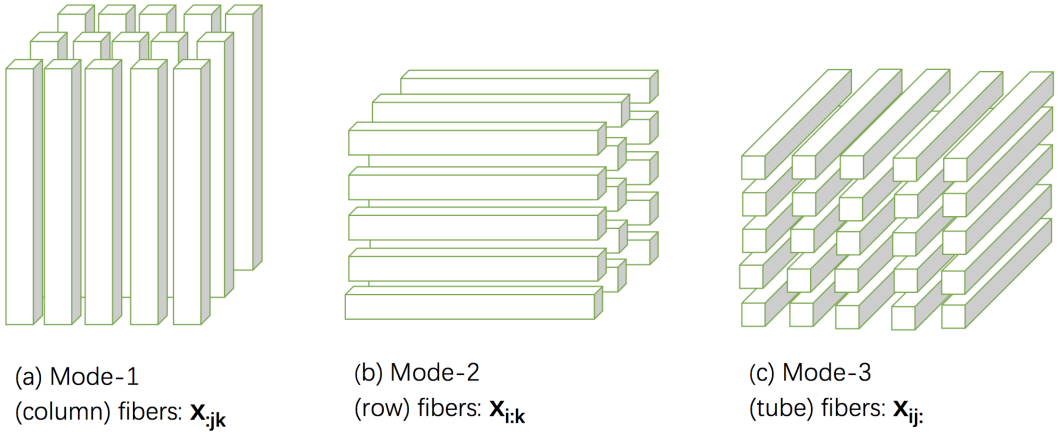


Figure 2.1: Column, row, and tube fibers of a mode-3 tensor

2.2 Canonical polyadic decomposition (CPD)

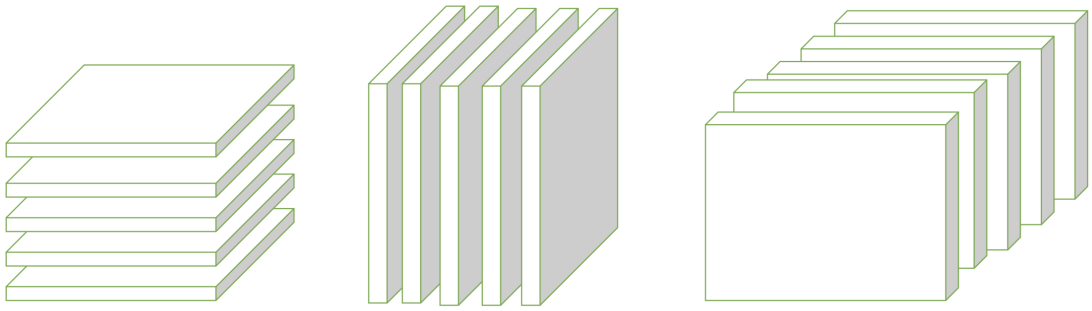
2.2.1 Tensor Fibers and Slices

We can have tensor fibers and slices by fixing some of the given tensor's indices. *Fibers* are created when fixing all but one index, *slices* are created when fixing all but two indices. For a third order tensor the fibers are given as $\boldsymbol{x}_{:jk}$ (column), $\boldsymbol{x}_{i:k}$ (row), and $\boldsymbol{x}_{ij:}$ (tube); the slices are given as $\boldsymbol{X}_{::k}$ (frontal), $\boldsymbol{X}_{:j:}$ (lateral), $\boldsymbol{X}_{i::}$ (horizontal). We show the examples of fibers and slices for a 3-way tensor in Figure 2.1 and 2.2.

2.2.2 Rank-one Tensor

Before talking about rank-one tensor, let's take a look at rank-one matrix. A rank-one matrix $\boldsymbol{X}$ of size $I \times J$ is an outer product of two vectors: $\boldsymbol{X}(i,j) = \boldsymbol{a}(i)\boldsymbol{b}(j)$, $\forall i \in \{1, \cdots, I\}$, $\forall j \in \{1, \cdots, J\}$: i.e.,

$$\boldsymbol{X} = \boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}^T \tag{2.6}$$



(a) Horizontal slices: $\boldsymbol{X}_{i::}$ (b) Lateral slices: $\boldsymbol{X}_{:j:}$ (c) Frontal slices: $\boldsymbol{X}_{::k}$

Figure 2.2: Horizontal, lateral and frontal slices of a mode-3 tensor

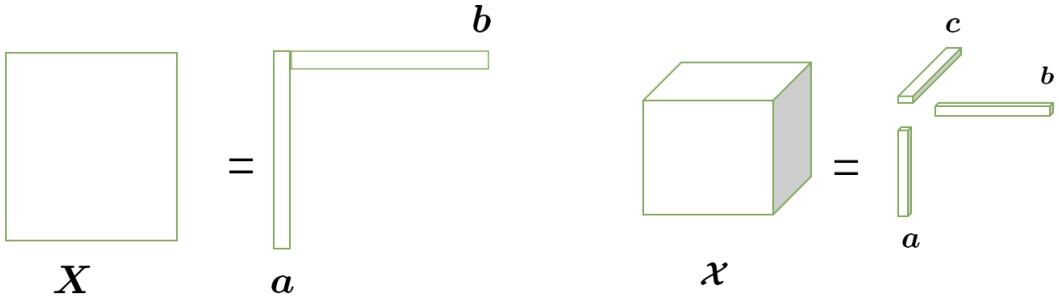


Figure 2.3: Rank-one matrix and rank-one third-order tensor

A rank-one third-order tensor $\boldsymbol{\mathcal{X}}$ of size $I \times J \times K$ is an outer product of three vectors: $\boldsymbol{\mathcal{X}}(i,j,k) = \boldsymbol{a}(i)\boldsymbol{b}(j)\boldsymbol{c}(k)$: i.e.,

$$\boldsymbol{\mathcal{X}} = \boldsymbol{a} \circ \boldsymbol{b} \circ \boldsymbol{c} \tag{2.7}$$

And we show a simple example of rank-one matrix and rank-one third-order tensor in Figure 2.3.

2.2.3 CPD

The polyadic decomposition (PD) means that we can write an Nth-order tensor $\boldsymbol{\mathcal{X}}$ as a sum of rank-1 terms and each of them is an outer product of N nonzero vectors. For

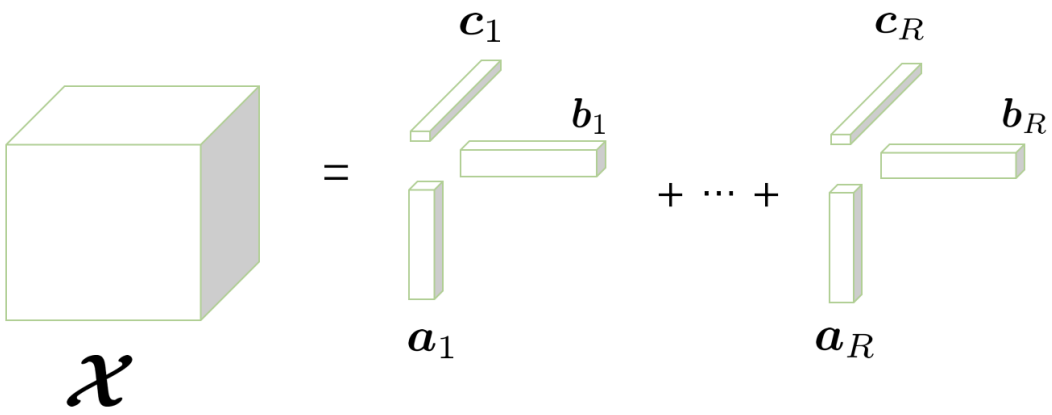


Figure 2.4: A (canonical) polyadic decomposition writes a tensor as a (minimal) sum of R rank-1 terms.

example, for a third-order tensor, we can write

$$\boldsymbol{\mathcal{X}} = \sum_{r=1}^{R} \boldsymbol{a_r} \circ \boldsymbol{b_r} \circ \boldsymbol{c_r} \tag{2.8}$$

We show an example of polyadic decomposition in Figure 1.4.

The minimum number of rank-one terms R that makes (2.8) hold is the rank of $\boldsymbol{\mathcal{X}}$. Furthermore, we name PD with rank R as the canonical polyadic decomposition (CPD).

CPD has a significant property which is its essential uniqueness under mild conditions. Given tensor $\boldsymbol{\mathcal{X}}$ of rank R , its CPD is essentially unique if and only if the R rank-1 terms in its decomposition are unique. Nontheless, computing CPD is an NP-hard problem. In the next chapter, some classic CPD algorithms will be introduced.

2.2.4 Matricization

There is one very important tensor algebra operation which we call it mactricization. Mactricization can be explained by a third-order tensor example. We always consider a third-order tensor $\boldsymbol{\mathcal{X}}$ as a set of slices, and each slice is a matrix. We can use matricization operation to reorder a tensor into a matrix. The operation consists of taking out each

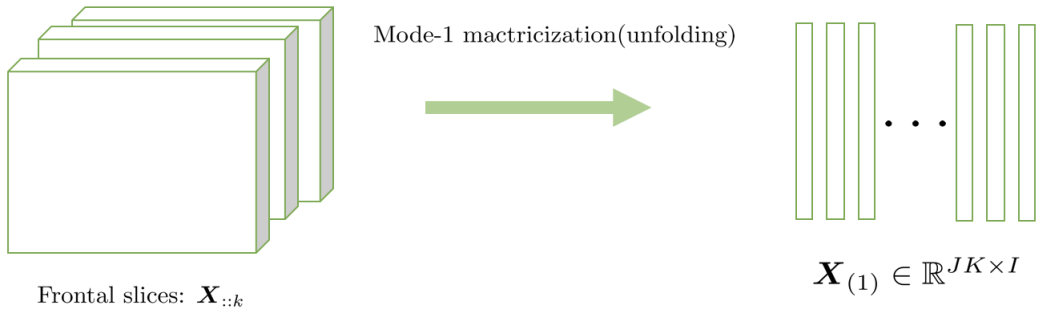


Figure 2.5: Mode-1 mactricization(unfolding) of a third-order tensor

slice and vectorizing it, then rearranging them into a matrix. If we are given a third-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ with its factor matrix $\boldsymbol{A} \in \mathbb{R}^{I \times R}$, $\boldsymbol{B} \in \mathbb{R}^{J \times R}$ and $\boldsymbol{C} \in \mathbb{R}^{K \times R}$, the mactricizations of tensor $\boldsymbol{\mathcal{X}}$ are given below:

$$\begin{aligned} \boldsymbol{X}_{(1)} &= (\boldsymbol{C} \odot \boldsymbol{B})\boldsymbol{A}^T \\ \boldsymbol{X}_{(2)} &= (\boldsymbol{C} \odot \boldsymbol{A})\boldsymbol{B}^T \\ \boldsymbol{X}_{(3)} &= (\boldsymbol{B} \odot \boldsymbol{A})\boldsymbol{C}^T \end{aligned} \tag{2.9}$$

In order to illustrate the formula introduced above, we show a simple example of the matricization of a third-order tensor in Figure 2.5.

2.3 Summary

In this chapter, we have introduced the fundamental knowledge of tensors and its different operations. Five different operations have been introduced here. After that, we talked about CPD and its properties. Based on all of these, we will discuss different algorithms for solving CPD problems.

Chapter 3 Related Work

In this chapter, we review some existing CPD algorithms. Firstly, we will introduce Alternating Least Squares (ALS) [11] which is the classical algorithm to solve CPD problems. We will also introduce matricized tensor times Khatri-Rao product (MTTKRP) which is the most costly operation in the ALS [11] algorithm. Then, we will introduce two constrained CPD algorithms: AO-ADMM [25] and APG [52], in which subproblems are solved by constrained optimization techniques. Following that, we will introduce two stochastic optimization based algorithms: RBS [49] and CPRAND [4] in which researchers take advantage of sampling strategies to reduce per-iteration complexity. The details of these algorithms will be described in this chapter.

3.1 Classical Algorithm

The most popular and arguably method in CPD is what we call "Alternating Least Squares" (ALS) [11]. The key step for alternating least squares is to fix all other factor matrices and try to optimize the only left one non-fixed matrix. Then we need to repeat this step again and again for every factor matrix until reaching some stopping criterion [37].

3.1.1 Alternating Least Squares (ALS)

We can write down the formulation of third-order CPD case as follows:

$$\underset{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}}{\text{minimize}} \left\| \boldsymbol{\mathcal{X}} - \sum_{r=1}^{R} \boldsymbol{A}(:,\ r) \circ \boldsymbol{B}(:,r) \circ \boldsymbol{C}(:,r) \right\|_F^2 . \tag{3.1}$$

To tackle the above, we can rewrite this problem in a matricized form:

$$\begin{aligned} \boldsymbol{X}_{(1)} &= (\boldsymbol{C} \odot \boldsymbol{B})\boldsymbol{A}^T \\ \boldsymbol{X}_{(2)} &= (\boldsymbol{C} \odot \boldsymbol{A})\boldsymbol{B}^T \\ \boldsymbol{X}_{(3)} &= (\boldsymbol{B} \odot \boldsymbol{A})\boldsymbol{C}^T \end{aligned} \tag{3.2}$$

For the third-order tensor case, the ALS algorithm would perform the following steps repeatedly until convergence.

$$\begin{aligned} \boldsymbol{A} &\leftarrow \arg\min_{\boldsymbol{A}} ||\boldsymbol{X}_{(1)} - (\boldsymbol{C} \odot \boldsymbol{B})\boldsymbol{A}^T||_F^2 \\ \boldsymbol{B} &\leftarrow \arg\min_{\boldsymbol{B}} ||\boldsymbol{X}_{(2)} - (\boldsymbol{C} \odot \boldsymbol{A})\boldsymbol{B}^T||_F^2 \\ \boldsymbol{C} &\leftarrow \arg\min_{\boldsymbol{C}} ||\boldsymbol{X}_{(3)} - (\boldsymbol{B} \odot \boldsymbol{A})\boldsymbol{C}^T||_F^2 \end{aligned} \tag{3.3}$$

The optimal solution to this minimization problem is given by

$$\begin{aligned} \boldsymbol{A}^T &= [(\boldsymbol{C} \odot \boldsymbol{B})]^\dagger \boldsymbol{X}_{(1)} = (\boldsymbol{C}^T\boldsymbol{C} * \boldsymbol{B}^T\boldsymbol{B})^{-1}(\boldsymbol{C} \odot \boldsymbol{B})^T\boldsymbol{X}_{(1)} \\ \boldsymbol{B}^T &= [(\boldsymbol{C} \odot \boldsymbol{A})]^\dagger \boldsymbol{X}_{(2)} = (\boldsymbol{C}^T\boldsymbol{C} * \boldsymbol{A}^T\boldsymbol{A})^{-1}(\boldsymbol{C} \odot \boldsymbol{A})^T\boldsymbol{X}_{(2)} \\ \boldsymbol{C}^T &= [(\boldsymbol{B} \odot \boldsymbol{A})]^\dagger \boldsymbol{X}_{(3)} = (\boldsymbol{B}^T\boldsymbol{B} * \boldsymbol{A}^T\boldsymbol{A})^{-1}(\boldsymbol{B} \odot \boldsymbol{A})^T\boldsymbol{X}_{(3)} \end{aligned} \tag{3.4}$$

Let's take a look at the flops analysis of optimal solution for $\boldsymbol{A}$:

- $\boldsymbol{C}^T\boldsymbol{C}$ will consume $\mathcal{O}(R^2K)$ flops and $\boldsymbol{B}^T\boldsymbol{B}$ will consume $\mathcal{O}(R^2J)$ flops. $(\boldsymbol{C}^T\boldsymbol{C} * \boldsymbol{B}^T\boldsymbol{B})^{-1}$ is a $R \times R$ matrix with inverse operation $\mathcal{O}(R^3)$. R is usually not very large.

- Using $\boldsymbol{H}$ to denote the Khatri–Rao product of different factor matrix:

$$\boldsymbol{H}_{(1)} = \boldsymbol{C} \odot \boldsymbol{B}; \boldsymbol{H}_{(2)} = \boldsymbol{C} \odot \boldsymbol{A}; \boldsymbol{H}_{(3)} = \boldsymbol{B} \odot \boldsymbol{A}. \tag{3.5}$$

- The so-called *matricized tensor times Khatri-Rao product* (MTTKRP) operation, i.e.,

$$\mathsf{MTTKRP}: \qquad \boldsymbol{H}_{(n)}^T\boldsymbol{X}_{(n)}$$

- Assuming $\boldsymbol{X}_{(1)} \in \mathbb{R}^{JK \times I}$ and $\boldsymbol{H}_{(1)} \in \mathbb{R}^{JK \times R}$,$\boldsymbol{H}_{(1)}^T\boldsymbol{X}_{(1)}$ will cost $\mathcal{O}(IJKR)$ flops.

Algorithm 1 ALS algorithm

1: **Input**: Third-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I\times J\times K}$; rank R; initialization $\{\boldsymbol{A}^{(0)}\}, \{\boldsymbol{B}^{(0)}\}, \{\boldsymbol{C}^{(0)}\}$, step size $\{\alpha^{(t)}\}_{t=0,\dots}$
2: $t \leftarrow 0$;
3: **repeat**
4: $\quad \boldsymbol{A}^{(t)^T} \leftarrow [(\boldsymbol{C}^{(t-1)} \odot \boldsymbol{B}^{(t-1)})]^{\dagger}\boldsymbol{X}_{(1)} = (\boldsymbol{C}^{(t-1)T}\boldsymbol{C}^{(t-1)} * \boldsymbol{B}^{(t-1)T}\boldsymbol{B}^{(t-1)})^{-1}(\boldsymbol{C}^{(t-1)} \odot \boldsymbol{B}^{(t-1)})^T\boldsymbol{X}_{(1)}$
5: $\quad \boldsymbol{B}^{(t)^T} \leftarrow [(\boldsymbol{C}^{(t-1)} \odot \boldsymbol{A}^{(t)})]^{\dagger}\boldsymbol{X}_{(2)} = (\boldsymbol{C}^{(t-1)T}\boldsymbol{C}^{(t-1)} * \boldsymbol{A}^{(t)T}\boldsymbol{A}^{(t)})^{-1}(\boldsymbol{C}^{(t-1)} \odot \boldsymbol{A}^{(t)})^T\boldsymbol{X}_{(2)}$
6: $\quad \boldsymbol{C}^{(t)^T} \leftarrow [(\boldsymbol{B}^{(t)} \odot \boldsymbol{A}^{(t)})]^{\dagger}\boldsymbol{X}_{(3)} = (\boldsymbol{B}^{(t)T}\boldsymbol{B}^{(t)} * \boldsymbol{A}^{(t)T}\boldsymbol{A}^{(t)})^{-1}(\boldsymbol{B}^{(t)} \odot \boldsymbol{A}^{(t)})^T\boldsymbol{X}_{(3)}$
7: **until** stopping criterion is reached
8: **Output**: $\{\boldsymbol{A}^{(t)T}\}, \{\boldsymbol{B}^{(t)T}\}, \{\boldsymbol{C}^{(t)T}\}$

In particular, the so-called *matricized tensor times Khatri-Rao product* (MTTKRP) operation is always the most costly operation in computing ALS [11] algorithm. This is quite costly even if I, J, K is moderately large. Many prior works [46], [55] used sparsity of $\boldsymbol{\mathcal{X}}$ to accelerate MTTKRP. Once the tensor is dense, methods in [46], [55] do not work.

Here we can see that the ALS algorithm listed in Algorithm 1 is easy to understand and code. And we do not need to tune any parameters in the algorithm which is convenient for us to apply. However, convergence can be slow when dealing with large-scale tensors and MTTKRP of ALS will be costly.

3.2 Constrained Case Algorithms

In a lot of problems, we consider:

$$\underset{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}}{\text{minimize}} \left\| \boldsymbol{\mathcal{X}} - \sum_{r=1}^{R} \boldsymbol{A}(:,\ r) \circ \boldsymbol{B}(:,r) \circ \boldsymbol{C}(:,r) \right\|_F^2 + r_1(\boldsymbol{A}) + r_2(\boldsymbol{B}) + r_3(\boldsymbol{C}) \tag{3.6}$$

where $r_n(\cdot)$ denotes constraints and regularizations. In many applications, feasible solutions might lie in the constrained space. For example, when dealing with hyperspectral images [26] which involve two spatial domain and one spectrum domain, all the elements in hyperspectral images are nonnegative.

In this section, we will introduce two algorithms: AO-ADMM [25] and APG [52]. Both of them are under ALS [11] framework and solve subproblems by using constrained optimization techniques.

3.2.1 AO-ADMM

Consider a constrained CPD problem as below:

$$\underset{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}}{\text{minimize}} \left\| \boldsymbol{\mathcal{X}} - \sum_{r=1}^{R} \boldsymbol{A}(:,\ r) \circ \boldsymbol{B}(:,r) \circ \boldsymbol{C}(:,r) \right\|_F^2 + r_1(\boldsymbol{A}) + r_2(\boldsymbol{B}) + r_3(\boldsymbol{C}) \tag{3.7}$$

where $r_n(\cdot)$ represents constraints or regularizations here. We can add different constraints and regularizations to factor matrix $\boldsymbol{A}, \boldsymbol{B}$ and $\boldsymbol{C}$ by employing different constraints and regularizations $r(\cdot)$. For instance, we can use a nonnegativity constraint by applying the indicator function of $\mathbb{R}_+$ and sparsity regularization by employing $\|\|_1$.

It is obvious that equation (3.7) is a non-convex problem. And we always employ ALS (alternating least squares) framework to solve this problem. If there are no constraints or regularizations $r(\cdot)$ here, this framework is the same as the alternating least squares (ALS) algorithm. Huang [25] has proposed AO-ADMM which combines the insights of ALS framework and ADMM (alternating direction method of multipliers). ADMM is known as a popular framework for solving constrained optimization problems [9]. AO-ADMM employs the advantage of both methods. For the AO framework, it uses its monotonically decreasing objective function. For the ADMM framework, it leverages its flexible ability to handle different constraints.

In fact, AO-ADMM resolves the constrained CPD problem with a sequence of outer and inner iterations. As we can see, the outer iteration aims at optimizing one of the matrix factors. Then, the inner iteration focuses on solving constraints. And we apply this method for all factor matrices repeatedly until some stop criterion is achieved.

When it comes to ADMM, Huang has proposed that we can write the primal and dual variables as $\boldsymbol{H}_{(1)} = \boldsymbol{C} \odot \boldsymbol{B} \in \mathbb{R}^{JK \times R}$, $\boldsymbol{A} \in \mathbb{R}^{I \times R}$ and $\boldsymbol{U}_{(1)} \in \mathbb{R}^{I \times R}$, respectively. Here we will introduce an auxiliary variable $\hat{\boldsymbol{A}} \in \mathbb{R}^{R \times I}$ which satisfies a constrained

Algorithm 2 Solve (3.8) using ADMM [25]

1: **Input**: $\boldsymbol{X}_{(1)}$, $\boldsymbol{H}_{(1)}$, $\boldsymbol{A}$, $\boldsymbol{U}_{(1)}$, R
2: $\boldsymbol{G} = \boldsymbol{H}_{(1)}^T \boldsymbol{H}_{(1)}$;
3: $\rho = \text{trace}(\boldsymbol{G})/R$;
4: Calculate $\boldsymbol{L}$ from the Cholesky decomposition of $\boldsymbol{G} + \rho \boldsymbol{I} = \boldsymbol{L}\boldsymbol{L}^{\boldsymbol{T}}$;
5: $\boldsymbol{F} = \boldsymbol{H}^T \boldsymbol{X}_{(1)}$;
6: **repeat**
7: $\quad \tilde{\boldsymbol{A}} \leftarrow \boldsymbol{L}^{-\boldsymbol{T}} \boldsymbol{L}^{-\boldsymbol{1}} (\boldsymbol{F} + \rho(\boldsymbol{A} + \boldsymbol{U}_{(1)})^T)$
8: $\quad \boldsymbol{A} \leftarrow \arg\min_{\boldsymbol{A}} r(\boldsymbol{A}) + \frac{\rho}{2} \|\boldsymbol{A} - \tilde{\boldsymbol{A}}^T + \boldsymbol{U}_{(1)}\|_F^2$
9: $\quad \boldsymbol{U}_{(1)} \leftarrow \boldsymbol{U}_{(1)} + \boldsymbol{A} - \tilde{\boldsymbol{A}}^T$
10: **until** stopping criterion is reached
11: **Output**: $\{\boldsymbol{A}\}$ and $\{\boldsymbol{U}_{(1)}\}$

optimization problem in the formulation of ADMM:

$$\begin{aligned} \underset{\boldsymbol{A}, \tilde{\boldsymbol{A}}}{\text{minimize}} \quad & \|\boldsymbol{X}_{(1)} - \boldsymbol{H}_{(1)} \tilde{\boldsymbol{A}}\|_F^2 + r(\boldsymbol{A}) \\ & \boldsymbol{A} = \tilde{\boldsymbol{A}}^T. \end{aligned} \tag{3.8}$$

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

$$\begin{aligned} \tilde{\boldsymbol{A}} &\leftarrow (\boldsymbol{H}_{(1)}^T \boldsymbol{H}_{(1)} + \rho \boldsymbol{I})^{-1} (\boldsymbol{H}_{(1)}^T \boldsymbol{X}_{(1)} + \rho(\boldsymbol{A} + \boldsymbol{U}_{(1)}^T); \\ \boldsymbol{A} &\leftarrow \arg\min_{\boldsymbol{A}} r(\boldsymbol{A}) + \frac{\rho}{2} \|\boldsymbol{A} - \tilde{\boldsymbol{A}}^T + \boldsymbol{U}_1\|_F^2; \\ \boldsymbol{U}_{(1)} &\leftarrow \boldsymbol{U}_{(1)} + \boldsymbol{A} - \tilde{\boldsymbol{A}}^T. \end{aligned} \tag{3.9}$$

Solving (3.8) using ADMM algorithm is shown in Algorithm 2:

Huang [25] has mentioned that setting $\rho = \text{trace}(\boldsymbol{G})/R$ in Line 3 was inspired by [41]. And the choice of ρ here is an approximation of optimal ρ which was proposed in [41]. In order to save computations of Line 7 in Algorithm 2, Huang also proposed that we can employ Cholesky decomposition to compute $(\boldsymbol{G} + \rho \boldsymbol{I})$ in Line 4 of Algorithm 2.

Combining the two different frameworks, Huang [25] introduced the complete AO-ADMM framework in Algorithm 3. We update factor matrices by using Algorithm 2 repeatedly. It requires $\mathcal{O}(IJKR)$ flops to compute the MTTKRP operation in each iteration and is often the most resource-consuming step of the AO-ADMM framework.

Let's take a look at per-iteration complexity of AO-ADMM [25]. In line 2, there

Algorithm 3 Solving (3.7) using AO-ADMM [25]

1: **Initialize**: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$
2: **Initialize**: $\boldsymbol{U}_{(1)}$, $\boldsymbol{U}_{(2)}$, $\boldsymbol{U}_{(3)}$ to be all zero matrices
3: **repeat**
4: $\quad \boldsymbol{X}_{(1)} = (\boldsymbol{C} \odot \boldsymbol{B})\boldsymbol{A}^T$
5: $\quad \boldsymbol{X}_{(2)} = (\boldsymbol{C} \odot \boldsymbol{A})\boldsymbol{B}^T$
6: $\quad \boldsymbol{X}_{(3)} = (\boldsymbol{B} \odot \boldsymbol{A})\boldsymbol{C}^T$
7: $\quad \boldsymbol{H}_{(1)} = \boldsymbol{C} \odot \boldsymbol{B}$
8: $\quad \boldsymbol{H}_{(2)} = \boldsymbol{C} \odot \boldsymbol{A}$
9: $\quad \boldsymbol{H}_{(3)} = \boldsymbol{B} \odot \boldsymbol{A}$
10: $\quad$ update $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{U}_{(1)}$, $\boldsymbol{U}_{(2)}$, $\boldsymbol{U}_{(3)}$ using Alg. 2 initialized with the previous $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{U}_{(1)}$, $\boldsymbol{U}_{(2)}$, $\boldsymbol{U}_{(3)}$
11: **until** some stopping criterion is reached
12: **Output**: $\{\boldsymbol{A}\}, \{\boldsymbol{B}\}, \{\boldsymbol{C}\}$

are $R^2 \times (2I - 1)$ flops. For line 3, there is R flops. In line 4, the flops of Cholesky decomposition are $1/3(R^3 + R^2 \times I)$. In line 5, it costs $J \times K \times R(2I - 1)$ flops. For line 7, there are R^3 flops for the two inverse. And $(\boldsymbol{L}^T)^{-1}(\boldsymbol{L}^{-1})$ costs $R^2 \times (2R - 1)$ flops. $(\boldsymbol{A} + \boldsymbol{U}_{(1)})$ consumes $I \times R$ flops. And the multiplication part consumes $I \times R$ flops and add costs $I \times R$ flops. The operation of $(\boldsymbol{L}^T)^{-1}(\boldsymbol{L}^{-1})$ multiples the result of $(\boldsymbol{F}+\rho(\boldsymbol{A}+\boldsymbol{U}_{(1)})^T)$ will cost $I \times R \times (2R-1)$ flops. And update part costs $J \times K \times R(2R-1)$ flops. For this part, it loops for 5 times in each iteration. In line 8, the flops consuming depends on the proximal operator here. For line 9, the add and subtract parts cost $2 \times J \times K \times R$ flops. As we can see here, the most costly operation in AO-ADMM is still MTTKRP.

The convergence of AO-ADMM is summarized in Proposition 1.

Proposition 1 *If the sequence generated by AO-ADMM in Alg. 3 is bounded, then for third or higher-order tensor, AO-ADMM converges to a stationary point of (3.7).*

The proof can be found in Theorem 2 [25].

AO-ADMM is a good strategy to solve constrained tensor decomposition problems which take the advantage of the ADMM frame work to handle different situations. However, MTTKRP can not be avoided when we apply ADMM algorithm which will be costly.

3.2.2 Alternating Proximal Gradient (APG)

Consider the optimization problem below:

$$\underset{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}}{\text{minimize}} \left\| \boldsymbol{\mathcal{X}} - \sum_{r=1}^{R} \boldsymbol{A}(:,\ r) \circ \boldsymbol{B}(:,r) \circ \boldsymbol{C}(:,r) \right\|_F^2 + r_1(\boldsymbol{A}) + r_2(\boldsymbol{B}) + r_3(\boldsymbol{C}) \tag{3.10}$$

where we have three different blocks $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ here. We can see from the equation that the feasible set of variables and the objective function are not convex jointly, but convex in each block $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$. Note that

$$\left\| \boldsymbol{\mathcal{X}} - \sum_{r=1}^{R} \boldsymbol{A}(:,\ r) \circ \boldsymbol{B}(:,r) \circ \boldsymbol{C}(:,r) \right\|_F^2 \tag{3.11}$$

is differentiable and we assume that r_i for $i = 1, 2, 3$ are the convex functions.

Xu [52] proposed an ALS frame based proximal gradient method to solve the constrained CPD problems. In his approach, if we fix all other variables except one of the variable, then this objective function becomes convex. We also define the function $r_1(\boldsymbol{A}) = \infty$ if $\boldsymbol{A} \notin \text{dom}(r_1)$, $r_2(\boldsymbol{B}) = \infty$ if $\boldsymbol{B} \notin \text{dom}(r_2)$ and $r_3(C) = \infty$ if $\boldsymbol{C} \notin \text{dom}(r_3)$.

Since it is very hard for us to update all the variables at the same time, it is much easier for people to update a block at a time. Similar to ALS, people have been used alternating minimization method to solve this problem for a long time. When updating a block at a time, we name this method as block coordinate descent (BCD) which updates variables block by block. We define a function f:

$$f = \left\| \boldsymbol{\mathcal{X}} - \sum_{r=1}^{R} \boldsymbol{A}(:,\ r) \circ \boldsymbol{B}(:,r) \circ \boldsymbol{C}(:,r) \right\|_F^2 \tag{3.12}$$

And we have:

$$\begin{aligned} \boldsymbol{X}_{(1)} &= (\boldsymbol{C} \odot \boldsymbol{B})\boldsymbol{A}^T \\ \boldsymbol{X}_{(2)} &= (\boldsymbol{C} \odot \boldsymbol{A})\boldsymbol{B}^T \\ \boldsymbol{X}_{(3)} &= (\boldsymbol{B} \odot \boldsymbol{A})\boldsymbol{C}^T \end{aligned} \tag{3.13}$$

$$\begin{aligned} \boldsymbol{H}_{(1)} &= \boldsymbol{C} \odot \boldsymbol{B} \\ \boldsymbol{H}_{(2)} &= \boldsymbol{C} \odot \boldsymbol{A} \\ \boldsymbol{H}_{(3)} &= \boldsymbol{B} \odot \boldsymbol{A} \end{aligned} \tag{3.14}$$

When considering updating $\boldsymbol{A}$ at iteration t, from above, we can have:

$$f(\boldsymbol{A}) = \left\| \boldsymbol{X}_{(1)} - \boldsymbol{H}_{(1)} \boldsymbol{A}^T \right\|_F^2 \tag{3.15}$$

and

$$\nabla_{\boldsymbol{A}} f = \boldsymbol{A}^{(t)} \boldsymbol{H}_{(1)}^T \boldsymbol{H}_{(1)} - \boldsymbol{X}_{(1)}^T \boldsymbol{H}_{(1)} \tag{3.16}$$

Let

$$\boldsymbol{H}_{(1)}^{(t-1)} = \boldsymbol{C}^{(t-1)} \odot \boldsymbol{B}^{(t-1)} \tag{3.17}$$

We take

$$L_{(1)}^{(t-1)} = ||\boldsymbol{H}_{(1)}^{(t-1)T} \boldsymbol{H}_{(1)}^{(t-1)}||_2, \qquad \omega_1^{(t-1)} = \min(\tilde{\omega}^{(t-1)}, \delta_\omega \sqrt{\frac{L_{(1)}^{(t-2)}}{L_{(1)}^{(t-1)}}}) \tag{3.18}$$

$\delta_\omega < 1$ is preselected, and $\tilde{\omega}^{(t-1)} = \frac{k^{(t-1)}-1}{k^{(t)}}$ with

$$k^{(0)} = 1, k^{(t)} = \frac{1}{2}(1 + \sqrt{1 + 4(k^{(t-1)})^2}) \tag{3.19}$$

And we use an extrapolation weight to update

$$\tilde{\boldsymbol{A}}^{(t-1)} = \boldsymbol{A}^{(t-1)} + \omega_1^{(t-1)} (\boldsymbol{A}^{(t-1)} - \boldsymbol{A}^{(t-2)}) \tag{3.20}$$

and let

$$\tilde{\boldsymbol{G}}_{(1)}^{(t-1)} = (\boldsymbol{A}^{(t-1)} H_{(1)}^{(t-1)T} - \boldsymbol{X}_{(1)}) H_{(1)}^{(t-1)} \tag{3.21}$$

be the gradient. Then the update rule is:

$$A^{(t)} = \arg\min_{\boldsymbol{A}} < \tilde{\boldsymbol{G}}_{(1)}^{(t-1)}, \boldsymbol{A} - \tilde{\boldsymbol{A}}^{(t-1)} > + \frac{L_{(1)}^{(t-1)}}{2} ||\boldsymbol{A} - \tilde{\boldsymbol{A}}^{(t-1)}||_F^2 + r_1(\boldsymbol{A}) \tag{3.22}$$

Algorithm 4 Block coordinate descent method for solving the problem (3.10) [52]

1: **Input**: Nonnegative 3-way tensor $\boldsymbol{\mathcal{X}}$ and rank R
2: **Initialize**: Choose a positive number $\sigma_\omega < 1$ and randomize $\boldsymbol{A}_{(n)}^{-1} = \boldsymbol{A}_{(n)}^{0}$, $n = 1, ..., N$, as nonnegative matrices of appropriate sizes
3: **while** stopping criterion is not satisfied and $t = 1, 2, \ldots$ and $i = 1, 2, 3$ **do**
4: Compute $L_{(i)}^{(t-1)}$, and set $\omega_{(i)}^{(t-1)}$ according to (3.18)
5: Let $\tilde{\boldsymbol{A}}^{(t-1)} = \boldsymbol{A}^{(t-1)} + \omega_{(1)}^{(t-1)}(\boldsymbol{A}^{(t-1)} - \boldsymbol{A}^{(t-2)})$,
6: $\tilde{\boldsymbol{B}}^{(t-1)} = \boldsymbol{B}^{(t-1)} + \omega_{(1)}^{(t-1)}(\boldsymbol{B}^{(t-1)} - \boldsymbol{B}^{(t-2)})$,
7: $\tilde{\boldsymbol{C}}^{(t-1)} = \boldsymbol{C}^{(t-1)} + \omega_{(1)}^{(t-1)}(\boldsymbol{C}^{(t-1)} - \boldsymbol{C}^{(t-2)})$ Update $\boldsymbol{A}, \boldsymbol{B}$ and $\boldsymbol{C}$ according to (3.23)
8: **if** $f(\boldsymbol{A}^{(t)}) \geq f(\boldsymbol{A}^{(t-1)})$ **then**
Reupdated $\boldsymbol{A}^{(t)}$ according to (3.23) with $\tilde{\boldsymbol{A}}^{(t-1)} = \boldsymbol{A}^{(t-1)}$;
9: **if** $f(\boldsymbol{B}^{(t)}) \geq f(\boldsymbol{B}^{(t-1)})$ **then**
Reupdated $\boldsymbol{B}^{t}$ according to (3.23) with $\tilde{\boldsymbol{B}}^{(t-1)} = \boldsymbol{B}^{(t-1)}$;
10: **if** $f(\boldsymbol{C}^{(t)}) \geq f(\boldsymbol{C}^{(t-1)})$ **then**
Reupdated $\boldsymbol{C}^{(t)}$ according to (3.23) with $\tilde{\boldsymbol{C}}^{(t-1)} = \boldsymbol{C}^{(t-1)}$;
11: **if** stopping criterion is reached **then**
Return $\{\boldsymbol{A}\}, \{\boldsymbol{B}\}, \{\boldsymbol{C}\}$

which can be written in the closed form

$$\boldsymbol{A}^{t} = \max(0, \tilde{\boldsymbol{A}}^{(t-1)} - \tilde{\boldsymbol{G}}_1^{(t-1)}/L_1^{(t-1)}) \tag{3.23}$$

Following the steps above, we can update $\boldsymbol{B}$ and $\boldsymbol{C}$ in the same manner. Xu [52] also mentioned that the extrapolation operation (3.18) for updating (3.20) significantly accelerates the convergence of APG in their applications.

At the end of the iteration t, we check whether $f(\boldsymbol{A}^{(t)}) \geq f(\boldsymbol{A}^{(t-1)})$. If so, we reupdate $\boldsymbol{A}^{(t)}$ by (3.23) with $\tilde{\boldsymbol{A}}^{(t-1)} = \boldsymbol{A}^{(t-1)}$

We show the complete APG [52] algorithm in Algorithm 4:

Let us take a look at per-iteration complexity of APG [52]. In line 3, computing $L_{(n)}^{(t-1)}$ costs $R^2(2\times J\times K-1)+2\times R^2$. Computing $w_{(n)}^{(t-1)}$ costs 3 flops. In line 4, it costs $3\times I\times R$ flops. In line 5, it costs $R^2(2\times J\times K-1)+I\times R(2R-1)+I\times R(2\times J\times K-1)+I\times R$ to compute $G_{(n)}^{(t-1)}$. And updating $A_{(n)}^{(t)}$ consumes $3\times I\times R$. Here, the most costly operation is still MTTKRP.

Proposition 2 *Let* $\{\boldsymbol{A}^{(t)}\}$ *be the sequence generated by Algorithm 4. Assume that* $\{\boldsymbol{A}^{(t)}\}$ *is bounded and there is a positive constant* l *such that* $l \leq L_n^{(t)}$ *for all* t *and* n. *Then* $\{\boldsymbol{A}^{(t)}\}$ *convergences to a critical point* $\overline{\boldsymbol{A}}$.

This proposition shows that once $\{\boldsymbol{A}^{(t)}\}$ is bounded, $L_{(n)}^{(t)}$ is also bounded. And under the condition that $l \leq L_{(n)}^{(t)}$ for all t and n, $\{\boldsymbol{A}^{(t)}\}$ will converge to a critical point $\overline{\boldsymbol{A}}$. The proof can be found in Theorem 3.1 [52].

APG is flexible with different constraints and regularizations. It can handle probability simplex, sparsity, and so on by using the proximal gradient technique. However, MTTKRP can not be avoided when we do the gradient-type update.

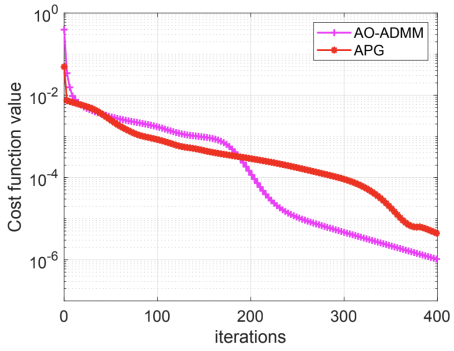


Figure 3.1: AO-ADMM [25] and APG [52] exhibit the same convergence behavior. The tensor under test has a size of $50 \times 50 \times 50$ and the rank is 10. The latent factors are constrained to be non-negative. AO-ADMM and APG consume similar flops when dealing with constrained CPD problems.

In Figure 3.1, we show a simple numerical experiment of the performance by using AO-ADMM and APG. We test a tensor with size $50 \times 50 \times 50$, and the rank is 10. We also set all the latent factors are constrained to be nonnegative. From the figure, we found that AO-ADMM and APG have similar convergence behavior. It also shows that when reaching the same cost function value, AO-ADMM and APG will consume similar flops.

3.3 Stochastic Algorithms

In this section, we will introduce two algorithms: CPRAND [4] and RBS [49] which employ stochastic optimization to reduce per-iteration complexity. CPRAND [4] leverages fiber sampling, which is similar to our sampling approach. RBS [49] uses subtensor sampling, which is also a promising method.

3.3.1 RBS

Consider the optimization problem as below:

$$\min_{\{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}\}} \; f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}) \tag{3.24}$$

in which f is a decomposable function:

$$f = \frac{1}{2}\|\boldsymbol{\mathcal{X}} - \sum_{r=1}^{R} \boldsymbol{A}(:,\ r) \circ \boldsymbol{B}(:,r) \circ \boldsymbol{C}(:,r)\|_F^2 \tag{3.25}$$

Vervliet [49] proposed that if we define tensor block by its index sets $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$, then the gradient of f is only nonzero for the variables belonging to the block index set.

As we can see in Algorithm 5, a random subtensor is sampled in each iteration. And variables in this subtensor will be updated in each update process. Vervliet also mentioned that instead of using an exact Hessian matrix in updating rule (Line 10, 11 and 12), the algorithm takes advantage of relatively cheap approximation of the Hessian matrix. The details for using the approximation Hessian matrix can be found in Part 3, section B in [49]. Vervliet uses Δ_t as his step size which is decreased in order to achieve convergence and to improve the accuracy.

Vervliet [49] proposed that we randomly choose a sample subtensor $\boldsymbol{\mathcal{X}}_{\text{sub}}$ by selecting a random subset $\mathcal{B}_n$ of B_n indices from $\mathcal{I}_n = \{1, \cdots, I_n\}$. And we will select N new subtensors in each iteration. We define $Q_n = I_n/B_n$ be the number of subtensors per dimension and will permute the elements within index set after every Q_n iterations. Following this sample strategy, we will continuously select indices $\mathcal{B}_n$ when all the variables in the dimension are updated. The block size and random selection play a significant role in the algorithm since they will impact the robustness, the total computation time and

Algorithm 5 Randomized block sampling CPD [49]

1: **Initialize**: factor matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$
2: **for** $n = 1, ..., N$ **do**
3: Randomly generate sample indices
4: $\mathcal{B}_n \subseteq \mathcal{I}_n = \{1, \cdots, I_n\}, n = 1, 2, 3$
5: Let $\boldsymbol{\mathcal{X}}_{\text{sub}} = \boldsymbol{\mathcal{X}}(\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3)$
6: $\boldsymbol{A}_{\text{sub}} = \boldsymbol{A}^t(\mathcal{B}_1, :)$
7: $\boldsymbol{B}_{\text{sub}} = \boldsymbol{B}^t(\mathcal{B}_2, :)$
8: $\boldsymbol{C}_{\text{sub}} = \boldsymbol{C}^t(\mathcal{B}_3, :)$
9: $\{\boldsymbol{A}_{\text{sub}}\} \leftarrow$ update $(\boldsymbol{\mathcal{X}}_{\text{sub}}, \{\boldsymbol{A}_{\text{sub}}\}, \Delta_t)$
10: $\{\boldsymbol{B}_{\text{sub}}\} \leftarrow$ update $(\boldsymbol{\mathcal{X}}_{\text{sub}}, \{\boldsymbol{B}_{\text{sub}}\}, \Delta_t)$
11: $\{\boldsymbol{C}_{\text{sub}}\} \leftarrow$ update $(\boldsymbol{\mathcal{X}}_{\text{sub}}, \{\boldsymbol{C}_{\text{sub}}\}, \Delta_t)$
12: Set $\boldsymbol{A}^{t+1} = \boldsymbol{A}^t$ and $\boldsymbol{A}^{t+1}(\mathcal{B}_n, :) = \boldsymbol{A}_{\text{sub}}$
13: $\boldsymbol{B}^{t+1} = \boldsymbol{B}^t$ and $\boldsymbol{B}^{t+1}(\mathcal{B}_n, :) = \boldsymbol{B}_{\text{sub}}$
14: $\boldsymbol{C}^{t+1} = \boldsymbol{C}^t$ and $\boldsymbol{C}^{t+1}(\mathcal{B}_n, :) = \boldsymbol{C}_{\text{sub}}$
15: $t \leftarrow t + 1$
16: **if** stopping criterion is reached **then**
Return $\{\boldsymbol{A}\}, \{\boldsymbol{B}\}, \{\boldsymbol{C}\}$

the result accuracy. We can apply any CPD algorithm to compute the update in Algorithm 5. Vervliet applies two different ways to do the update procedure. The first one is alternating least squares (ALS) and the second one is nonlinear least squares (NLS) [49].

Let's take a look at per-iteration complexity of RBS [49]. In line 4, it costs $|\mathcal{F}_n| \times N$ flops. In line 5, it costs $|\mathcal{F}_n|$ flops. In line 10, I get from Part 4.2 of [47] that it will cost $R \times |\mathcal{F}_n|^3 + N \times R \times |\mathcal{F}_n|^3 + N \times R^2 \times |\mathcal{F}_n|^2 + 2 * (N \times R^2 \times |\mathcal{F}_n| + N \times R^2 \times |\mathcal{F}_n|^2)$ flops for updating factor matrix. As we can see from this flop analysis, per-iteration complexity has been reduced to a relatively small number which is related to the size of our sampled subtensor.

RBS [49] is very scalable and many existing algorithms can be applied to the sampled subtensor, for example, ALS (alternating least squares). However, RBS is not flexible with constraints. Since constraints on the column of $\boldsymbol{A}_{(n)}$ or the entire $\boldsymbol{A}_{(n)}$ can not be handled, Vervliet [49] adopts the sample strategy as $\min(R + 20, 2 \times R)$ in RBS paper. Once we meet the case that R is larger than $\min(I, J, K)$, the sample strategy here does not work.

In Figure 3.3, we show the impact of cost function value with different subtensor

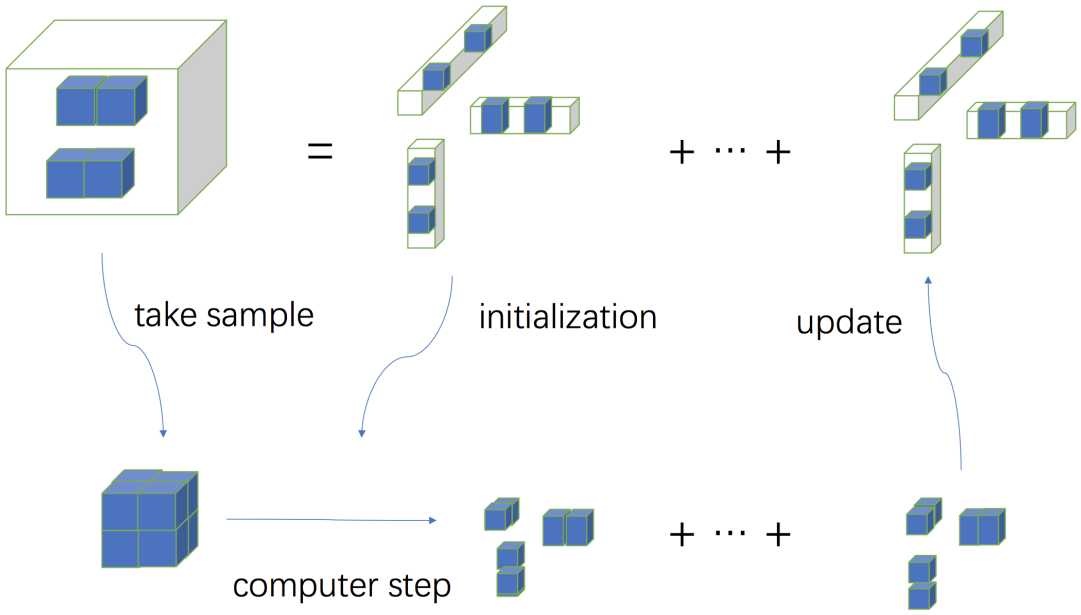


Figure 3.2: Randomized block sampling CPD

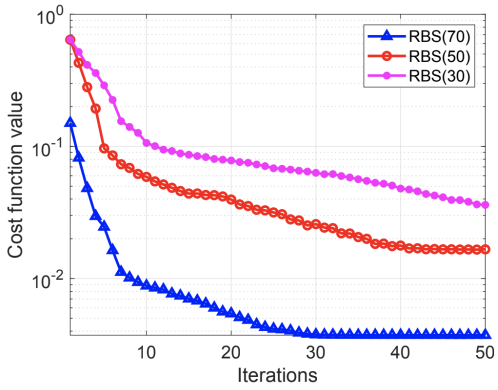


Figure 3.3: RBS [49] shows different convergence behavior under different size of sampling blocks. The tensor under test has a size of $100 \times 100 \times 100$ and the rank is 50. When the subtensor is not identifiable, RBS will converge with a relatively high cost function value.

sizes. We did this experiment by using a tensor with size of $100 \times 100 \times 100$ and the

rank is 50. By Vervliet's [49] default, the sampled subtensor size should be 70. We also tried different subtensors with size equal to 30 and 50. Figure 3.3 shows that if the size of subtensor is too small, RBS may take the risk of losing identifiability.

3.3.2 CPRAND

Consider the CPD problem as below:

$$\underset{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}}{\text{minimize}} \left\| \boldsymbol{\mathcal{X}} - \sum_{r=1}^{R} \boldsymbol{A}(:,\ r) \circ \boldsymbol{B}(:,r) \circ \boldsymbol{C}(:,r) \right\|_F^2 \tag{3.26}$$

When applying stochastic optimization methods to the problem above, Battaglino [4] proposed that we can uniformly sample rows from $\boldsymbol{H}_{(n)}$ (n is the index of different modes) with its corresponding rows from $\boldsymbol{X}_{(n)}^T$. Here, we set $|\mathcal{F}_n|$ as the number of sampling fibers that we need for CPRAND algorithm. In order to make CPRAND algorithm more accurate and robust, Battaglino proposed that we can set $|\mathcal{F}_n| > \max\{I, J, K, R\}$. We get the samples from $\{1, \cdots, JK\}$, $\{1, \cdots, IK\}$, $\{1, \cdots, IJ\}$ and denote it as $|\mathcal{F}_n|$, so we have $|\mathcal{F}_n|$ as our mini batch. Every row in $\boldsymbol{H}_{(n)}$ and $\boldsymbol{X}_{(n)}^T$ will have the same chance to be selected and each time we will uniformly select from all the rows. The samples we have in different iterations are independent.

Consider the optimization problem:

$$\boldsymbol{A} \leftarrow \arg\min_{\boldsymbol{A}} ||\boldsymbol{X}_{(1)} - (\boldsymbol{C} \odot \boldsymbol{B})\boldsymbol{A}^T|| \tag{3.27}$$

Battaglino also mentioned that it is obvious that forming the full Khatri-Rao product $\boldsymbol{H}_{(1)} = \boldsymbol{C} \odot \boldsymbol{B}$ will consume a lot of computing resources, therefore we would like to simplify the computing process without actually forming $\boldsymbol{H}_{(1)}$. When we want to sample the jth row of $\boldsymbol{H}_{(1)}$, we can find the corresponding row indices of other factor matrix as (j, k). Finally, we find that the jth row of $\boldsymbol{H}_{(1)}$ is the Hadamard product of the appropriate rows of the factor matrices. And we have the equation as below:

$$\boldsymbol{H}_{(1)}(j,:) = \boldsymbol{B}(j,:) * \boldsymbol{C}(k,:). \tag{3.28}$$

Since every time in updating procedure we will not just choose one row for samples,

every $\mathcal{F}_n$ here is a vector with different choice of j-th column. We will get $\boldsymbol{H}(\mathcal{F}_n, :)$ with size $|\mathcal{F}_n| \times R$. Under the same strategy, we can construct $\boldsymbol{X}^T(\mathcal{F}_n, :)$ in this way. Therefore, we have avoided the process to directly construct $\boldsymbol{X}^T(\mathcal{F}_n, :)$ and $\boldsymbol{H}^T(\mathcal{F}_n, :)$.

Battaglio named the randomized version algorithm as CPRAND [4] which is shown in Algorithm 7. CPRAND has very simple updates and it is well-aligned with the ALS [11] structure. However, CPRAND can not handle different constraints and regularizations. And it needs to sample $|\mathcal{F}_n| \geq R$ fibers at each iteration, in which R might be large in CPD problems.

Let's take a look at per iteration complexity of CPRAND [4]. In line 5, it costs $|\mathcal{F}_n| \times N$ flops. In line 6, it costs $|\mathcal{F}_n| \times R \times (N-1)$ flops. In line 7, it costs 0 flops here. In line 8, it costs $2 \times |\mathcal{F}_n| \times R^2 + 2 \times |\mathcal{F}_n| \times R \times I_1 + R^2 \times I_1$ flops. As we can see from the analysis, CPRAND [4] has reduced per iteration complexity by fiber sampling strategy. Under this sampling fibers strategy, we can save a lot of computing resources.

Table 3.1: Performance of the algorithms under various I's, $R = 100$; all the algorithms are stopped after computing 30 MTTKRPs.

Algorithm	Metric	I		
		200	300	400
`AdaCPD`	MSE	**0.0121**	**0.0016**	$\mathbf{1.0068 \times 10^{-4}}$
`CPRAND`	MSE	0.0459	0.0056	0.0025
`AdaCPD`	Cost	**0.0058**	$\mathbf{4.8050 \times 10^{-4}}$	$\mathbf{3.5958 \times 10^{-5}}$
`CPRAND`	Cost	0.0136	0.0018	0.0011

CPRAND [4] has reduced per iteration complexity compared with AO-ADMM [25] and APG [52] when dealing with large scale CPD problems. Under the setting of large scale tensor decomposition, we also focus on the accuracy of our applied algorithm. In Table 3.1, we compare CPRAND [4] with our proposed AdaCPD [18] which also uses fiber sampling strategy. We did the experiments with a fixed tensor rank $R = 100$ and three different tensor size $(200, 300, 400)$. The results in Table 3.1 show that CPRAND will have a higher cost function value when doing same MTTKRP with AdaCPD. It means that if we want to find a high accuracy algorithm in our practical problems, CPRAND may be not suitable for that scenario.

Algorithm 6 Sampled Khatri-Rao Product (SKR) [4]

1: **Initialize**: factor matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$
2: $\boldsymbol{H}(\mathcal{F}_n, :) \leftarrow \mathbf{1} \in \mathbb{R}^{|\mathcal{F}_n| \times R}$;
3: **for** $m = 1, \cdots, n-1, n+1, \cdots, N$ **do**
4: $\quad \boldsymbol{A}(\mathcal{F}_n, :) \leftarrow \boldsymbol{A}(\texttt{index}(:, m), :)$
5: $\quad \boldsymbol{H}(\mathcal{F}_n, :) \leftarrow \boldsymbol{H}(\mathcal{F}_n, :) * \boldsymbol{A}((\mathcal{F}_n, :))$
6: **return**: $\boldsymbol{H}(\mathcal{F}_n, :)$

3.4 Summary

In this chapter, we have introduced five algorithms for solving CPD problems: ALS [11], AO-ADMM [25], APG [52], CPRAND [4], and RBS [49]. ALS [11] is the classical algorithm people use to handle the CPD problem by fixing all other variables instead of one and does the update in this manner. AO-ADMM [25] and APG [52] focus on constrained CPD problems which employ the ALS [11] framework and solve subproblems by constrained optimization techniques. However, MTTKRP can not be avoided in these two algorithms. CPRAND [4] and RBS [49] leverage stochastic optimization, for example, subtensor sampling and fiber sampling, which reduce per-iteration complexity. However, both of them cannot handle constrained CPD problems.

Algorithm 7 CPRAND [4]

Initialize: factor matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$
for $n = 1, 2, 3$ **do**
    Define sampling operator $\mathcal{F}_n \in \mathbb{R}^{\times \prod_{m \neq n} I_m}$

    $\boldsymbol{H}_{(1)}(\mathcal{F}_n, :) \leftarrow \text{SKR}(\mathcal{F}_n, \boldsymbol{C}, \boldsymbol{B})$

    $\boldsymbol{H}_{(2)}(\mathcal{F}_n, :) \leftarrow \text{SKR}(\mathcal{F}_n, \boldsymbol{C}, \boldsymbol{A})$

    $\boldsymbol{H}_{(3)}(\mathcal{F}_n, :) \leftarrow \text{SKR}(\mathcal{F}_n, \boldsymbol{B}, \boldsymbol{A})$

    $\boldsymbol{X}^T_{(n)}(\mathcal{F}_n, :) \leftarrow \mathcal{F}_n \boldsymbol{X}^T_{(n)}$

    $\boldsymbol{A} \leftarrow \arg\min_{\boldsymbol{A}} \left\| \boldsymbol{X}_{(1)}(\mathcal{F}_n, :) - \boldsymbol{H}_{(1)}(\mathcal{F}_n, :)\boldsymbol{A}^T \right\|_F^2$

    $\boldsymbol{B} \leftarrow \arg\min_{\boldsymbol{B}} \left\| \boldsymbol{X}_{(2)}(\mathcal{F}_n, :) - \boldsymbol{H}_{(2)}(\mathcal{F}_n, :)\boldsymbol{B}^T \right\|_F^2$

    $\boldsymbol{C} \leftarrow \arg\min_{\boldsymbol{C}} \left\| \boldsymbol{X}_{(3)}(\mathcal{F}_n, :) - \boldsymbol{H}_{(3)}(\mathcal{F}_n, :)\boldsymbol{C}^T \right\|_F^2$

    Normalize columns of $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ and update $\boldsymbol{\lambda}$.
if stopping criterion is reached **then**
    Return $\{\boldsymbol{A}\}, \{\boldsymbol{B}\}, \{\boldsymbol{C}\}$

Chapter 4 Proposed Algorithms

In this thesis, we have proposed a new stochastic optimization strategy for CPD[1]. Our algorithms are based on alternating least squares and fiber sampling. And it is highly favorable that our sampling size which is under our control can be much smaller than the tensor rank. It is the key step for our methods to have a low per-iteration complexity. An amount of constraints and regularizations that are commonly used in signal processing and data mining can be easily handled by our proposed algorithms—which is reminiscent of *stochastic proximal gradient* (SPG) [23, 51]. Meanwhile, we also provide convergence proof to strengthen our algorithms in this thesis.

4.1 Basic Idea: Unconstrained Case

Consider the optimization problem as below:

$$\min_{\{\boldsymbol{A}_{(n)}\}_{n=1}^{N}} \quad f(\boldsymbol{A}_{(1)}, \ldots, \boldsymbol{A}_{(N)}). \tag{4.1}$$

Inspired by alternating least squares and tensor fiber sampling strategies, we proposed our stochastic optimization algorithms. In detail, at each iteration, we sample a set of mode-n fibers for a certain n as the method in [4] does. However, instead of exactly solving the least squares subproblems (3.3) for all the modes following a Gauss-Seidel manner in each iteration, we update $\boldsymbol{A}_{(n)}$ using a doubly stochastic procedure. To be more precise, at iteration f, we first randomly sample a mode index $n \in \{1, ..., N\}$. Then, we randomly sample a set of mode-n fibers that are indexed by $\mathcal{F}_n \subset \{1, ..., J_n\}$ (Figure 4.1). Let $\boldsymbol{G}^{(r)} \in \mathbb{R}^{I_1 \times R} \times \cdots \times \mathbb{R}^{I_N \times R}$ be a collection of matrices, representing

[1]This work is based on two papers. The first one is from arXiv [19], the second one is from ICASSP 2019 [18]

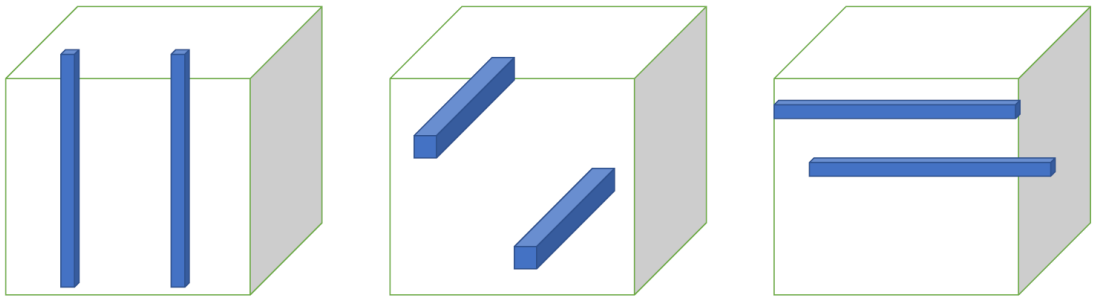


Figure 4.1: Sampled fibers of a third-order tensor

the stochastic gradient as:

$$\begin{aligned} \boldsymbol{G}_{(n)}^{(t)} &= \frac{1}{|\mathcal{F}_n|}\Big(\boldsymbol{A}_{(n)}^{(t)}\boldsymbol{H}_{(n)}^T(\mathcal{F}_n)\boldsymbol{H}_{(n)}(\mathcal{F}_n) - \boldsymbol{X}_{(n)}^T(\mathcal{F}_n)\boldsymbol{H}_{(n)}(\mathcal{F}_n)\Big) \\ \boldsymbol{G}_{(n')}^{(t)} &= \mathbf{0}, \quad n' \neq n, \end{aligned} \tag{4.2}$$

where $\boldsymbol{G}_{(n)}^{(t)}$ denotes the nth block of $\boldsymbol{G}^{(t)}$, and we use the simplified notations

$$\boldsymbol{X}_{(n)}(\mathcal{F}_n) = \boldsymbol{X}_{(n)}(\mathcal{F}_n, :), \quad \boldsymbol{H}_{(n)}(\mathcal{F}_n) = \boldsymbol{H}_{(n)}(\mathcal{F}_n, :).$$

We update the latent variables by following the step below:

$$\boldsymbol{A}_{(n)}^{(t+1)} \leftarrow \boldsymbol{A}_{(n)}^{(t)} - \alpha^{(t)}\boldsymbol{G}_{(n)}^{(t)}, \quad n = 1, ..., N. \tag{4.3}$$

Note that $\boldsymbol{G}_{(n)}^{(t)}$ is nothing but a stochastic approximation which we apply it to approximate the full gradient of Problem (4.1) with respect to the chosen mode-n variable $\boldsymbol{A}_{(n)}$. And the update process is similar to the classical stochastic gradient descent algorithm which we choose a minibatch size $|\mathcal{F}_n|$) for solving the problem in (3.3).

Our proposed method takes advantage of low per-iteration complexity since we have avoided the most resource-consuming update $\boldsymbol{H}_{(n)}^T\boldsymbol{X}_{(n)}$ in algorithms such as those in [25, 52]. We construct $\boldsymbol{X}_{(n)}^T(\mathcal{F}_n, :)\boldsymbol{H}_{(n)}(\mathcal{F}_n, :)$ only by consuming $\mathcal{O}(|\mathcal{F}_n|FI_n)$ flops which we can control the size of $|\mathcal{F}_n|$. One observation of our algorithm is that we randomly sample

a block at the beginning which is not similar with the classical ALS-type algorithms updating the block variables $\boldsymbol{A}_{(n)}$ cyclically. And this small procedure helps us a lot in analyzing convergence properties.

4.2 Constrained and Regularized Case

When it comes to real life applications, we always need to take regularizations or constraints on $\boldsymbol{A}_{(n)}$'s into conisderation. Since our proposed algorithm updates the entire $\boldsymbol{A}_{(n)}$ in each iteration, it is natural for us to include different types of widely used constraints and regularizations. Compared with the entry sampling based approaches in [7, 49], our proposed algorithm is more flexible for handling different situations. And we can extend our unconstrained cases to constrained cases by using the formula shown below:

$$\begin{aligned} \min_{\{\boldsymbol{A}_{(n)}\}_{n=1}^{N}} \quad & f(\boldsymbol{\theta}) + \sum_{n=1}^{N} h_n(\boldsymbol{A}_{(n)}) \\ \text{subject to} \quad & \boldsymbol{A}_{(n)} \in \mathcal{A}_n, \end{aligned} \tag{4.4}$$

where $f(\boldsymbol{\theta})$ is the objective function of (4.1), $h_n(\boldsymbol{A}_{(n)})$ denotes a structure-promoting regularizer on $\boldsymbol{A}_{(n)}$. Note that $\boldsymbol{A}_{(n)} \in \mathcal{A}_n$ can also be written as a regularization $h_n\left(\boldsymbol{A}_{(n)}\right)$ if $h_n(\cdot)$ is defined as the indicator function of set $\mathcal{A}_n$, i.e.,

$$h_n(\boldsymbol{A}) = \mathcal{I}(\mathcal{A}_n) = \begin{cases} 0, & \boldsymbol{A} \in \mathcal{A}_n \\ \infty, & \text{o.w.} \end{cases}$$

where $\mathcal{I}(\mathcal{X})$ denotes the indicator function of the set $\mathcal{X}$. Using the same fiber sampling strategy as in the previous subsection, we update $\boldsymbol{A}_{(n)}$ by

$$\boldsymbol{A}_{(n)}^{(t+1)} \leftarrow \arg\min_{\boldsymbol{A}_{(n)}} \ \left\|\boldsymbol{A}_{(n)} - \left(\boldsymbol{A}_{(n)}^{(t)} - \alpha^{(t)}\boldsymbol{G}_{(n)}^{(t)}\right)\right\|_F^2 + h_n\left(\boldsymbol{A}_{(n)}\right) \tag{4.5a}$$

$$\boldsymbol{A}_{(n')}^{(t+1)} \leftarrow \boldsymbol{A}_{(n')}^{(t)}, \quad n' \neq n \tag{4.5b}$$

Problem (4.5a) is also known as the proximal operator of $h_n(\cdot)$, which is often denoted as

$$\boldsymbol{A}_{(n)}^{(t+1)} \rightarrow \mathsf{Prox}_{h_n}\left(\boldsymbol{A}_{(n)}^{(t)} - \alpha^{(t)}\boldsymbol{G}_{(n)}^{(t)}\right). \tag{4.6}$$

Algorithm 8 BrasCPD

Input: N-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$; rank R; initialization $\{\boldsymbol{A}_{(n)}^{(0)}\}$, step size $\{\alpha^{(t)}\}_{t=0,\ldots}$
$t \leftarrow 0$;
repeat
    uniformly sample n from $\{1, \ldots, N\}$, then sample $\mathcal{F}_n$ from $\{1, \ldots, J_n\}$ with $|\mathcal{F}_n| = B$
    form the stochastic gradient $\boldsymbol{G}^{(t)} \leftarrow$ (4.2)
    update $\boldsymbol{A}_{(n)}^{(t+1)} \leftarrow$ (4.5a), $\boldsymbol{A}_{(n')}^{(t+1)} \leftarrow \boldsymbol{A}_{(n')}^{(t)}$ for $n' \neq n$
    $t \leftarrow t + 1$
until stopping criterion is reached
Output: $\{\boldsymbol{A}_{(n)}^{(t)}\}_{n=1}^{N}$

Many $h_n(\cdot)$'s admit simple closed-form solutions for their respective proximal operators, e.g., when $h_n(\cdot)$ is the indicator function of the nonnegative orthant and $h_n(\cdot) = \|\cdot\|_1$; see Table 4.1 and more details in [25, 36]. The complexity of computing (4.6) is often similar to that of the plain update in (4.3), and thus is also computationally efficient. An overview of the proposed algorithm can be found in algorithm 8, which we name ***B**lock-**Ra**ndomized **S**GD for **CPD*** (`BrasCPD`).

Table 4.1: Proximal/projection operator of some frequently used regularizations and constraints.

$h(\cdot)$	prox./proj. solution	complexity
$\|\cdot\|_1$	soft-thresholding	$\mathcal{O}(d)$
$\|\cdot\|_2$	re-scale	$\mathcal{O}(d)$
$\|\cdot\|_{2,1}$	block soft-thresholding	$\mathcal{O}(d)$
$\|\cdot\|_0$	hard-thresholding	$\mathcal{O}(d)$
$\mathcal{I}(\Delta)$	randomized pivot search [17]	$\mathcal{O}(d)$ in expectation
$\mathcal{I}(\mathbb{R}_+)$	max	$\mathcal{O}(d)$
monotonic	monotone regression [30]	$\mathcal{O}(d)$
unimodal	unimodal regression [10]	$\mathcal{O}(d^2)$

†In the table, d is the number of optimization variables.

4.3 Convergence Properties

Note that `BrasCPD` does not fall into any known framework of block stochastic gradient optimization, and thus its convergence properties are not immediately clear. Two most relevant works from the optimization literature are [52] and [50]. However, the work in [52] considers Gauss-Seidel type block SGD (i.e., cyclically updating the blocks), instead of the block-randomized version as `BrasCPD` uses. The work in [50] considers block-randomized SGD, but only for the convex case. In addition, many assumptions made in [50,52] for their respective generic optimization problems are not easily satisfied by our CPD problem. In this section, we offer tailored convergence analyses for `BrasCPD`.

4.3.1 Unconstrained Case

To proceed, we will use the following assumptions:

Assumption 1 *The stepsize schedule follows the Robbins-Monro rule [40]:*

$$\sum_{t=0}^{\infty} \alpha^{(t)} = \infty, \quad \sum_{t=0}^{\infty} (\alpha^{(t)})^2 < \infty.$$

Assumption 2 *The updates $\boldsymbol{A}_{(n)}^{(t)}$ are bounded for all n, t.*

Assumption 1 is a principle for stepsize scheduling, which is commonly used in stochastic approximation. Assumption 2 is a working assumption that we make to simplify the analysis. It is considered a relatively strong assumption, since it is hard to check or guarantee. Nevertheless, unbounded iterates rarely happen in practice, if the stepsize is well controlled.

There are also an array of problem structures that are useful for studying convergence of the algorithm.

Fact 1 *The LS fitting part in the objective function* (4.4) *(i.e.$f(\boldsymbol{\theta})$) satisfies*

$$\begin{aligned} f(\boldsymbol{\theta}) \leq & f(\bar{\boldsymbol{\theta}}) + \langle \nabla_{\boldsymbol{A}_{(n)}} f(\bar{\boldsymbol{\theta}}), \boldsymbol{A} - \bar{\boldsymbol{A}}_{(n)} \rangle \\ & + \frac{\bar{L}_{(n)}}{2} \|\boldsymbol{A} - \bar{\boldsymbol{A}}_{(n)}\|_F^2, \end{aligned} \tag{4.7}$$

where $\bar{L}_{(n)} \geq \lambda_{\max}(\bar{\boldsymbol{H}}_{(n)}^{\top}\bar{\boldsymbol{H}}_{(n)})$, $\bar{\boldsymbol{\theta}}$ *is a feasible point, and* $\bar{\boldsymbol{A}}_{(n)}$ *and* $\bar{\boldsymbol{H}}_{(n)}$ *are extracted/constructed from* $\bar{\boldsymbol{\theta}}$ *following the respective definitions.*

Eq. (4.7) holds because the objective function $f(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{A}_{(n)}$ is a plain least squares fitting criterion, which is known to have a Lipschitz continuous gradient—and the smallest Lipschitz constant is $\lambda_{\max}(\bar{\boldsymbol{H}}_{(n)}^{\top}\bar{\boldsymbol{H}}_{(n)})$.

The second fact is instrumental in proving convergence of the algorithm:

Fact 2 *Denote* $\xi_{(t)}$ *and* $\zeta^{(t)}$ *as the random variables that are responsible for selecting the mode and fibers in iteration* t*, respectively. Also denote* $\mathcal{B}^{(t)} = \{\xi^{(1)}, \zeta^{(1)}, \ldots, \xi^{(t-1)}, \zeta^{(t-1)}\}$*, i.e., the filtration up to* t*. The block-wise stochastic gradient constructed in* (4.2) *is an unbiased estimation for the full gradient w.r.t.* $\boldsymbol{A}_{(\xi^{(t)})}$*, i.e.,*

$$\mathbb{E}_{\zeta^{(t)}}\left[\boldsymbol{G}_{(\xi^{(t)})}^{(t)} \mid \mathcal{B}^{(t)}, \xi^{(t)}\right] = \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} t(\boldsymbol{\theta}^{(t)}), \tag{4.8}$$

if $\zeta^{(t)}$ *admits the following probability mass function (PMF):*

$$\mathsf{Pr}(\zeta^{(t)} = i) = \frac{1}{J_n}, \ \forall i \in \{1, \ldots, J_n\}. \tag{4.9}$$

The proof of the above is straightforward and thus skipped. The Fact says that even if our block stochastic gradient is not exactly an unbiased estimation for $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$, it is an unbiased estimation for the "block gradient" $\nabla_{\boldsymbol{A}_{(n)}} f(\boldsymbol{\theta}^{(t)})$. This fact will prove quite handy in establishing convergence. In fact, the two-level sampling strategy (i.e., block sampling and fiber sampling, respectively), makes the gradient estimation w.r.t. $\boldsymbol{\theta}$ unbiased up to a scaling factor (see Appendix A.1). This connection intuitively suggests that the proposed algorithm should behave similarly as an ordinary single-block stochastic gradient descent algorithm.

We first have the following convergence property:

Proposition 3 *Consider the case where* $h_n(\cdot) = 0$ *and Assumptions 1-2 hold. Then, the solution sequence produced by* `BrasCPD` *satisfies:*

$$\lim_{t\to\infty} \inf \mathbb{E}\left[\left\|\nabla_{\boldsymbol{A}_{(n)}} t\left(\boldsymbol{A}_{(1)}^{(t)}, \ldots, \boldsymbol{A}_{(N)}^{(t)}\right)\right\|^2\right] = 0, \ \forall n.$$

The proof is relegated to Appendix. The above proposition implies that there exists a subsequence of the solution sequence that converges to a stationary point in expectation. We should mention that the SGD/stochastic proximal gradient type update and the block sampling step are essential for establishing convergence—and using the exact solution to sketched version problem in [4] may not have such convergence properties.

4.3.2 Constrained/Regularized Case

To understand convergence of the proximal gradient version with $h_n(\cdot) \neq 0$, denote $\Phi(\boldsymbol{\theta}) = f(\boldsymbol{\theta}) + \sum_{n=1}^{N} h_n(\boldsymbol{\theta})$ as the objective function. Our optimality condition amounts to $\boldsymbol{P}_{(n)}^{(t)} = \mathbf{0}$, $\forall\ n$, where

$$\boldsymbol{P}_{(n)}^{(t)} = \frac{1}{\alpha^{(t)}} \left(\boldsymbol{A}_{(n)}^{(t)} - \mathsf{Prox}_{h_n} \left(\boldsymbol{A}_{(n)}^{(t)} - \alpha^{(t)} \nabla_{\boldsymbol{A}_{(n)}} f(\boldsymbol{\theta}^{(t)}) \right) \right);$$

i.e., the optimality condition is satisfied in a blockwise fashion [39, 53]. Hence, our goal of this section is to show that $\mathbb{E}[\|\boldsymbol{P}_{(n)}^{(r)}\|^2]$ for all n vanishes when t grows. We will use the following assumption:

Assumption 3 *There exists a sequence $\sigma^{(t)}$ for $t = 0, 1, \ldots$, such that*

$$\mathbb{E}_{\zeta^{(t)}} \left[\left\| \boldsymbol{G}_{(\xi^{(t)})}^{(t)} - \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}) \right\|^2 \ \middle|\ \mathcal{B}^{(t)}, \xi^{(t)} \right] \leq (\sigma^{(t)})^2,$$

and

$$\sum_{t=0}^{\infty} (\sigma^{(t)})^2 < \infty. \tag{4.10}$$

We show that `BrasCPD` produces a convergent solution sequence in the following proposition:

Proposition 4 *Assume that Assumptions 1-3 hold. Also assume that $h_n(\cdot)$ is a convex function. Then, the solution sequence produced by `BrasCPD` satisfies*

$$\lim_{t \to \infty} \inf \mathbb{E} \left[\left\| \boldsymbol{P}_{(n)}^{(t)} \right\|^2 \right] = 0,\ \forall n.$$

Remark 1 *Note that the convergence result in Proposition 4 inherits one possible draw-*

back from single-block stochastic proximal gradient algorithms for nonsmooth nonconvex optimization. To be specific, the relatively strong assumption in (4.10) *needs to be assumed for ensuring convergence. Assumption 3 essentially means that the variance of the gradient estimation error* $\boldsymbol{\delta}^{(t)}_{(\xi^{(t)})} = \boldsymbol{G}^{(t)}_{(\xi^{(t)})} - \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)})$ *decreases and converges to zero. This is not entirely trivial. One way to fulfill this assumption is to increase the minibatch size along the iterations, e.g., by setting [22, 53]:*

$$|\mathcal{F}_n^{(t)}| = \mathcal{O}(\lceil t^{1+\epsilon} \rceil), \quad \forall \epsilon > 0.$$

Then, one can see that $(\sigma^{(t)})^2 = \mathcal{O}(\frac{1}{\lceil t^{1+\epsilon} \rceil})$, *so that* $\sum_{t=0}^{\infty} (\sigma^{(t)})^2 < \infty$. *Another popular way for achieving* (4.10) *is to use some advanced variance reduction techniques such as SVRG [51]—which may go beyond the scope of this paper and thus is left out of the discussion. Also notice that as the convergence analysis is pessimistic, in practice constant minibatch size works fairly well—as we will see soon.*

4.4 An Adaptive Stepsize Scheme

One may have noticed that the convergence theories in Propositions 3-4 do not specify the sequence $\alpha^{(r)}$ except two constraints as in Assumption 1. This oftentimes gives rise to agonizing tuning experience for practitioners when implementing stochastic algorithms.

Recently, a series of algorithms were proposed in the machine learning community for adaptive stepsize scheduling when training deep neural networks [15, 28, 54]. Most of these works are variants of the `Adagrad` algorithm [16]. The insight of `Adagrad` can be understood as follows: If one optimization variable has been heavily updated before, then it is given a smaller stepsize for the current iteration (and a larger stepsize otherwise). This way, all the optimization variables can be updated in a balanced manner. `Adagrad` was proposed for single-block algorithms, and this simple strategy also admits many provable benefits under the context of convex optimization [16]. For our multi-block nonconvex problem, we extend the idea and propose the following updating rule: In

iteration t, if $\xi^{(t)} = n$, then, for all $i \in \{1, \ldots, I_n\}$ and all $t \in \{1, \ldots, T\}$, we have

$$[\boldsymbol{\eta}_{(n)}^{(t)}]_{i,t} \leftarrow \frac{\eta}{\left(b + \sum_{p=0}^{t-1} [\boldsymbol{G}_{(n)}^{(p)}]_{i,t}^2\right)^{1/2+\epsilon}}, \tag{4.11a}$$

$$\boldsymbol{A}_{(n)}^{(t+1)} \leftarrow \boldsymbol{A}_{(n)}^{(t)} - \boldsymbol{\eta}_{(n)}^{(t)} * \boldsymbol{G}_{(n)}^{(t)}, \tag{4.11b}$$

$$\boldsymbol{A}_{(n')}^{(t+1)} \leftarrow \boldsymbol{A}_{(n')}^{(t)}, \tag{4.11c}$$

where $\eta, b, \epsilon > 0$. The `Adagrad` version of block-randomized CPD algorithm is very simple to implement. The algorithm is summarized in Algorithm 9, which is named `AdaCPD`.

As one will soon see, such a simple stepsize strategy is very robust to a large number of scenarios under test—i.e., in most of the cases, `AdaCPD` performs well without tuning the stepsize schedule. In addition, the `AdaCPD` algorithm works well for both the constrained and unconstrained case.

Proving convergence for nonconvex `Adagrad`-like algorithms is quite challenging [12, 31]. In this work, we show that the following holds:

Proposition 5 *Assume $h_n(\cdot) = 0$ for all n, and that $\mathsf{Pr}(\xi^{(t)} = n) > 0$ for all t and n. Under the Assumptions 1-2, the solution sequence produced by `AdaCPD` satisfies*

$$\mathsf{Pr}\left(\lim_{t\to\infty} \|\nabla_{\boldsymbol{A}_{(n)}} f(\boldsymbol{\theta}^{(t)})\|^2 = 0\right) = 1.$$

Proposition 5 asserts that the algorithm converges almost surely. The proof is relegated to Appendix A. Our proof extends the idea from a recent paper [31] that focuses on using `Adagrad` for solving single-block nonconvex problems. As mentioned, our two-level sampling strategy makes our algorithm very similar to single-block SGD with a scaled gradient estimation (cf. Appendix A.1), and thus with careful modifications the key proof techniques in [31] goes through. Nevertheless, we detail the proof for being self-containing.

4.5 Summary

In this chapter, we have proposed two algorithms: BrasCPD and AdaCPD. Both of them have lightweight updates and circumvents MTTKRP operations. Meanwhile, they can

Algorithm 9 AdaCPD

Input: N-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$; rank R; sample size B , initialization $\{\boldsymbol{A}_{(n)}^{(0)}\}$
$t \leftarrow 0$;
repeat
    uniformly sample n from $\{1, \ldots, N\}$, then sample $\mathcal{F}_n$ from $\{1, \ldots, J_n\}$ with $|\mathcal{F}_n| = B$
    form the stochastic gradient $\boldsymbol{G}^{(t)} \leftarrow$ (4.2)
    determine the step size $\boldsymbol{\eta}_{(n)}^{(t)} \leftarrow$ (4.11a)
    update $\boldsymbol{A}_{(n)}^{(t+1)} \leftarrow$ (4.11b), $\boldsymbol{A}_{(n')}^{(t+1)} \leftarrow \boldsymbol{A}_{(n')}^{(t)}$ for $n' \neq n$
    $t \leftarrow t + 1$
until stopping criterion is reached
Output: $\{\boldsymbol{A}_{(n)}^{(t)}\}_{n=1}^{N}$

easily handle many commonly used constraints and regularizations. More important, we have showed the proof of convergence for the two algorithms in which some stochastic optimization method may be unclear.

Chapter 5 Experiments

In this section, we use simulations and real-data experiments to showcase the effectiveness of the proposed algorithm.

5.1 Synthetic Data Simulations

5.1.1 Data Generation

Throughout this subsection, we use synthetic third-order tensors (i.e., $N = 3$) whose latent factors are drawn from i.i.d. uniform distribution between 0 and 1—unless otherwise specified. This way, large and *dense* tensors can be created. For simplicity, we set $I_n = I$ for all n and test the algorithms on tensors having different I_n's and F's. In some simulations, we also consider CPD for noisy tensors, i.e., factoring data tensors that have the following signal model:

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} + \boldsymbol{\mathcal{N}},$$

where $\boldsymbol{\mathcal{X}}$ is the noiseless low-rank tensor and $\boldsymbol{\mathcal{N}}$ denotes the additive noise. We use zero-mean i.i.d. Gaussian noise with variance σ_N^2 in our simulations, and the signal-to-noise ratio (SNR) (in dB) is defined as $\mathsf{SNR} = 10\log_{10}\left(\frac{\frac{1}{\prod_{n=1}^{N} I_n}\|\boldsymbol{\mathcal{X}}\|^2}{\sigma_N^2}\right)$.

5.1.2 Baselines

A number of baseline algorithms are employed as benchmarks. Specifically, we mainly use the `AO-ADMM` algorithm [25] and the `APG` algorithm [52] as our baselines since they are the most flexible algorithms with the ability of handling many different regularizations and constraints. We also present the results output by the `CPRAND` algorithm [4]. Note that we are preliminarily interested in constrained/regularized CPD. Because `CPRAND` operates without constraints, the comparison is not entirely fair (e.g., `CPRAND` can potentially

attain smaller cost values since it has a much larger feasible set). Nevertheless, we employ it as a benchmark since it uses the same fiber sampling strategy as ours. All the algorithms are initialized with the same random initialization; i.e., $\boldsymbol{A}_{(0)}$'s entries follow the uniform distribution between 0 and 1.

5.1.3 Parameter Setting

For `BrasCPD`, we set the stepsize to be

$$\alpha^{(t)} = \frac{\alpha}{t^{\beta}}, \tag{5.1}$$

where t is the number of iterations, $\beta = 10^{-6}$ and α typically takes a value in between 0.001 and 0.1, and we try multiple choices of α in our simulations. The batch size $|\mathcal{F}_n|$ is set to be below 25, which will be specified later. For `AdaCPD`, we fix $\beta = 10^{-6}$ and $\eta = 1$ for all the simulations. For `CPRAND`, we follow the instruction in the original paper [4] and sample $10F\log_2 F$ fibers for each update.

5.1.4 Performance Metrics

To measure the performance, we employ two metrics. The first one is the value of the cost function, i.e., $\mathsf{cost} = (1/\prod_{n=1}^{N} I_n) \times f(\boldsymbol{\theta}^{(t)})$. The second one is the estimation accuracy of the latent factors, $\boldsymbol{A}_{(n)}$ for $n = 1, \ldots, N$. The accuracy is measured by the *mean squared error* (MSE) which is as defined in [14, 20]:

$$\mathsf{MSE} = \min_{\pi(r)\in\{1,\ldots,R\}} \frac{1}{R}\sum_{r=1}^{R}\left\| \frac{\boldsymbol{A}_{(n)}(:,\pi(r))}{\|\boldsymbol{A}_{(n)}(:,\pi(r))\|_2} - \frac{\widehat{\boldsymbol{A}}_{(n)}(:,r)}{\|\widehat{\boldsymbol{A}}_{(n)}(:,r)\|_2}\right\|_2^2 \tag{5.2}$$

where $\widehat{\boldsymbol{A}}_{(n)}$ denotes the estimate of $\boldsymbol{A}_{(n)}$ and $\pi(r)$'s are under the constraint $\{\pi(1), \ldots, \pi(R)\} = \{1, \ldots, R\}$—which is used to fix the intrinsic column permutation in CPD.

Since the algorithms under test have very different operations and subproblem-solving strategies, it may be challenging to find an exactly unified complexity measure. In this section, we show the peformance of the algorithms against the number of MTTKRP

operations $\boldsymbol{H}_{(n)}^T \boldsymbol{X}_{(n)}$ used, since $\boldsymbol{H}_{(n)}^T \boldsymbol{X}_{(n)}$ is the most costly step that dominates the complexity of all the algorithms under comparison. All the simulations are conducted in Matlab. The results are averaged from ten random trials with different tensors.

5.2 Results

Fig. 5.1 shows the average MSEs of the estimated latent factors by the algorithms under a much larger scale simulation, where $I_1 = I_2 = I_3 = 300$ and $R = 100$. We set $|F_n| = 18$ so that the proposed algorithms use 5,000 iterations to compute a full MTTKRP. All the algorithms use nonnegativity constraints except `CPRAND`. There are several observations in order: First, the stochastic algorithms (i.e., `BrasCPD`, `AdaCPD`, and `CPRAND`) are much more efficient relative to the deterministic algorithms (`AO-ADMM` and `APG`). After 30 MTTKRPs computed, the stochastic algorithms often have reached a reasonable level of MSE. This is indeed remarkable, since 30 MTTKRPs are roughly equivalent to 10 iterations of `AO-ADMM` and `APG`. Second, two of the proposed stochastic algorithms largely outperforms `CPRAND`. In particular, `BrasCPD` with $\alpha = 0.1$ gives the most promising performance. However, the performance of `BrasCPD` is affected a bit significantly by the parameters α. One can see that using $\alpha = 0.05$ and $\alpha = 0.01$ the algorithm does not give so promising results under this setting. Third, `AdaCPD` yields the second lowest MSEs, but its MSE curve starts saturating and decreases slower after it reaches MSE=10^{-3}. This is understandable, since the 'size' of $\boldsymbol{\eta}$ is shrinking after each iteration and thus the stepsize can vanish after a large number of iterations. Nevertheless, MSE=10^{-3} is already very satisfactory, and `AdaCPD` shows surprising robustness to changing scenarios, without changing any setup in its stepsize scheduling strategy. Fig. 5.2 shows the cost values against the number of full MTTKRPs computed, which is consistent to what we observed in Fig. 5.1.

Table 5.1 shows the MSEs and cost values of output by the algorithms when the tensor rank varies under $I = 300$. All the algorithms are stopped after 30 full MTTKRPs are used. One can see that `BrasCPD` in general exhibits the lowest MSEs if a proper α is chosen, under the employed stepsize schedule in (5.1). However, one can see that when F changes, there is a risk that `BrasCPD` runs into numerical issues and yields unbounded solutions. This suggests that `BrasCPD` may need extra care for tuning its stepsize. On the other hand, `AdaCPD` always outputs reasonably good results. The MSEs output by

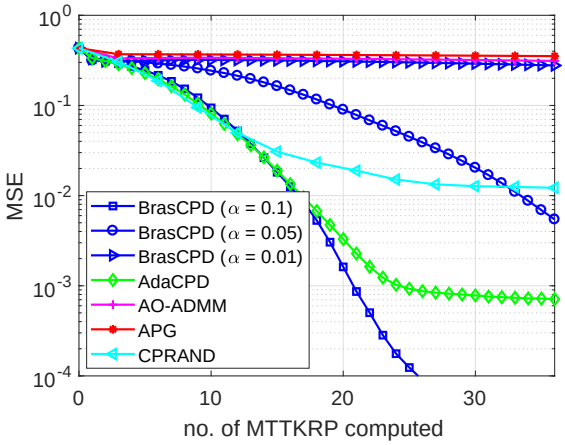


Figure 5.1: MSE of the algorithms. $I_1 = I_2 = I_3 = 300$ and $R = 100$. $\boldsymbol{A}_{(n)} \geq \mathbf{0}$.

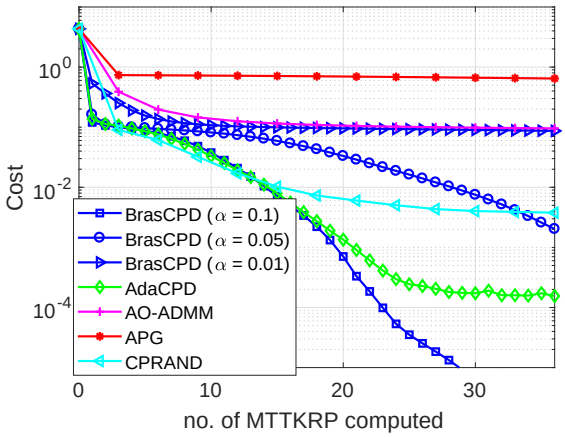


Figure 5.2: cost of the algorithms. $I_1 = I_2 = I_3 = 300$ and $R = 100$. $\boldsymbol{A}_{(n)} \geq \mathbf{0}$.

AdaCPD is slightly higher relative to BrasCPD, but is much lower compared to those of the baslines. More importantly, AdaCPD runs without tuning the stepsize parameters—which shows the power of the adaptive stepsize scheduling strategy.

Table 5.1: Performance of the estimated latent factors by the algorithms under different R; $I = 300$; all the algorithms are stopped after computing 30 MTTKRPs. "NaN" means the algorithm outputs unbounded solutions. $\boldsymbol{A}_{(n)} \geq \boldsymbol{0}$.

Algorithm	Metric	R			
		100	200	300	400
BrasCPD (α=0.1)	MSE	$\mathbf{8.4375 \times 10^{-5}}$	NaN	NaN	NaN
BrasCPD (α =0.05)	MSE	0.0126	**0.0494**	**0.0894**	NaN
BrasCPD (α =0.01)	MSE	0.2882	0.3142	0.3235	0.3239
AdaCPD	MSE	0.0016	0.1247	0.1467	**0.2382**
AO-ADMM	MSE	0.3190	0.3124	0.3093	0.3033
APG	MSE	0.3574	0.3527	0.3538	0.3545
CPRAND	MSE	0.0056	0.0967	0.2115	0.2404
BrasCPD (α =0.1)	Cost	$\mathbf{2.3759 \times 10^{-5}}$	NaN	NaN	NaN
BrasCPD (α =0.05)	Cost	0.0046	**0.0397**	**0.1162**	NaN
BrasCPD (α =0.01)	Cost	0.0903	0.1832	0.2687	0.3461
AdaCPD	Cost	4.8050×10^{-4}	0.1555	0.1684	**0.2144**
AO-ADMM	Cost	0.0990	0.1952	0.2800	0.3520
APG	Cost	0.6649	1.3629	2.0664	2.7707
CPRAND	Cost	0.0018	0.0691	0.1842	0.2481

Table 5.2: Performance of the algorithms under various I's, $R = 100$; all the algorithms are stopped after computing 30 MTTKRPs. $\boldsymbol{A}_{(n)} \geq \boldsymbol{0}$.

Algorithm	Metric	I			
		100	200	300	400
BrasCPD (α=0.1)	MSE	0.2432	0.0474	$\mathbf{8.4375 \times 10^{-5}}$	$\mathbf{1.2052 \times 10^{-9}}$
BrasCPD (α =0.05)	MSE	0.2724	0.2000	0.0126	1.0631×10^{-4}
BrasCPD (α =0.01)	MSE	0.2906	0.3086	0.2882	0.2127
AdaCPD	MSE	0.2214	**0.0121**	0.0016	1.0068×10^{-4}
AO-ADMM	MSE	0.2561	0.3171	0.3190	0.3235
APG	MSE	0.3107	0.3459	0.3574	0.3635
CPRAND	MSE	**0.1857**	0.0459	0.0056	0.0025
BrasCPD (α =0.1)	Cost	0.0795	0.0179	$\mathbf{2.3759 \times 10^{-5}}$	$\mathbf{3.5023 \times 10^{-10}}$
BrasCPD (α =0.05)	Cost	0.0862	0.0668	0.0046	2.8758×10^{-5}
BrasCPD (α =0.01)	Cost	0.1453	0.0981	0.0903	0.2127
AdaCPD	Cost	0.0814	**0.0058**	4.8050×10^{-4}	3.5958×10^{-5}
AO-ADMM	Cost	0.0843	0.0957	0.0990	0.1008
APG	Cost	0.5936	0.6450	0.6649	0.6776
CPRAND	Cost	**0.0566**	0.0136	0.0018	0.0011

Tables 5.3-5.4 show the performance of the algorithms under different SNRs. Except for adding noise, other settings are the same as those in Fig. 5.1. In a noisy environment, the ability of handling constraints/regularizations is essential for a CPD algorithm, since prior information on the latent factors can help improve estimation accuracy. Table 5.3

Table 5.3: Performance of the algorithms under various SNRs; all the algorithms are stopped after computing 30 MTTKRPs. $I_1 = I_2 = I_3 = 300$, $R = 100$. $\boldsymbol{A}_{(n)} \geq \mathbf{0}$.

Algorithm	Metric	SNR			
		10	20	30	40
BrasCPD (α=0.1)	MSE	0.3685	0.0225	**0.0024**	**0.0003**
BrasCPD (α =0.05)	MSE	0.2962	0.0198	0.0066	0.0044
BrasCPD (α =0.01)	MSE	**0.3125**	0.2823	0.2774	0.2758
AdaCPD	MSE	0.3285	**0.0192**	0.0025	0.0004
AO-ADMM	MSE	0.3330	0.3135	0.3118	0.3101
APG	MSE	0.3524	0.3521	0.3521	0.3520
CPRAND	MSE	1.6047	0.0367	0.0104	0.0100
BrasCPD (α =0.1)	Cost	1.9627	0.2081	0.0212	**0.0021**
BrasCPD (α =0.05)	Cost	0.9086	0.0918	**0.0110**	0.0025
BrasCPD (α =0.01)	Cost	0.2812	0.1058	0.0885	0.0865
AdaCPD	Cost	0.3137	**0.0671**	0.0155	0.0024
AO-ADMM	Cost	**0.1533**	0.0999	0.0954	0.0948
APG	Cost	0.6445	0.6441	0.6430	0.6435
CPRAND	Cost	0.8038	0.0811	0.0100	0.0039

Table 5.4: Performance of the algorithms under various SNRs after computing 30 MTTKRPs. $I_1 = I_2 = I_3 = 300$, $R = 100$. $\mathbf{1}^\top \boldsymbol{A}_{(n)} = \rho \mathbf{1}^\top$, $\boldsymbol{A}_{(n)} \geq \mathbf{0}$. $\rho = 300$.

Algorithm	Metric	SNR			
		10	20	30	40
BrasCPD (α =0.1)	MSE	0.4697	0.4423	0.3956	0.4320
BrasCPD (α =0.05)	MSE	0.4443	0.4267	0.4135	0.4146
BrasCPD (α =0.01)	MSE	0.3940	**0.0335**	0.0033	0.0003
AdaCPD	MSE	**0.2983**	0.0611	**0.0011**	**0.0002**
AO-ADMM	MSE	0.3206	0.2996	0.2973	0.2972
APG	MSE	0.2761	0.2760	0.2760	0.2760
CPRAND	MSE	1.6020	0.0466	0.0045	**0.0112**
BrasCPD (α =0.1)	Cost	14274.5141	12059.7192	7386.9652	10944.0721
BrasCPD (α =0.05)	Cost	11424.7030	10159.3117	9059.4911	9152.1151
BrasCPD (α =0.01)	Cost	229.6424	24.8565	2.5698	0.2571
AdaCPD	Cost	**14.8627**	**3.4755**	0.5916	**0.1318**
AO-ADMM	Cost	9.6097	6.1642	5.8643	5.8359
APG	Cost	36.5461	36.5095	36.5059	36.5055
CPRAND	Cost	51.5269	5.2663	**0.5413**	0.2570

and Table 5.4 test the cases where $\boldsymbol{A}_{(n)}$ is elementwise nonnegative and the columns of $\boldsymbol{A}_{(n)}$ reside in a scaled version of the probability simplex, respectively. One can see from the two tables that both `BrasCPD` (with a proper α) and `AdaCPD` work very well. In Table 5.4, one can see that `BrasCPD` again shows its sensitivity to the choice of α, with $\alpha = 0.1$ and 0.05 actually not working. We also note that when the SNR is low, `CPRAND` is not as competitive, perhaps because it cannot use constraints to incorporate prior information of the $\boldsymbol{A}_{(n)}$'s—this also shows the importance of being able to handle various constraints.

5.3 Real-Data Experiment

In this subsection, we test our algorithm on a constrained tensor decomposition problem; i.e., we apply the proposed `BrasCPD` and `AdaCPD` to factor hyperspectral images. Hyperspectral images (HSIs) are special images with pixels measured at a large number of wavelengths. Hence, an HSI is usually stored as a third-order tensor with two spatial coordinates and one spectral coordinate. HSIs are dense tensors and thus are suitable for testing the proposed algorithms. We use sub-images of the Indian Pines dataset that has a size of $145 \times 145 \times 220$ and the Pavia University dataset[1] that has a size of $610 \times 340 \times 103$.

Tables 5.5-5.6 show the cost values of the nonnegativity constrained optimization algorithms under different ranks, after computing 10 MTTKRPs for all three modes, which corresponds to 10 iterations for `AO-ADMM` and `APG` (we use this "all-mode MTTKRP" in this section since the tensors are unsymmetrical and thus single-mode MTTKRPs cannot be directly translated to iterations in batch algorithms). One can see that the proposed algorithms show the same merits as we have seen in the simulations: `BrasCPD` can exhibit very competitive performance when α is properly chosen (e.g., when $R = 10$ and $\alpha = 4$ for the Indian Pines dataset); in addition, `AdaCPD` gives consistently good performance without tuning the stepsize manually. Particularly, on the Pavia University dataset, `AdaCPD` gives much lower cost values compared to other algorithms. Fig. 5.3 shows how the cost values change along with the iterations on the Pavia University data using $R = 200$.

[1]Both datasets are available online: `http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes`

Table 5.5: Performance of the algorithms on the Pavia University dataset under different R's.

Algorithm	Metric	R			
		10	20	30	40
`BrasCPD` (α =4)	Cost	$\mathbf{6.8343 \times 10^{-4}}$	4.7777×10^{-4}	$\mathbf{3.4738 \times 10^{-4}}$	$\mathbf{2.9053 \times 10^{-4}}$
`BrasCPD` (α =3)	Cost	6.8507×10^{-4}	4.8550×10^{-4}	4.1556×10^{-4}	3.1278×10^{-4}
`BrasCPD` (α =2)	Cost	6.9877×10^{-4}	5.7753×10^{-4}	5.4205×10^{-4}	4.2504×10^{-4}
`AdaCPD`	Cost	7.0677×10^{-4}	$\mathbf{4.6180 \times 10^{-4}}$	3.5328×10^{-4}	2.9848×10^{-4}
`AO-ADMM`	Cost	7.2503×10^{-4}	5.5708×10^{-4}	5.1489×10^{-4}	5.1505×10^{-4}
`APG`	Cost	1.9392×10^{-3}	1.8952×10^{-3}	1.8818×10^{-3}	1.8675×10^{-3}

Table 5.6: Performance of the algorithms on the Indian Pines dataset under different R's.

Algorithm	Metric	R	
		100	200
`BrasCPD` (α =0.5)	Cost	2.7193×10^{-3}	2.5275×10^{-3}
`BrasCPD` (α =0.3)	Cost	3.6496×10^{-3}	5.3453×10^{-3}
`BrasCPD` (α =0.1)	Cost	6.4221×10^{-3}	5.7509×10^{-3}
`AdaCPD`	Cost	$\mathbf{1.7269 \times 10^{-3}}$	$\mathbf{9.0080 \times 10^{-4}}$
`AO-ADMM`	Cost	6.2494×10^{-3}	4.5879×10^{-3}
`APG`	Cost	7.2966×10^{-3}	7.2647×10^{-3}

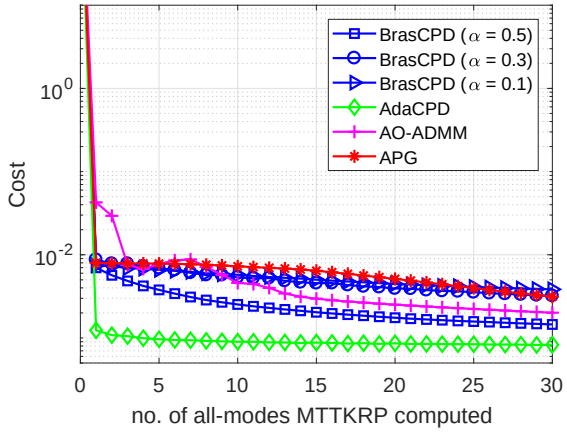


Figure 5.3: No. of all-mode MTTKRPs v.s. cost values output by the algorithms when applied to the Pavia University dataset. $R = 200$. Nonnegativity constraint is added.

Chapter 6 Conclusion

We presented a problem setting in which a large-scale dense tensor is being decomposed to many factors. We first propose a ***B**lock-**Ra**ndomized **S**GD for **CPD*** (`BrasCPD`) algorithm which can easily handle a variety of constraints and regularizations that are commonly used in signal processing and data analytics which is reminiscent of stochastic proximal gradient. And we compare MTTKRPs which is the most resource consuming operations along all the algorithms and find that our proposed BrasCPD outperforms a number of state-of-art unconstrained and constrained CPD algorithms.

Inspired by the deep neural network optinization algorithms, we also proposed `Adagrad` algorithm which takes tha advantage of using small stepsize to update heavily updating variables for current iteration(and a large stepsize otherwise). And this strategy can help us to upadate the variables in a balanced manner.

Both the algorithms can help us to achieve a reasonably high accuracy when using the same MTTKRP operations. In synthetic data, it is obviously that our proposed BrasCPD and AdaCPD outperform over all the baselines. When it comes to the real world dataset, the cost function value of our algorithms are much lower than any other state-of-art algorithms.

It is highly favorable that our algorithms have two significant advantages. First, being able to quickly improve estimation accuracy of the latent factors can help us save a lot of computing resources. Second, they are flexible with incorporating constraints and regularizations which is important to handle our real life applications.

Bibliography

[1] Andrew L Alexander, Jee Eun Lee, Mariana Lazar, and Aaron S Field. Diffusion tensor imaging of the brain. *Neurotherapeutics*, 4(3):316–329, 2007.

[2] Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *The Journal of Machine Learning Research*, 15(1):2773–2832, 2014.

[3] Animashree Anandkumar, Daniel Hsu, and Sham M Kakade. A method of moments for mixture models and hidden markov models. In *Conference on Learning Theory*, pages 33–1, 2012.

[4] Casey Battaglino, Grey Ballard, and Tamara G Kolda. A practical randomized cp tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, 39(2):876–901, 2018.

[5] Amir Beck and Luba Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM journal on Optimization*, 23(4):2037–2060, 2013.

[6] Dimitri P Bertsekas. Nonlinear optimization. *Athena Scientific, Belmont*, 1999.

[7] Alex Beutel, Partha Pratim Talukdar, Abhimanu Kumar, Christos Faloutsos, Evangelos E Papalexakis, and Eric P Xing. Flexifact: Scalable flexible factorization of coupled tensors on hadoop. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 109–117. SIAM, 2014.

[8] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[9] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[10] Rasmus Bro and Nicholaos D Sidiropoulos. Least squares algorithms under unimodality and non-negativity constraints. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 12(4):223–247, 1998.

[11] J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. *Psychometrika*, 35(3):283–319, 1970.

[12] Xiangyi Chen, Sijia Liu, Ruoyu Sun, and Mingyi Hong. On the convergence of a class of adam-type algorithms for non-convex optimization. *arXiv preprint arXiv:1808.02941*, 2018.

[13] Pierre Comon, Xavier Luciani, and André LF De Almeida. Tensor decompositions, alternating least squares and other tales. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 23(7-8):393–405, 2009.

[14] Lieven De Lathauwer and Joséphine Castaing. Blind identification of underdetermined mixtures by simultaneous matrix diagonalization. *IEEE Transactions on Signal Processing*, 56(3):1096–1105, 2008.

[15] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

[16] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[17] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l 1-ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 272–279. ACM, 2008.

[18] Xiao Fu, Cheng Gao, Hoi-To Wai, and Kejun Huang. Block-randomized stochastic proximal gradient for constrained low-rank tensor factorization. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7485–7489. IEEE, 2019.

[19] Xiao Fu, Cheng Gao, Hoi-To Wai, and Kejun Huang. Block-randomized stochastic proximal gradient for low-rank tensor factorization. *arXiv preprint arXiv:1901.05529*, 2019.

[20] Xiao Fu, Wing-Kin Ma, Kejun Huang, and Nicholas D Sidiropoulos. Blind separation of quasi-stationary sources: Exploiting convex geometry in covariance domain. *IEEE Transactions on Signal Processing*, 63(9):2306–2320, 2015.

[21] Xiao Fu, Nicholas D Sidiropoulos, John H Tranter, and Wing-Kin Ma. A factor analysis framework for power spectra separation and multiple emitter localization. *IEEE Transactions on Signal Processing*, 63(24):6581–6594, 2015.

[22] Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[23] Saeed Ghadimi and Guanghui Lan. Accelerated gradient methods for nonconvex nonlinear and stochastic programming. *Mathematical Programming*, 156(1-2):59–99, 2016.

[24] Christopher J Hillar and Lek-Heng Lim. Most tensor problems are np-hard. *Journal of the ACM (JACM)*, 60(6):45, 2013.

[25] Kejun Huang, Nicholas D Sidiropoulos, and Athanasios P Liavas. A flexible and efficient algorithmic framework for constrained matrix and tensor factorization. *IEEE Trans. Signal Processing*, 64(19):5052–5065, 2016.

[26] Charilaos I Kanatsoulis, Xiao Fu, Nicholas D Sidiropoulos, and Wing-Kin Ma. Hyperspectral super-resolution: A coupled tensor factorization approach. *IEEE Transactions on Signal Processing*, 66(24):6503–6517, 2018.

[27] U Kang, Evangelos Papalexakis, Abhay Harpale, and Christos Faloutsos. Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 316–324. ACM, 2012.

[28] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[30] Joseph B Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29(2):115–129, 1964.

[31] Xiaoyu Li and Francesco Orabona. On the convergence of stochastic gradient descent with adaptive stepsizes. *arXiv preprint arXiv:1805.08114*, 2018.

[32] Athanasios P Liavas and Nicholas D Sidiropoulos. Parallel algorithms for constrained tensor factorization via alternating direction method of multipliers. *IEEE Transactions on Signal Processing*, 63(20):5450–5463, 2015.

[33] Carmeliza Navasca, Lieven De Lathauwer, and Stefan Kindermann. Swamp reducing technique for tensor decomposition. In *2008 16th European Signal Processing Conference*, pages 1–5. IEEE, 2008.

[34] Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[35] Evangelos E Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. Parcube: Sparse parallelizable tensor decompositions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 521–536. Springer, 2012.

[36] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[37] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. Introduction to tensor decompositions and their applications in machine learning. *arXiv preprint arXiv:1711.10781*, 2017.

[38] Bhaskar D Rao and Kenneth Kreutz-Delgado. An affine scaling methodology for best basis selection. *IEEE Transactions on signal processing*, 47(1):187–200, 1999.

[39] Meisam Razaviyayn, Mingyi Hong, and Zhi-Quan Luo. A unified convergence analysis of block successive minimization methods for nonsmooth optimization. *SIAM Journal on Optimization*, 23(2):1126–1153, 2013.

[40] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[41] Ernest K Ryu and Stephen Boyd. Primer on monotone operator methods. *Appl. Comput. Math*, 15(1):3–43, 2016.

[42] Amnon Shashua and Tamir Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *Proceedings of the 22nd international conference on Machine learning*, pages 792–799. ACM, 2005.

[43] Nicholas D Sidiropoulos, Rasmus Bro, and Georgios B Giannakis. Parallel factor analysis in sensor array processing. *IEEE transactions on Signal Processing*, 48(8):2377–2388, 2000.

[44] Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017.

[45] Nicholas D Sidiropoulos and Xiangqian Liu. Identifiability results for blind beamforming in incoherent multipath with small delay spread. *IEEE Transactions on Signal Processing*, 49(1):228–236, 2001.

[46] Shaden Smith and George Karypis. A medium-grained algorithm for sparse tensor factorization. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 902–911. IEEE, 2016.

[47] Laurent Sorber, Marc Van Barel, and Lieven 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 Journal on Optimization*, 23(2):695–720, 2013.

[48] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 374–383. ACM, 2006.

[49] Nico Vervliet and Lieven De Lathauwer. A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors. *IEEE Journal of Selected Topics in Signal Processing*, 10(2):284–295, 2016.

[50] Huahua Wang and Arindam Banerjee. Randomized block coordinate descent for online and stochastic optimization. *arXiv preprint arXiv:1407.0107*, 2014.

[51] Lin Xiao and Tong Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

[52] Yangyang Xu and Wotao Yin. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on imaging sciences*, 6(3):1758–1789, 2013.

[53] Yangyang Xu and Wotao Yin. Block stochastic gradient iteration for convex and nonconvex optimization. *SIAM Journal on Optimization*, 25(3):1686–1716, 2015.

[54] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[55] Shuo Zhou, Sarah M Erfani, and James Bailey. Sced: A general framework for sparse tensor decomposition with constraints and elementwise dynamic learning. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 675–684. IEEE, 2017.

APPENDIX

Chapter A Convergence analysis

A.1 Connection between $\nabla f(\boldsymbol{\theta}^{(t)})$ and $\boldsymbol{G}^{(t)}$

Let us consider the following conditional expectation:

$$\begin{aligned}
\overline{\boldsymbol{G}}_{(n)}^{(t)} &= \mathbb{E}\big[\boldsymbol{G}_{(n)}^{(t)} \,|\, \mathcal{B}^{(t)}\big] = \mathbb{E}\big[\boldsymbol{G}_{(n)}^{(t)} \,|\, \{\boldsymbol{A}_{(n)}^{(t)}\}_{n=1}^{N}\big] \\
&= \mathbb{E}_{n'}\left[\frac{1}{J_{\frac{n'}{\mathcal{F}_n}}}\big(\boldsymbol{A}_{(n')}^{(t)}\boldsymbol{H}_{(n')}^{\top}\boldsymbol{H}_{(n')} - \boldsymbol{X}_{(n')}^{\top}\boldsymbol{H}_{(n')}\big)\right] \\
&\overset{(a)}{=} \sum_{n'=1}^{N}\frac{\delta(n'-n)}{NJ_{\frac{n'}{\mathcal{F}_n}}}\big(\boldsymbol{A}_{(n')}^{(t)}\boldsymbol{H}_{(n')}^{\top}\boldsymbol{H}_{(n')} - \boldsymbol{X}_{(n')}^{\top}\boldsymbol{H}_{(n')}\big) \\
&= \frac{1}{NJ_{\frac{n'}{\mathcal{F}_n}}}\big(\boldsymbol{A}_{(n)}^{(t)}\boldsymbol{H}_{(n)}^{\top}\boldsymbol{H}_{(n)} - \boldsymbol{X}_{(n)}^{\top}\boldsymbol{H}_{(n)}\big) \qquad \text{(A.1)}
\end{aligned}$$

where $\delta(\cdot)$ is the Dirac function and the expectation in *(a)* is taken over the possible modes n'. The last equality shows that $\overline{\boldsymbol{G}}_{(n)}^{(t)}$ is a scaled version of the gradient of the objective function of (4.1) taken w.r.t. $\boldsymbol{A}_{(n)}^{(t)}$. Hence, the block sampling step together with fiber sampling entails us an easy way to estimate the *full gradient w.r.t. all the latent factors* in an unbiased manner.

A.2 Proof of Proposition 1

To show Proposition 3, we will need the following [6, Proposition 1.2.4]:

Lemma 1 *Let $\{a_t\}_t$ and $\{b_t\}_t$ be two nonnegative sequences such that b_t is bounded, $\sum_{t=0}^{\infty} a_t b_t$ converges and $\sum_{t=0}^{\infty} a_t$ diverges, then we have*

$$\liminf_{t\to\infty} b_t = 0.$$

To make our notations precise, let us denote $\xi^{(t)}$ as the random index of mode chosen

at iteration t and subsequently $\mathcal{F}^{(t)}_{(\xi^{(t)})}$ is the random set of fibers chosen. Under Assumption 2, we have $\|\boldsymbol{H}^{(t)}_{(\xi^{(t)})}\|_2^2 \leq L^{(t)}_{(\xi^{(t)})}$ where $\boldsymbol{H}^{(t)}_{(\xi^{(t)})} = \odot_{n'=1,n'\neq\xi^{(t)}}^{N} \boldsymbol{A}^{(t)}_{(n')}$ and $L^{(t)}_{(\xi^{(t)})} < \infty$. Combining with Fact 1, we observe the following bound:

$$\begin{aligned}
&f(\boldsymbol{\theta}^{(t+1)}) - f(\boldsymbol{\theta}^{(t)}) \\
&\leq \left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}), \boldsymbol{A}^{(t+1)}_{(\xi^{(t)})} - \boldsymbol{A}^{(t)}_{(\xi^{(t)})} \right\rangle + \frac{L}{2} \left\| \boldsymbol{A}^{(t+1)}_{(\xi^{(t)})} - \boldsymbol{A}^{(t)}_{(\xi^{(t)})} \right\|_F^2 \\
&= -\alpha^{(t)} \left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}), \boldsymbol{G}^{(t)}_{(\xi^{(t)})} \right\rangle + \frac{(\alpha^{(t)})^2 L}{2} \left\| \boldsymbol{G}^{(t)}_{(\xi^{(t)})} \right\|_F^2,
\end{aligned}$$

where we denote

$$L = \max_{t=0,\ldots,\infty} L^{(t)}_{(\xi^{(t)})} < \infty,$$

since $\boldsymbol{A}^{(t)}_{(n)}$ is bounded for all iterations.

Taking expectation conditioned on the filtration $\mathcal{B}^{(t)}$ and the chosen mode index $\xi^{(t)}$, we have

$$\begin{aligned}
&\mathbb{E}\left[f(\boldsymbol{\theta}^{(t+1)}) \mid \mathcal{B}^{(t)}, \xi^{(t)} \right] - f(\boldsymbol{\theta}^{(t)}) \\
&\leq -\alpha^{(t)} \left\| \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}) \right\|^2 \\
&\qquad + \frac{(\alpha^{(t)})^2 L}{2} \mathbb{E}\left[\left\| \boldsymbol{G}^{(t)}_{(\xi^{(t)})} \right\|_F^2 \mid \mathcal{B}^{(t)}, \xi^{(t)} \right] \\
&\leq -\alpha^{(t)} \left\| \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}) \right\|^2 + \frac{(\alpha^{(t)})^2 L M}{2}.
\end{aligned} \tag{A.2}$$

where the first inequality used the assumption that $L^{(t)}_{(\xi^{(t)})} \leq L$ and Fact 2, and the second inequality is again a consequence of Assumption 2, as we observe:

$$\begin{aligned}
\left\| \boldsymbol{G}^{(t)}_{(\xi^{(t)})} \right\| = \frac{1}{|\mathcal{F}_n|} \Big\| &\boldsymbol{A}^{(t)}_{(\xi^{(t)})} (\boldsymbol{H}^{(t)}_{(\xi^{(t)})}(\mathcal{F}_n))^\top \boldsymbol{H}^{(t)}_{(\xi^{(t)})}(\mathcal{F}_n) \\
&- \boldsymbol{X}^\top_{(\xi^{(t)})}(\mathcal{F}_n) \boldsymbol{H}^{(t)}_{(\xi^{(t)})}(\mathcal{F}_n) \Big\|
\end{aligned} \tag{A.3}$$

where in the right hand side we have dropped the superscript for $n^{(t)}$ for simplicity. As $\boldsymbol{X}_{(n)}$ is bounded for all n, and all the $\boldsymbol{A}^{(t)}_{(n)}$ are bounded under Assumption 2, we have $\|\boldsymbol{G}^{(t)}_{(\xi^{(t)})}\|^2 \leq M$ for all n, t and some $M < \infty$. Now, taking the expectation w.r.t. $\xi^{(t)}$

yields

$$\begin{aligned}
&\mathbb{E}_{\xi^{(t)}}\left[f(\boldsymbol{\theta}^{(t+1)}) \,|\, \mathcal{B}^{(t)}\right] - f(\boldsymbol{\theta}^{(t)}) \\
&\leq -\alpha^{(t)}\mathbb{E}_{\xi^{(t)}}\left[\left\|\nabla_{\boldsymbol{A}_{(n^{(t)})}} f(\boldsymbol{\theta}^{(t)})\right\|^2\right] + \frac{(\alpha^{(t)})^2 ML}{2}.
\end{aligned} \tag{A.4}$$

Finally, taking the total expectation, we have

$$\begin{aligned}
&\mathbb{E}\left[f(\boldsymbol{\theta}^{(t+1)})\right] - \mathbb{E}\left[f(\boldsymbol{\theta}^{(t)})\right] \\
&\leq -\alpha^{(t)}\mathbb{E}\left[\mathbb{E}_{\xi^{(t)}}\left[\left\|\nabla_{\boldsymbol{A}_{(n^{(t)})}} f(\boldsymbol{\theta}^{(t)})\right\|^2\right]\right] + \frac{(\alpha^{(t)})^2 ML}{2}.
\end{aligned} \tag{A.5}$$

Summing up the above from $q = 0$ to $q = t$, we have

$$\begin{aligned}
&\mathbb{E}\left[f(\boldsymbol{\theta}^{(q+1)})\right] - f(\boldsymbol{\theta}^{(0)}) \\
&\leq \sum_{q=0}^{t} -\alpha^{(q)}\mathbb{E}\left[\mathbb{E}_{\xi^{(q)}}\left[\left\|\nabla_{\boldsymbol{A}_{(n^{(q)})}} f(\boldsymbol{\theta}^{(q)})\right\|^2\right]\right] \\
&\quad + \sum_{q=0}^{t} \frac{(\alpha^{(q)})^2 ML}{2}.
\end{aligned} \tag{A.6}$$

Taking $t \to \infty$, the above implies that

$$\begin{aligned}
&\sum_{t=0}^{\infty} \alpha^{(t)}\mathbb{E}\left[\mathbb{E}_{\xi^{(t)}}\left[\left\|\nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)})\right\|^2\right]\right] \\
&\qquad \leq f(\boldsymbol{\theta}^{(0)}) - f(\boldsymbol{\theta}^{(\star)}) + \sum_{t=0}^{\infty} \frac{(\alpha^{(t)})^2 ML}{2},
\end{aligned} \tag{A.7}$$

where $f(\boldsymbol{\theta}^{(\star)})$ denotes the global optimal value. Note that the right hand side above is bounded from above because $\sum_{t=0}^{\infty}(\alpha^{(t)})^2 < \infty$. Hence, using Lemma 1, we can conclude that

$$\liminf_{t\to 0} \mathbb{E}\left[\mathbb{E}_{\xi^{(t)}}\left[\left\|\nabla_{\boldsymbol{A}_{(n^{(t)})}} f(\boldsymbol{\theta}^{(t)})\right\|^2\right]\right] = 0.$$

Finally, we conclude:

$$\liminf_{t\to 0} \mathbb{E}\left[\left\|\nabla f(\boldsymbol{\theta}^{(t)})\right\|^2\right] = 0,$$

since

$$\mathbb{E}_{\xi^{(t)}}\left[\left\|\nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)})\right\|^2\right] = \frac{1}{N}\sum_{n=1}^{N}\left\|\nabla_{\boldsymbol{A}_{(n)}} f(\boldsymbol{\theta}^{(t)})\right\|^2$$

by the fundamental theorem of expectation.

A.3 Proof of Proposition 2

A.3.1 Preliminaries

For the constrained case, let us denote $\Phi(\boldsymbol{\theta}) = f(\boldsymbol{\theta}) + \sum_{n=1}^{N} h_n(\boldsymbol{\theta})$ as the objective function. Unlike the unconstrained case where we measure convergence via observing if the gradient vanishes, the optimality condition of the constrained case is a bit more complicated. Here, the idea is to observe the "generalized gradient". To be specific, consider the following optimization problem

$$\underset{\boldsymbol{\theta}}{\text{minimize}}\ \ f(\boldsymbol{\theta}) + h(\boldsymbol{\theta}),$$

where $f(\boldsymbol{\theta})$ is continuously differentiable while h is convex but possibly nonsmooth. The deterministic proximal gradient algorithm for handling this problem is as follows:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \mathsf{Prox}_h\left(\boldsymbol{\theta}^{(t)} - \alpha^{(t)}\nabla f(\boldsymbol{\theta}^{(t)})\right).$$

Define $\boldsymbol{P}^{(t)} = \frac{1}{\alpha^{(t)}}\left(\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\right)$, the update can also be represented as $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \alpha^{(t)}\boldsymbol{P}^{(t)}$, which is analogous to the gradient descent algorithm. It can be shown that $\boldsymbol{P}^{(t)} = \mathbf{0}$ implies that the necessary optimality condition is satisfied, and thus $\boldsymbol{P}^{(t)}$ can be considered as a "generalized gradient".

In our case, let us denote $\Phi(\boldsymbol{\theta}) = f(\boldsymbol{\theta}) + \sum_{n=1}^{N} h_n(\boldsymbol{\theta})$ as the objective function. Our optimality condition amounts to $\boldsymbol{P}_{(n)}^{(t)} = \mathbf{0}, \quad \forall\ n$, where

$$\boldsymbol{P}_{(n)}^{(t)} = \frac{1}{\alpha^{(t)}}\left(\boldsymbol{A}_{(n)}^{(t+1)} - \mathsf{Prox}_{h_n}\left(\boldsymbol{A}_{(n)}^{(t)} - \alpha^{(t)}\nabla_{\boldsymbol{A}_{(n)}} f(\boldsymbol{\theta}^{(t)})\right)\right);$$

i.e., the optimality condition is satisfied in a blockwise fashion [39, 53]. Hence, our goal of this section is to show that $\mathbb{E}\left[\boldsymbol{P}_{(n)}^{(t)}\right]$ for all n vanishes when t goes to infinity.

A.3.2 Proof

Our update is equivalent to the following:

$$\begin{aligned} \boldsymbol{A}_{(n)}^{(t+1)} \leftarrow \arg\min_{\boldsymbol{A}_{(n)}} \; & \left\langle \boldsymbol{G}_{(n)}^{(t)}, \boldsymbol{A}_{(n)} - \boldsymbol{A}_{(n)}^{(t)} \right\rangle \\ & + \frac{1}{2\alpha^{(t)}} \left\| \boldsymbol{A}_{(n)} - \boldsymbol{A}_{(n)}^{(t)} \right\|^2 + h_n(\boldsymbol{A}_{(n)}) \end{aligned} \tag{A.8}$$

for a randomly selected n, which is a proximity operator. For a given $\xi^{(t)}$, we have

$$\begin{aligned} & h_{\xi^{(t)}}\left(\boldsymbol{A}_{(\xi^{(t)})}^{(t+1)}\right) - h_{\xi^{(t)}}\left(\boldsymbol{A}_{(\xi^{(t)})}^{(t)}\right) \\ \leq & -\left\langle \boldsymbol{G}_{(\xi^{(t)})}^{(t)}, \boldsymbol{A}_{(\xi^{(t)})}^{(t+1)} - \boldsymbol{A}_{(\xi^{(t)})}^{(t)} \right\rangle - \frac{1}{2\alpha^{(t)}} \left\| \boldsymbol{A}_{(\xi^{(t)})}^{(t+1)} - \boldsymbol{A}_{(\xi^{(t)})}^{(t)} \right\|^2 \end{aligned}$$

by the optimality of $\boldsymbol{A}_{(\xi^{(t)})}^{(t+1)}$ for solving Problem (A.8).

By the block Lipschitz continuity of the smooth part (cf. Fact 1), we have

$$\begin{aligned} f(\boldsymbol{\theta}^{(t+1)}) - f(\boldsymbol{\theta}^{(t)}) \leq & \left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}), \boldsymbol{A}_{(\xi^{(t)})}^{(t+1)} - \boldsymbol{A}_{(\xi^{(t)})}^{(t)} \right\rangle \\ & + \frac{L_{(\xi^{(t)})}^{(t)}}{2} \left\| \boldsymbol{A}_{(\xi^{(t)})}^{(t+1)} - \boldsymbol{A}_{(\xi^{(t)})}^{(t)} \right\|^2, \end{aligned}$$

where f denotes the smooth part in the objective function and

$$L_{(\xi^{(t)})}^{(t)} = \lambda_{\max}\left(\left(\boldsymbol{H}_{(\xi^{(t)})}^{(t)} \right)^\top \boldsymbol{H}_{(\xi^{(t)})}^{(t)} \right) \leq L.$$

Combining the two inequalities, we have

$$\begin{aligned} \Phi(\boldsymbol{\theta}^{(t+1)}) \leq & \Phi(\boldsymbol{\theta}^{(t)}) - \alpha^{(t)} \left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}) - \boldsymbol{G}_{(\xi^{(t)})}^{(t)}, \boldsymbol{p}_{(\xi^{(t)})}^{(t)} \right\rangle \\ & + \left(\frac{L(\alpha^{(t)})^2}{2} - \frac{\alpha^{(t)}}{2} \right) \| \boldsymbol{p}_{(\xi^{(t)})}^{(t)} \|^2 \end{aligned} \tag{A.9}$$

where we define

$$\boldsymbol{p}_{(\xi^{(t)})}^{(t)} = \frac{1}{\alpha^{(t)}} \left(\boldsymbol{A}_{(\xi^{(t)})}^{(t+1)} - \boldsymbol{A}_{(\xi^{(t)})}^{(t)} \right).$$

The inequality in (A.9) can be further written as

$$\begin{aligned}
&\Phi(\boldsymbol{\theta}^{(t+1)}) - \Phi(\boldsymbol{\theta}^{(t)}) \\
&\leq -\alpha^{(t)} \left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}) - \boldsymbol{G}^{(t)}_{(\xi^{(t)})}, \boldsymbol{p}^{(t)}_{(\xi^{(t)})} - \boldsymbol{P}^{(t)}_{(\xi^{(t)})} \right\rangle \\
&\quad - \alpha^{(t)} \left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}) - \boldsymbol{G}^{(t)}_{(\xi^{(t)})}, \boldsymbol{P}^{(t)}_{(\xi^{(t)})} \right\rangle \\
&\quad + \left(\frac{L(\alpha^{(t)})^2}{2} - \frac{\alpha^{(t)}}{2} \right) \|\boldsymbol{p}^{(t)}_{(\xi^{(t)})}\|^2,
\end{aligned} \tag{A.10}$$

Again, taking expectation conditioning on the filtration $\mathcal{B}^{(t)}$ and $\xi^{(t)}$, we have

$$\begin{aligned}
&\mathbb{E}_{\zeta^{(t)}} \left[\Phi(\boldsymbol{\theta}^{(t+1)}) | \mathcal{B}^{(t)}, \xi^{(t)} \right] - \Phi(\boldsymbol{\theta}^{(t)}) \\
&\leq \alpha^{(t)} \mathbb{E}_{\zeta^{(t)}} \left[\left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}) - \boldsymbol{G}^{(t)}_{(\xi^{(t)})}, \boldsymbol{P}^{(t)}_{(\xi^{(t)})} - \boldsymbol{p}^{(t)}_{(\xi^{(t)})} \right\rangle | \mathcal{B}^{(t)}, \xi^{(t)} \right] \\
&\quad + \left(\frac{L(\alpha^{(t)})^2}{2} - \frac{\alpha^{(t)}}{2} \right) \mathbb{E}_{\zeta^{(t)}} \left[\left\| \boldsymbol{p}^{(t)}_{(\xi^{(t)})} \right\|^2 \middle| \mathcal{B}^{(t)}, \xi^{(t)} \right],
\end{aligned} \tag{A.11}$$

i.e., the second term on the right hand side of (A.10) becomes zero because of Fact 2. The first term on the right hand side of (A.11) can be bounded via the following chain of inequalities:

$$\begin{aligned}
&\mathbb{E}_{\zeta^{(t)}} \left[\left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}) - \boldsymbol{G}^{(t)}_{(\xi^{(t)})}, \boldsymbol{P}^{(t)}_{(\xi^{(t)})} - \boldsymbol{p}^{(t)}_{(\xi^{(t)})} \right\rangle \middle| \mathcal{B}^{(t)}, \xi^{(t)} \right] \\
&\quad \leq \mathbb{E}_{\zeta^{(t)}} \left[\left\| \boldsymbol{\delta}^{(t)} \right\| \left\| \boldsymbol{P}^{(t)}_{(\xi^{(t)})} - \boldsymbol{p}^{(t)}_{(\xi^{(t)})} \right\| \;\middle|\; \mathcal{B}^{(t)}, \xi^{(t)} \right] \\
&\quad \leq \mathbb{E}_{\zeta^{(t)}} \left[\|\boldsymbol{\delta}^{(t)}\|^2 \mid \mathcal{B}^{(t)}, \xi^{(t)} \right] \\
&\quad \leq (\sigma^{(t)})^2
\end{aligned} \tag{A.12}$$

where for the first inequality we have applied the Cauchy-Schwartz inequality, and for the second inequality we have used the non-expansiveness of the proximal operator of convex $h_n(\cdot)$. Taking expectation w.r.t. $\xi_{(t)}$ and then total expectation on both sides of

(A.11), we have

$$\mathbb{E}\left[\Phi(\boldsymbol{\theta}^{(t+1)})\right] - \mathbb{E}\left[\Phi(\boldsymbol{\theta}^{(t)})\right] \tag{A.13}$$
$$\leq \alpha^{(t)}(\sigma^{(t)})^2 + \left(\frac{L(\alpha^{(t)})^2}{2} - \frac{\alpha^{(t)}}{2}\right)\mathbb{E}\left[\left\|\boldsymbol{p}^{(t)}_{(\xi^{(t)})}\right\|^2\right].$$

Summing up through $p = 0$ to $p = t-1$, we have

$$\mathbb{E}\left[\Phi(\boldsymbol{\theta}^{(t)})\right] - \Phi(\boldsymbol{\theta}^{(0)}) \tag{A.14}$$
$$\leq \sum_{p=0}^{t} \alpha^{(p)}(\sigma^{(p)})^2 + \sum_{p=0}^{t}\left(\frac{L(\alpha^{(p)})^2}{2} - \frac{\alpha^{(p)}}{2}\right)\mathbb{E}\left[\left\|\boldsymbol{p}^{(p)}_{(\xi^{(p)})}\right\|^2\right].$$

Since we have assumed $\alpha^{(t)} < 1/L$, we have $\frac{L(\alpha^{(t)})^2}{2} - \frac{\alpha^{(t)}}{2} < 0$. Therefore, we have

$$\sum_{p=0}^{t}\left(\frac{\alpha^{(p)}}{2} - \frac{L(\alpha^{(p)})^2}{2}\right)\mathbb{E}\left[\left\|\boldsymbol{p}^{(p)}_{(\xi^{(p)})}\right\|^2\right].$$
$$\leq \Phi(\boldsymbol{\theta}^{(0)}) - \Phi(\boldsymbol{\theta}^{(\star)}) + \sum_{p=0}^{t}\alpha^{(p)}(\sigma^{(p)})^2 \tag{A.15}$$

Taking $t \to \infty$, and by the assumption that $\sum_{t=0}^{\infty}\alpha^{(t)}(\sigma^{(t)})^2 < \infty$, we can conclude that

$$\lim_{t\to\infty}\inf \mathbb{E}\left[\left\|\boldsymbol{p}^{(t)}_{(\xi^{(t)})}\right\|^2\right] = 0,$$

using Lemma 1.

One can see that

$$\mathbb{E}\left[\left\|\boldsymbol{P}^{(t)}_{(\xi^{(t)})}\right\|^2\right] \leq 2\mathbb{E}\left[\left\|\boldsymbol{p}^{(t)}_{(\xi^{(t)})}\right\|^2\right] + 2\mathbb{E}\left[\left\|\boldsymbol{p}^{(t)}_{(\xi^{(t)})} - \boldsymbol{P}^{(t)}_{(\xi^{(t)})}\right\|^2\right]$$
$$\leq 2\mathbb{E}\left[\left\|\boldsymbol{p}^{(t)}_{(\xi^{(t)})}\right\|^2\right]$$
$$+ 2\mathbb{E}\left[\mathbb{E}_{\zeta^{(t)}}\left[\left\|\boldsymbol{G}^{(t)}_{(\xi^{(t)})} - \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)})\right\|^2 \middle| \mathcal{B}^{(t)}, \xi^{(t)}\right]\right]$$
$$\leq 2\mathbb{E}\left[\left\|\boldsymbol{p}^{(t)}_{(\xi^{(t)})}\right\|^2\right] + 2(\sigma^{(r)})^2. \tag{A.16}$$

where the last inequality is obtained via applying the nonexpansive property again. Note that both terms on the right hand side converge to zero. Hence, this relationship implies that

$$\lim_{t\to\infty} \inf \mathbb{E}\left[\left\|\boldsymbol{P}^{(t)}_{(\xi^{(t)})}\right\|^2\right] = 0.$$

Note that by our sampling strategy, we have

$$\mathbb{E}\left[\left\|\boldsymbol{P}^{(t)}_{(\xi^{(t)})}\right\|^2\right] = \mathbb{E}\left[\mathbb{E}_{\xi^{(t)}}\mathbb{E}_{\zeta^{(t)}}\left[\left\|\boldsymbol{P}^{(t)}_{(\xi^{(t)})}\right\|^2 \middle| \mathcal{B}^{(t)}, \xi^{(t)}\right]\right].$$

However, since $\boldsymbol{P}^{(t)}_{(\xi^{(t)})}$ is not affected by the random seed $\zeta^{(t)}$, we have

$$\begin{aligned}\mathbb{E}\left[\left\|\boldsymbol{P}^{(t)}_{(\xi^{(t)})}\right\|^2\right] &= \mathbb{E}\left[\mathbb{E}_{\xi^{(t)}}\left[\left\|\boldsymbol{P}^{(t)}_{(\xi^{(t)})}\right\|^2 \middle| \mathcal{B}^{(t)}\right]\right] \\ &= \mathbb{E}\left[\sum_{n=1}^{N} \frac{1}{N}\left\|\boldsymbol{P}^{(t)}_{(n)}\right\|^2\right].\end{aligned}$$

This proves the proposition.

A.4 Proof of Proposition 3

The insight of the proof largely follows the technique for single-block `Adagrad` [31], with some careful modifications to multiple block updates. One will see that the block sampling strategy and the block-wise unbiased gradient estimation are key to apply the proof techniques developed in [31] to our case. To show convergence, let us first consider the following lemma:

Lemma 2 *[31] Let $a_0 > 0$, $a_i \geq 0$, $i = 1, \ldots, T$ and $\beta > 1$. Then, we have*

$$\sum_{t=1}^{T} \frac{a_t}{(a_0 + \sum_{i=1}^{t} a_i)^{\beta}} \leq \frac{1}{(\beta-1)a_0^{\beta-1}}.$$

The proof is simple and elegant; see [31, Lemma 4].

Lemma 3 *[31] Consider a random variable X. If $\mathbb{E}[X] < \infty$, then $\mathsf{Pr}(X < \infty) = 1$.*

Let us consider the block-wise again:

$$f(\boldsymbol{\theta}^{(t+1)}) \leq f(\boldsymbol{\theta}^{(t)}) + \left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}), \boldsymbol{A}_{(\xi^{(t)})}^{(t+1)} - \boldsymbol{A}_{(\xi^{(t)})}^{(t)} \right\rangle + \frac{L_{\xi^{(t)}}^{(t)}}{2} \left\| \boldsymbol{A}_{(\xi^{(t)})}^{(t+1)} - \boldsymbol{A}_{(\xi^{(t)})}^{(t)} \right\|^2. \tag{A.17}$$

Plugging in our update rule under `AdaCPD`, one can see that

$$\begin{aligned} f(\boldsymbol{\theta}^{(t+1)}) &\leq f(\boldsymbol{\theta}^{(t)}) + \left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}), -\boldsymbol{\eta}_{(\xi^{(t)})}^{(t)} \circledast \boldsymbol{G}_{(\xi^{(t)})}^{(t)} \right\rangle \\ &\quad + \frac{L_{\xi^{(t)}}^{(t)}}{2} \left\| \boldsymbol{\eta}_{(\xi^{(t)})}^{(t)} \circledast \boldsymbol{G}_{(\xi^{(t)})}^{(t)} \right\|^2 \\ &= f(\boldsymbol{\theta}^{(t)}) - \left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}), \boldsymbol{\eta}_{(\xi^{(t)})}^{(t)} \circledast \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}) \right\rangle \\ &\quad + \left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}), \boldsymbol{\eta}_{(\xi^{(t)})}^{(t)} \circledast \left(\nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}) - \boldsymbol{G}_{(\xi^{(t)})}^{(t)} \right) \right\rangle \\ &\quad + \frac{L_{\xi^{(t)}}^{(t)}}{2} \left\| \boldsymbol{\eta}_{(\xi^{(t)})}^{(t)} \circledast \boldsymbol{G}_{(\xi^{(t)})}^{(t)} \right\|^2 \end{aligned} \tag{A.18}$$

Taking expectation w.r.t. $\zeta^{(t)}$ (the random seed that is responsible for selecting fibers) conditioning on the filtration $\mathcal{B}^{(t)}$ and the selected block $\xi^{(t)}$, the middle term is zero—since the block stochastic gradient is unbiased [cf. Fact 2]. Hence, we have reached the following

$$\begin{aligned} \mathbb{E}_{\zeta^{(t)}} \left[f(\boldsymbol{\theta}^{(t+1)}) | \mathcal{B}^{(t)}, \xi^{(t)} \right] &\leq \mathbb{E}_{\zeta^{(t)}} \left[f(\boldsymbol{\theta}^{(t)}) | \mathcal{B}^{(t)}, \xi^{(t)} \right] \\ &\quad - \mathbb{E}_{\zeta^{(t)}} \left[\left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}), \boldsymbol{\eta}_{(\xi^{(t)})}^{(t)} \circledast \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}) \right\rangle | \mathcal{B}^{(t)}, \xi^{(t)} \right] \\ &\quad + \frac{L_{\xi^{(t)}}^{(t)}}{2} \mathbb{E}_{\zeta^{(t)}} \left[\left\| \boldsymbol{\eta}_{(\xi^{(t)})}^{(t)} \circledast \boldsymbol{G}_{(\xi^{(t)})}^{(t)} \right\|^2 \Big| \mathcal{B}^{(t)}, \xi^{(t)} \right] \end{aligned} \tag{A.19}$$

Taking total expectation on both sides, we have

$$\begin{aligned}\mathbb{E}\left[f(\boldsymbol{\theta}^{(t+1)})\right] &\leq \mathbb{E}\left[f(\boldsymbol{\theta}^{(t)})\right] \\ &\quad - \mathbb{E}\left[\left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}), \boldsymbol{\eta}^{(t)}_{(\xi^{(t)})} \circledast \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}) \right\rangle\right] \\ &\quad + \mathbb{E}\left[\frac{L^{(t)}_{\xi^{(t)}}}{2} \left\| \boldsymbol{\eta}^{(t)}_{(\xi^{(t)})} \circledast \boldsymbol{G}^{(t)}_{(\xi^{(t)})} \right\|^2\right]. \end{aligned} \tag{A.20}$$

From the above inequality and the assumption that $L^{(t)}_{(n)}$ is bounded from above by L, we can conclude that

$$\begin{aligned}&\sum_{t=1}^{T} \mathbb{E}\left[\left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}), \boldsymbol{\eta}^{(t)}_{(\xi^{(t)})} \circledast \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}) \right\rangle\right] \\ &\leq f(\boldsymbol{\theta}^{(0)}) - f(\boldsymbol{\theta}^{(\star)}) + \sum_{t=1}^{T} \frac{L}{2} \mathbb{E}\left[\left\| \boldsymbol{\eta}^{(t)}_{(\xi^{(t)})} \circledast \boldsymbol{G}^{(t)}_{(\xi^{(t)})} \right\|^2\right]\end{aligned}$$

which is by summing up all the inequalities in (A.19) from $t = 1$ to T.

Let us observe the last term on the right hand side and take $F \to \infty$:

$$\begin{aligned}&\mathbb{E}\left[\sum_{t=1}^{\infty} \left\| \boldsymbol{\eta}^{(t)}_{(\xi^{(t)})} \circledast \boldsymbol{G}^{(t)}_{(\xi^{(t)})} \right\|^2\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{\infty} \sum_{i=1}^{J_{\xi^{(t)}}} \sum_{r=1}^{R} \left[\boldsymbol{\eta}^{(t)}_{(\xi^{(t)})}\right]^2_{i,r} \left[\boldsymbol{G}^{(t)}_{(\xi^{(t)})}\right]^2_{i,r}\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{\infty} \sum_{i=1}^{J_{\xi^{(t)}}} \sum_{r=1}^{R} \left[\boldsymbol{\eta}^{(t+1)}_{(\xi^{(t)})}\right]^2_{i,r} \left[\boldsymbol{G}^{(t)}_{(\xi^{(t)})}\right]^2_{i,r}\right. \\ &\quad \left. + \sum_{t=1}^{\infty} \sum_{i=1}^{J_{\xi^{(t)}}} \sum_{r=1}^{R} \left(\left[\boldsymbol{\eta}^{(t)}_{(\xi^{(t)})}\right]^2_{i,r} - \left[\boldsymbol{\eta}^{(t+1)}_{(\xi^{(t)})}\right]^2_{i,r} \right) \left[\boldsymbol{G}^{(t)}_{(\xi^{(t)})}\right]^2_{i,r}\right].\end{aligned} \tag{A.21}$$

Note that we have exchanged the order of the limits and expectations, since the expectation is taking on nonnegative terms. Using Lemma 2, one can easily show the first

term above satisfies is bounded from above by

$$\frac{C_1}{2\epsilon\beta^{2\epsilon}}$$

where $0 < C_1 < \infty$ is a constant. To see the second term is also bounded, observe that

$$\begin{aligned}
&\sum_{t=1}^{\infty}\left(\left[\boldsymbol{\eta}^{(t)}_{(\xi^{(t)})}\right]^2_{i,r} - \left[\boldsymbol{\eta}^{(t+1)}_{(\xi^{(t)})}\right]^2_{i,r}\right)\left[\boldsymbol{G}^{(t)}_{(\xi^{(t)})}\right]^2_{i,r} \\
&\leq \max_{t\geq 0}\left[\boldsymbol{G}^{(t)}_{(\xi^{(t)})}\right]^2_{i,r}\sum_{t=1}^{\infty}\left(\left[\boldsymbol{\eta}^{(t)}_{(\xi^{(t)})}\right]^2_{i,r} - \left[\boldsymbol{\eta}^{(t+1)}_{(\xi^{(t)})}\right]^2_{i,r}\right) \\
&\leq \max_{t\geq 0}\left[\boldsymbol{G}^{(t)}_{(\xi^{(t)})}\right]^2_{i,r}\left[\boldsymbol{\eta}^{(t)}_{(\xi^{(t)})}\right]^2_{i,r} \\
&\leq \left[\boldsymbol{\eta}^{(t)}_{(\xi^{(t)})}\right]^2_{i,r}\left(\left[\nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)})\right]^2_{i,r}\right. \\
&\quad \left. + \left(\left[\nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)})\right]_{i,r} - \left[\boldsymbol{G}^{(t)}_{(\xi^{(t)})}\right]_{i,r}\right)^2\right)
\end{aligned} \tag{A.22}$$

Since we have assumed that $\boldsymbol{A}^{(t)}_{(n)}$'s are bounded, the right hand side is bounded from above. Therefore, we have reached the conclusion

$$\mathbb{E}\left[\sum_{t=1}^{\infty}\left\langle \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)}), \boldsymbol{\eta}^{(t)}_{(\xi^{(t)})} * \nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)})\right\rangle\right] < \infty.$$

Applying Lemma 3, one can see that

$$\mathsf{Pr}\left(\sum_{t=1}^{\infty}\left[\boldsymbol{\eta}^{(t)}_{(\xi^{(t)})}\right]_{i,r}\left[\nabla_{\boldsymbol{A}_{(\xi^{(t)})}} f(\boldsymbol{\theta}^{(t)})\right]^2_{i,r} < \infty\right) = 1.$$

Since $\mathsf{Pr}(\xi^{(t)} = n) > 0$, one immediate result is that any n appears infinitely many times in the sequence $t = 1, \ldots, \infty$, according to the second Borel-Cantelli lemma. This leads to

$$\mathsf{Pr}\left(\sum_{j=1}^{\infty}\left[\boldsymbol{\eta}^{(t_j(n))}_{(n)}\right]_{i,r}\left[\nabla_{\boldsymbol{A}_{(n)}} f(\boldsymbol{\theta}^{(t_j(n))})\right]^2_{i,r} < \infty\right) = 1,$$

holds for $n = 1, \ldots, N$, where $t_1(n), \ldots, t_j(n), \ldots$ is the subsequence of $\{t\}$ such that

block n is sampled for updating.

Hence, with probability one there exists a subsequence $t_1(n), \ldots, t_\infty(n)$ such that at the corresponding iterations block n is sampled for updating. It is not hard to show that

$$\sum_{j=1}^{\infty} \left[\boldsymbol{\eta}_{(n)}^{(t_j(n))}\right]_{i,r} = \infty,$$

by the assumption that $\boldsymbol{A}_{(n)}^{(t)}$ are all bounded. This directly implies that

$$\sum_{t=1}^{\infty} \left[\boldsymbol{\eta}_{(n)}^{(t)}\right]_{i,r} = \infty, \quad \forall n.$$

Hence, by Lemma 2, we have

$$\lim_{t\to\infty} \; [\nabla_{\boldsymbol{A}_{(n)}} f(\boldsymbol{\theta}^{(t)})]_{i,r}^2 = 0$$

with probability one.