STOCHASTIC OPTIMIZATION FOR COUPLED TENSOR DECOMPOSITION WITH APPLICATIONS IN STATISTICAL LEARNING

Shahana Ibrahim and Xiao Fu

School of Electrical Engineering and Computer Science
Oregon State University

ABSTRACT

Coupled tensor decomposition aims at factoring a number of tensors that share some of their latent factors. Existing algorithms for coupled *canonical polyadic decomposition* (CPD) face serious scalablity challenges, especially when the number of tensors is large. However, a large amount of coupled tensors naturally arise in timely applications such as statistical learning, e.g., when estimating the *joint* probability mass function (PMF) of many random variables from marginal PMFs. Stochastic algorithms that admit lightweight updates exist for coupled decomposition, but these algorithms cannot handle complex constraints (e.g., the probability simplex constraint that is important in statistical learning) due to their sampling patterns. This work puts forth a simple data-sampling and block variable-updating strategy for simultaneously factoring a large number of coupled tensors. The proposed algorithm enjoys low per-iteration complexity and can easily handle constraints on latent factors. We also show that this multi-block algorithm admits a nice connection to the classic single-block stochastic proximal gradient (SPG), and thus it naturally inherits convergence properties of SPG. Synthetic and real-data experiments show that the proposed algorithm is very promising for statistical learning problems.

***Index Terms*—** Coupled canonical polyadic decomposition, stochastic optimization, statistical learning

1. INTRODUCTION

Tensors with shared latent factors (or, *coupled tensors*) arise in many fields across signal processing, machine learning, and data analytics. For example, [1] uses coupled tensor decomposition for array processing and harmonic retrieval. The work in [2] factors tensors and matrices together to enhance performance of recommender systems. The work in [3] proposes a joint probability mass function learning framework by decomposing a large number of coupled tensors. Another recent work formulates the crowdsourcing (ensemble learning) problem in machine learning as a coupled decomposition problem [4].

Tensor decomposition under a certain model (e.g., canonical polyadic decomposition (CPD) [5] and Tucker [6]) in general poses very hard optimization problems. Coupling tensors together increases the computational difficulty in practice. Classic methods mostly focus on a small number of tensors (e.g., two tensors) who share one or two latent factors [7–10]. For such cases, many algorithms for single tensor decomposition can be easily extended to handle the coupled decomposition problem. However, in some problems like statistical learning, a large amount of coupled tensors can arise [3, 4]. For example, consider a case where $K = 256$ discrete random variables and one wishes to estimate the joint probability mass function (PMF). Then, by the coupled tensor formulation in [3], we have $\binom{K}{3} = 2,763,520$ different possible third-order tensors to jointly decompose. This number is prohibitively large. On the other hand, in machine learning problems, e.g., classification, K corresponds to the number of features—which can easily reach thousands.

Jointly decomposing a large number of latent factor-coupled tensors poses very serious complexity challenges, and the classic single tensor decomposition algorithms such as *alternating least squares* (ALS) may no longer prevail. In fact, ALS-based methods were used in [3, 4] to tackle their respective statistical learning problems. However, both algorithms can only handle a relatively small number of coupled tensors. This means that the number of random variables that can be involved in their statistical inference problems is very limited, thereby raising questions about scalability of coupled tensor-based statistical learning. Stochastic gradient-based coupled tensor decomposition is conceptually suitable for handling such cases since they work with sampled data and admit low per-iteration complexity [9, 11]. Nevertheless, these methods usually cannot effectively handle constraints that are of interest in statistical learning (e.g., probability simplex constraints), due to their sampling patterns.

This work puts forth a simple yet effective coupled tensor decomposition algorithm, tailored for large-scale statistical learning problems such as joint PMF learning [3] and ensemble learning [4]. Our idea is based on a two-stage sampling paradigm. Specifically, the method first samples a latent factor to update, and then samples a tensor that contains this factor—and performs one-step proximal gradient (PG) with respect to (w.r.t.) the chosen factor using the selected data. This way, all the operations involved are lightweight in terms both memory and computational flops. The algorithm is reminiscent of block randomized *block coordinate descent* (BCD) and *stochastic proximal gradient* (SPG), and thus can easily handle complex constraints and regularizations. We also show that the algorithm is essentially a single-block SPG algorithm with a scaled version of an unbiased stochastic oracle, thereby enjoying all convergence properties of the classic SPG. Synthetic and real-data experiments on a couple of stochastic learning problems show the proposed method is very promising.

2. BACKGROUND

2.1. A Closer Look at The Motivating Examples

• **Joint PMF Learning:** In [3], Kargas *et al.* established an intriguing link between coupled CPD and joint PMF estimation of a large number of random variables. Consider a joint PMF $\mathsf{Pr}(Z_1 = i_1, \ldots, Z_K = i_K)$, where Z_k can take I_k different values. This can

Email: (ibrahish,xiao.fu)@oregonstate.edu. This work was supported in part by National Science Foundation under Projects NSF ECCS-1808159 and NSF ECCS-1608961.

be represented as a Kth-order tensor $\underline{\boldsymbol{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_K}$ with

$$\underline{\boldsymbol{X}}(i_1, \ldots, i_K) = \mathsf{Pr}(Z_1 = i_1, \ldots, Z_K = i_K).$$

Note that since every nonnegative tensor admits a nonnegative CPD [3, 6], we can *represent* the joint PMF as

$$\underline{\boldsymbol{X}}(i_1, \ldots, i_K) = \sum_{f=1}^{F} \prod_{k=1}^{K} \boldsymbol{\lambda}(f) \boldsymbol{A}_k(i_k, f), \tag{1}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^F$ and $\boldsymbol{A}_k \in \mathbb{R}^{I_k \times F}$, and all the latent factors (i.e., $\boldsymbol{A}_k$'s and $\boldsymbol{\lambda}$) are nonnegative. Note that one can also assume that the columns of $\boldsymbol{A}_k$'s have unit 1-norm, since $\boldsymbol{\lambda}$ can 'absorb' the scalings. Note that we also have $\mathbf{1}^\top \boldsymbol{\lambda} = 1$ and $\boldsymbol{\lambda} \geq \mathbf{0}$ since the tensor is a joint PMF [3]. Interestingly, the CPD representation can be considered as a *naive Bayesian* model, i.e., $\mathsf{Pr}(Z_1 = i_1, \ldots, Z_K = i_K) = \sum_{f=1}^{F} \prod_{k=1}^{K} \mathsf{Pr}(H = f) \mathsf{Pr}(Z_k = i_k | H = f)$, where we have

$$\mathsf{Pr}(H = f) = \boldsymbol{\lambda}(f), \quad \mathsf{Pr}(Z_k = i_k | H = f) = \boldsymbol{A}_k(i_k, f).$$

In practice, the joint PMF is of great interest for learning and inferences. However, it is impossible to directly estimate the joint PMF of a large number of random variables, since the sample complexity is prohibitive. However, marginal PMFs of two or three random variables are much easier to estimate. The idea of [3] is to estimate $\boldsymbol{A}_k$'s and $\boldsymbol{\lambda}$ from marginal PMFs of three variables—and then using the latent factors to construct the joint PMF. This is enabled by the link between the naive Bayesian model and high-order tensors. Specifically, marginalizing $\mathsf{Pr}(Z_1, \ldots, Z_K)$ to $\mathsf{Pr}(Z_\ell, Z_m, Z_n)$ is equivalent to 'collapsing' $K - 3$ modes of the tensor $\underline{\boldsymbol{X}}$, leading to [3] $\mathsf{Pr}(X_\ell = i_\ell, X_m = i_m, X_n = i_n) = \sum_{f=1}^{F} \boldsymbol{\lambda}(f) \boldsymbol{A}_\ell(i_\ell, f) \boldsymbol{A}_m(i_m, f) \boldsymbol{A}_n(i_n, f)$. Hence, the joint PMF learning problem boils down to jointly factoring

$$\underline{\boldsymbol{X}}_{\ell,m,n} = [\![\boldsymbol{\lambda}, \boldsymbol{A}_\ell, \boldsymbol{A}_m, \boldsymbol{A}_n]\!],$$

for all available ℓ, m, n, where we use $[\![\boldsymbol{\lambda}, \boldsymbol{A}_\ell, \boldsymbol{A}_m, \boldsymbol{A}_n]\!]$ to denote a third-order tensor with column normalized latent factors $\boldsymbol{A}_\ell$, $\boldsymbol{A}_m$, and $\boldsymbol{A}_n$, and $\boldsymbol{\lambda}$, i.e., the 'weights' of the rank-one components. The optimization problem is as follows:

$$\begin{aligned} \underset{\{\boldsymbol{A}_k\}_{k=1}^K, \boldsymbol{\lambda}}{\text{minimize}} \; & \sum_{\ell=1}^{K} \sum_{m=\ell+1}^{K} \sum_{n=m+1}^{K} \left\| \underline{\boldsymbol{X}}_{\ell,m,n} - [\![\boldsymbol{\lambda}, \boldsymbol{A}_\ell, \boldsymbol{A}_m, \boldsymbol{A}_n]\!] \right\|_F^2 \\ \text{subject to } & \mathbf{1}^\top \boldsymbol{A}_k = \mathbf{1}^\top, \; \boldsymbol{A}_k \geq \mathbf{0}, \; \forall k \\ & \mathbf{1}^\top \boldsymbol{\lambda} = 1, \; \boldsymbol{\lambda} \geq \mathbf{0}. \end{aligned} \tag{2}$$

• **Crowdsourcing:** In machine learning, data labeling is oftentimes crowdsourced to multiple annotators for efficiency and robustness. Since different annotators may create different labels for the same sample, an effective algorithm for result fusion is desired. The so-called Dawid-Skene method was proposed in 1979 for this purpose [12]. Consider a classification problem. Suppose that there are K annotators who label the data samples $\{\boldsymbol{f}_q\}_{q=1}^Q$ and provide a class label from $\{1, \ldots, F\}$. Let Z_k represent the response of the annotator k to a data sample. The goal is to estimate the true label of any data sample $\boldsymbol{f}_q$ from the annotator responses. The Dawid-Skene model assumes that given the ground-truth label, the responses of the annotators are conditionally independent, i.e., $\mathsf{Pr}(Z_1 = i_1, \ldots, Z_K = i_K | Y = f) = \sum_{f=1}^{F} \mathsf{Pr}(Y = f) \prod_{k=1}^{K} \mathsf{Pr}(Z_k = i_k | Y = f)$, where $f \in \{1, \ldots, F\}$ represents the class label, and i_k denotes the response of the kth annotator. If one defines a series of matrices $\boldsymbol{A}_k \in \mathbb{R}^{F \times F}$ and let $\boldsymbol{A}_k(i_k, f) := \mathsf{Pr}(Z_k = i_k | Y = f)$, $\boldsymbol{A}_f \in \mathbb{R}^{K \times K}$ can be considered as the 'confusion matrix' of annotator k. Also define the vector $\boldsymbol{\lambda} \in \mathbb{R}^K$ such that $\boldsymbol{\lambda}(f) := \mathsf{Pr}(Y = f)$; i.e., the prior probability of the ground-truth label Y. Then the crowdsourcing problem amounts to estimating $\boldsymbol{A}_k$ for $k = 1, \ldots, K$ and $\boldsymbol{\lambda}$. By observing the co-occurrences of different annotators, tensors following the form of $\underline{\boldsymbol{X}}_{\ell,m,n} = [\![\boldsymbol{\lambda}, \boldsymbol{A}_\ell, \boldsymbol{A}_m, \boldsymbol{A}_n]\!]$ can be constructed. Then, the problem of estimating the label prior and the confusion matrices becomes the same problem as in (2) again, with the same constraints as in the previous example.

2.2. Challenges, Prior Art

The idea of the algorithms in [3, 4] for handling Problem (2) can be summarized as follows. The algorithm cyclically updates each of $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_K, \boldsymbol{\lambda}$ when fixing the others. When updating $\boldsymbol{A}_k$, the subproblem can be written as a *constrained least squares* problem, i.e.,

$$\underset{\substack{\mathbf{1}^\top \boldsymbol{A}_k = \mathbf{1}^\top \\ \boldsymbol{A}_k \geq \mathbf{0}}}{\text{minimize}} \; \sum_{m \neq k} \sum_{\substack{n \neq k \\ n > m}} \| \boldsymbol{X}^{(1)}_{k,m,n} - (\boldsymbol{A}_n \odot \boldsymbol{A}_m) \boldsymbol{D} \boldsymbol{A}_k^\top \|_F^2, \tag{3}$$

where $\boldsymbol{D} = \mathrm{Diag}(\boldsymbol{\lambda})$ and $\odot$ denotes the Khatri-Rao product. The above is nothing but fitting a least squares model to a collection of tensors that share $\boldsymbol{A}_k$, and $\boldsymbol{X}^{(1)}_{k,m,n}$ is the *mode-1 unfolding* of the tensor $\underline{\boldsymbol{X}}_{k,m,n}$; see definition of tensor unfolding in [6]. The problem in (3) is convex and thus many off-the-shelf algorithms can be employed. The works in [3, 4] use an ADMM algorithm to handle it. The challenge is that when K is large, some key steps in the ADMM algorithm involve a large number of flops. In particular, consider the following step in the ADMM algorithm proposed in [3]:

$$(\boldsymbol{\lambda} \boldsymbol{\lambda}^\top) \sum_{m \neq k} \sum_{\substack{n \neq k \\ n > m}} (\boldsymbol{A}_n \odot \boldsymbol{A}_m)^\top \boldsymbol{X}^{(1)}_{k,m,n}. \tag{4}$$

This step needs $O\left(\binom{K}{2} I_k I_m I_n F\right)$ flops. In statistical learning problems such as classification, both K (number of random variables/features) and I (number of values each feature can take) can be very large. Consider a case where $K \approx I_\ell = I$, the complexity of (4) easily goes to the level of $O(I^5)$—which makes the algorithm quite slow even when I is moderate. In fact, computation of terms like $(\boldsymbol{A}_n \odot \boldsymbol{A}_m)^\top \boldsymbol{X}^{(1)}_{k,m,n}$ has been a bottleneck for even single tensor decomposition. This so-called *multiplication of tensor and Khatri-Rao product* (MTTKRP) operation [5, 6] is now repeated $\binom{K}{2}$ times in the coupled decomposition of interest, which worsens the complexity issue substantially.

Random sampling and stochastic gradient (SGD) have been considered for single/coupled tensor decomposition [7, 9, 13]. SGD is known for its lightweight per-iteration update, and is a good fit for tackling Problem (2). However, existing stochastic coupled tensor factorization methods sample part of the tensors (e.g., random entries $\underline{\boldsymbol{X}}_{\ell,m,n}(i_\ell, i_m, i_n)$) to update the latent factors, which is not suitable for the problem in (2). The reason is that random entries like $\underline{\boldsymbol{X}}_{\ell,m,n}(i_\ell, i_m, i_n)$ only contain information of the rows of $\boldsymbol{A}_k$'s, e.g., $\boldsymbol{A}_\ell(i_\ell, :)$, $\boldsymbol{A}_m(i_m, :)$, and $\boldsymbol{A}_n(i_n, :)$—but the statistical learning problems have constraints on the *columns* of $\boldsymbol{A}_k$'s (i.e., $\mathbf{1}^\top \boldsymbol{A}_k = \mathbf{1}^\top$). Therefore, new stochastic algorithms are desired for the problem of interest.

3. PROPOSED ALGORITHM

Our idea for handling (2) is natural yet simple. Instead of working with all the tensors all the time, we work with a randomly sampled tensor $\underline{\boldsymbol{X}}_{k,m,n}$ to update $\boldsymbol{A}_k$. This way, the computing many

Algorithm 1: `Proposed`

input : tensors $\{\underline{\boldsymbol{X}}_{\ell,m,n} \in \mathbb{R}^{I_\ell \times I_m \times I_n}\}_{\ell,m,n}$; rank F; initialization $\{\boldsymbol{A}_k^{(0)}\}$

$t \leftarrow 0$;
repeat
  sample k from $\{1,\dots,K+1\}$;
  sample (m,n) from $\{1,\dots,K\}$;
  if $k \leq K$ **then**
    form $\boldsymbol{H}_k = (\boldsymbol{A}_n \odot \boldsymbol{A}_m)\boldsymbol{D}$,
    $\boldsymbol{V}_k = (\boldsymbol{\lambda}\boldsymbol{\lambda}^\top) \circledast (\boldsymbol{A}_n^\top \boldsymbol{A}_n) \circledast (\boldsymbol{A}_m^\top \boldsymbol{A}_m)$;
    $\boldsymbol{G}_k \leftarrow \left(\boldsymbol{A}_k^{(t)} \boldsymbol{H}_k^\top \boldsymbol{H}_k - (\boldsymbol{X}_{k,m,n}^{(1)})^\top \boldsymbol{H}_k\right)$;
    $\boldsymbol{A}_k^{(t+1)} \leftarrow \mathsf{Proj}\left(\boldsymbol{A}_k^{(t)} - \alpha^{(t)} \boldsymbol{G}_k\right)$;
    $\boldsymbol{A}_\ell^{(t+1)} \leftarrow \boldsymbol{A}_\ell^{(t)}$ for $\ell \neq k$, $\boldsymbol{\lambda}^{(t+1)} \leftarrow \boldsymbol{\lambda}^{(t)}$;
    else
      sample (ℓ,m,n) from $\{1,\dots,K\}$;
      form $\boldsymbol{H}_k = \boldsymbol{A}_n \odot \boldsymbol{A}_m \odot \boldsymbol{A}_\ell$;
      $\boldsymbol{G}_{K+1} \leftarrow \left(\boldsymbol{H}_k^\top \boldsymbol{H}_k \boldsymbol{\lambda} - \boldsymbol{H}_k^\top \mathrm{vec}(\boldsymbol{X}_{k,m,n}^{(1)})\right)$;
      $\boldsymbol{\lambda}^{(t+1)} \leftarrow \mathsf{Proj}\left(\boldsymbol{\lambda}^{(t)} - \alpha^{(t)} \boldsymbol{G}_{K+1}\right)$;
      $\boldsymbol{A}_k^{(t+1)} \leftarrow \boldsymbol{A}_k^{(t)}, \forall k$;
    end
  end
  $t \leftarrow t+1$;
until *some stopping criterion is reached*;

output: $\{\boldsymbol{A}_k^{(t)}\}_{k=1}^K$

MTTKRPs at a single iteration can be avoided. Specifically, we propose the following two-level sampling strategy: At each iteration, we first sample a block variable to update; i.e., we sample $k \in \{1,\dots,K,K+1\}$, where the first K blocks correspond to the $\boldsymbol{A}_k$'s and the $(K+1)$th block the $\boldsymbol{\lambda}$. Then, we sample m,n from $\{1,\dots,k-1,k+1,\dots,K\}$ and update $\boldsymbol{A}_k$ using the following:

$$\boldsymbol{A}_k^{(t+1)} \leftarrow \mathsf{Proj}\left(\boldsymbol{A}_k^{(t)} - \alpha^{(t)} \boldsymbol{G}_k^{(t)}\right).$$

where $\boldsymbol{G}_k^{(t)} = \boldsymbol{A}_k^{(t)} \boldsymbol{V}_k - (\boldsymbol{X}_{k,m,n}^{(1)})^\top \boldsymbol{H}_k$, $\boldsymbol{H}_k = (\boldsymbol{A}_n \odot \boldsymbol{A}_m)\boldsymbol{D}$, $\boldsymbol{V}_k = (\boldsymbol{\lambda}\boldsymbol{\lambda}^\top) \circledast (\boldsymbol{A}_n^\top \boldsymbol{A}_n) \circledast (\boldsymbol{A}_m^\top \boldsymbol{A}_m)$, $\mathsf{Proj}(\boldsymbol{Z})$ projects the columns of $\boldsymbol{Z}$ onto the probability simplex, and $\circledast$ denotes the Hadamard product. The above is essentially a stochastic proximal gradient with respect to $\boldsymbol{A}_k$. Note that the projection step is important since it keeps the columns of $\boldsymbol{A}_k$ in the feasible set, which is particularly meaningful for statistical learning problems. We also let $\boldsymbol{A}_j^{(t+1)} \leftarrow \boldsymbol{A}_j^{(t)}$, $\forall j \neq k$ and $\boldsymbol{\lambda}^{(t+1)} \leftarrow \boldsymbol{\lambda}^{(t)}$. In addition, we let $\boldsymbol{G}_{k'}^{(t)} = \boldsymbol{0}$, $\forall k' \neq k$ and $\boldsymbol{g}^{(t)} = [\mathrm{vec}(\boldsymbol{G}_1^{(t)})^\top, \dots, \mathrm{vec}(\boldsymbol{G}_{K+1}^{(t)})^\top]^\top$ as the *overall stochastic oracle* in iteration t—which has an interesting connection to the gradient, as we will show shortly.

The vector $\boldsymbol{\lambda}$ can be updated in a very similar fashion. The algorithm is summarized in Algorithm 1. The proposed algorithm is naturally more lightweight compared to the algorithms in [3,4] since it only works with partial data in each iteration.

One remark is that the proposed two-layer sampling strategy makes sure that the information about the entire picked $\boldsymbol{A}_k$ (or $\boldsymbol{\lambda}$) is contained in the sampled data, and thus constraints on these latent factors can be easily imposed in the iterations. This is important, especially for real-data problems - since prior information can improve identification accuracy under severe noise or model mismatch.

As we will see, the proposed algorithm enjoys very competitive runtime performance. However, since in every iteration we only work with sampled data and a small portion of the optimization variables, a natural question is, does the algorithm even converge? To answer this question, we first show the following:

Proposition 1 *Denote the objective function in* (2) *as* $f(\boldsymbol{\theta})$ *where* $\boldsymbol{\theta} = [\boldsymbol{\theta}_1^\top, \dots, \boldsymbol{\theta}_K^\top]^\top$, $\boldsymbol{\theta}_k = \mathrm{vec}(\boldsymbol{A}_k)$ *for* $k = 1,\dots,K$ *and* $\boldsymbol{\theta}_{K+1} = \boldsymbol{\lambda}$. *Let* J_k *denote the number of available tensors whose mode-1 factor is* $\boldsymbol{A}_k$. *Also let* $\mathcal{B}^{(t)}$ *be the filtration up to iteration* $t-1$. *Then, by uniform sampling of the tensors, the gradient computed at iteration* t, $\boldsymbol{G}_k^{(t)}$ *satisfies* $\mathbb{E}\left[\boldsymbol{G}_k^{(t)} \mid \mathcal{B}^{(t)}\right] = C_k \nabla_{\boldsymbol{\theta}_k} f(\boldsymbol{\theta})$, $\forall k$ *where* $C_k > 0$ *is a certain constant.*

Proof: For any $k = 1, ..., K$, we have the following conditional expectation:

$$\begin{aligned}
\overline{\boldsymbol{G}}_k^{(t)} &= \mathbb{E}\left[\boldsymbol{G}_k^{(t)} | \mathcal{B}^{(t)}\right] = \mathbb{E}\left[\boldsymbol{G}_k^{(t)} | \{\boldsymbol{\theta}_k^{(t-1)}\}_{k=1}^K\right] \\
&= \mathbb{E}_{k'}\left[\frac{1}{J_{k'}} \sum_{m \neq k'} \sum_{\substack{n \neq k' \\ n > m}} \left(\boldsymbol{A}_{k'}^{(t-1)} \boldsymbol{V}_{k'} - (\boldsymbol{X}_{k',m,n}^{(1)})^\top \boldsymbol{H}_{k'}\right)\right] \\
&\overset{(a)}{=} \sum_{k'=1}^K \frac{\delta(k-k')}{(K+1)J_{k'}} \sum_{m \neq k'} \sum_{\substack{n \neq k' \\ n > m}} \left(\boldsymbol{A}_{k'}^{(t-1)} \boldsymbol{V}_{k'} - (\boldsymbol{X}_{k',m,n}^{(1)})^\top \boldsymbol{H}_{k'}\right) \\
&= C_k \sum_{m \neq k} \sum_{\substack{n \neq k \\ n > m}} \left(\boldsymbol{A}_k^{(t-1)} \boldsymbol{V}_k - (\boldsymbol{X}_{k,m,n}^{(1)})^\top \boldsymbol{H}_k\right) \qquad (5)
\end{aligned}$$

where $C_k = \frac{1}{(K+1)J_k}$ and $\delta(\cdot)$ is the Dirac function and the expectation in *(a)* is taken over the possible values of k. The last equality shows that $\overline{\boldsymbol{G}}_k^{(t)}$ is a scaled version of the gradient of the objective function of (3) taken w.r.t. $\boldsymbol{A}_k^{(t)}$. In the same fashion, it can be shown that $\overline{\boldsymbol{G}}_{K+1}^{(t)}$ is a scaled version of the gradient of the objective function (3) taken w.r.t. $\boldsymbol{\lambda}^{(r)}$. □

The proof is straightforward and shares the idea for stochastic single tensor decomposition in [14]. In spite of its simplicity, the implication is interesting—it means that by using such two-stage sampling, we are effectively applying stochastic proximal gradient to the original problem using an unbiased stochastic oracle. By this connection, all the convergence properties of the single-block stochastic proximal gradient (SPG) algorithm hold for the proposed block optimization algorithm. One caveat for nonconvex SPG is that, to ensure convergence to a stationary point, the variance should converge to zero along with the number of iterations [15]. Many variance reduction methods exist for circumventing this issue, e.g., SVRG-based methods [16–18] and growing batch size-based methods [15]. Nevertheless, in our experiments, we found that the algorithm works very well even without variance reduction.

Another key consideration in stochastic optimization is stepsize scheduling, i.e., determining $\alpha^{(t)}$ for each iteration. SPG normally works under the Robbins-Monroe rule, i.e., $\sum_{t=0}^\infty \alpha^{(t)} = \infty$ and $\sum_{t=0}^\infty (\alpha^{(t)})^2 < \infty$. In practice, practitioner may need more detailed guidance. In this work, we use the *Adagrad* rule as proposed in [14, 19].

4. EXPERIMENTS AND CONCLUSION

In this section, we use both synthetic and real data to showcase the effectiveness of the proposed algorithm. We use the block coordinate descent (BCD)-based algorithm developed in [3] to serve as the baseline, since it solves the same stochastic learning problem. For the real data part, the Kullback-Leibler (KL) divergence-cost version of the algorithm developed in [20] is also used as benchmark.

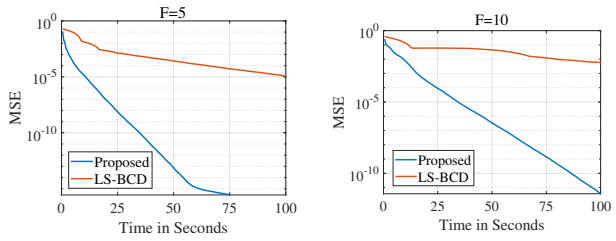

Fig. 1. Convergence of the proposed algorithm and the baseline for $N = 20, I_n = 15$ for different values of F

The two algorithms are denoted as `LS-BCD` and `KL-BCD`, respectively. For real data experiments in crowdsoursing, the algorithms in [4] and [21] are used as baselines and are denoted as `LS-ADMM` and `S-D&S`, respectively. `LS-ADMM` is based on a coupled tensor and matrix decomposition formulation, and `S-D&S` employs a symmetric orthogonal tensor decomposition technique to initialize the Dawid-Skene algorithm [12].

• **Synthetic Data** We consider a joint PMF recovery problem as described in Sec. 2. In this case, we have $K = 20$ random variables with each variable taking $I_k = 15$ discrete values. The rank of the joint PMF tensor is set to different values $F \in \{5, 10\}$. The columns of the conditional PMF matrices (factor matrices) $\boldsymbol{A}_k \in \mathbb{R}^{I_k \times F}$ and the prior probability vector $\boldsymbol{\lambda} \in \mathbb{R}^F$ are drawn from the probability simplex uniformly at random. We assume that the third order statistics of the random variables $\underline{\boldsymbol{X}}_{\ell,m,n} = [\![\boldsymbol{\lambda}, \boldsymbol{A}_\ell, \boldsymbol{A}_m, \boldsymbol{A}_n]\!]$, $\forall \ell, m, n \in [K]$ are available. In this case, we have $1,140$ third-order tensors. The average *Mean Squared Error* (MSE) of the factor matrices is computed as the performance metric. We stop all the algorithms when the relative change of their respective cost function becomes smaller than 10^{-4}.

The result is shown in Fig. 1. One can see that the algorithm outperforms the deterministic BCD algorithm in [3] in both accuracy and runtime by very large margin. In particular, when $F = 10$, the proposed method reaches MSE $\approx 10^{-4}$ after 25 seconds, while the deterministic algorithm has a MSE $\approx 10^{-1}$ at the same time. The performance gap is even larger when $F = 5$.

• **Real Data - Classification** In this case, different UCI datasets [1] are used to evaluate the classification performance using the proposed approach. Each dataset consists of feature vectors and the corresponding true labels for the data samples. Most features are continuous-valued, and we follow [3] to discretize the features values, and use $I_{\mathsf{ave}} = (1/K)\sum_{k=1}^{K} I_k$ to denote the averaged alphabet size of the discretized features for each dataset in Table 1. For each dataset, we run 10 Monte Carlo simulations by randomly partitioning the dataset into training, validation and testing sets. Using the training dataset, $\underline{\boldsymbol{X}}_{\ell,m,m}$ are estimated via counting the co-occurrences of the values taken by features ℓ, m and n. After identifying the parameters $\boldsymbol{A}_k$ and $\boldsymbol{\lambda}$, we use the *maximum a posteriori* (MAP) predictor to estimate the labels for each run and average the results. For each dataset, F is chosen by observing classification accuracy on the validation set.

The average runtime and the classification accuracy using different algorithms are given in Table 1. For all the datasets, one can see that the proposed method outperforms the other baselines in both accuracy and runtime. This is consistent to our observations in the synthetic experiments. In particular, for a challenging case where $K = 21$ (i.e., the Mushroom data), one can see that the proposed method is 8 times faster compared to the deterministic algorithm in [3] that solves the same optimization problem, with similar classification errors. Furthermore, the proposed algorithm is 47 times faster than the KL-objective function based algorithm in [20], without compromising accuracy.

[1] https://archive.ics.uci.edu/ml/datasets.html

Table 1. Real data-classification results using UCI dataset

	Misclassification(%)			Runtime(seconds)		
UCI Dataset (K, I_{avg}, F)	`Proposed`	`LS-BCD`	`KL-BCD`	`Proposed`	`LS-BCD`	`KL-BCD`
Nursery (9,4,15)	**0.086**	0.087	0.094	**2.98**	9.85	8.34
Car (7,4,15)	**0.097**	0.107	0.1068	**4.29**	14.79	7.725
Adult (15,14, 15)	**0.187**	0.247	†	**6.57**	48.49	†
Connect4 (22,7,15)	**0.338**	0.363	0.356	**6.54**	52.47	389.22
Credit (15,10,10)	**0.189**	0.347	0.254	**5.22**	40.02	30.51
Heart (9,3,10)	**0.198**	0.213	0.2113	**2.02**	8.87	8.11
Mushroom (21,6,15)	**0.042**	0.043	0.043	**8.19**	69.95	378.67
Voters (17,2,15)	**0.045**	0.076	0.053	**3.87**	27.44	27.64

† means the algorithm does not converge in 500 sec. and the result is not meaningful.

• **Real Data - Crowdsourcing** In this experiment, we create crowdsourcing data as follows: We use $K = 10$ different classification algorithms from the MATLAB machine learning toolbox such as various k-nearest neighbour classifiers, support vector machine classifiers and decision tree classifiers to serve as annotators. Using 20% of the available data samples, each annotator is trained. Then, we allow the annotators to label the unseen data samples with probability p. Setting $p < 1$ is equivalent to the practical scenario where not all data samples are annotated by an annotator. Once the annotator responses are available, we estimate the co-occurrences of the annotator responses ℓ, m and n to obatin $\underline{\boldsymbol{X}}_{\ell,m,n}$. Following this, the confusion matrices and prior probabilities are estimated via solving (2). Again, a MAP predictor for true labels can be built after obtaining $\boldsymbol{A}_k$'s and $\boldsymbol{\lambda}$. We perform 10 trials to take the average of the results and in each trial, a randomly selected testing set is labeled by the annotators with probability p. In our experiments, we set $p = 0.2$ for all annotators.

The performance of the algorithms are shown in Table 2. Similar to the classification case, the results show that in most of the cases, the proposed algorithm exhibits better performance compared to the baselines, both in accuracy and runtime.

Table 2. Real data-crowdsouring results using UCI dataset

	Misclassification(%)			Runtime(seconds)		
UCI Dataset (K,F)	`Proposed`	`LS-ADMM`	`S-D&S`	`Proposed`	`LS-ADMM`	`S-D&S`
Adult (10,2)	**0.182**	0.258	0.238	**0.19**	4.17	2.10
Connect4(10,3)	**0.273**	0.344	0.333	**0.72**	50.96	14.38
Credit (10,2)	**0.166**	0.175	**0.166**	**0.18**	0.45	1.49
Mushroom (10,2)	**0.061**	0.064	**0.061**	**0.18**	0.44	2.40

In this work, we proposed a stochastic sampling and optimization strategy for coupled tensor decomposition, tailored for statistical learning problems. The algorithm can handle a large number of latent factor-coupled tensors and can easily deal with a variety of constraints on the latent factors. Although the algorithm works with partial data and updates only a block of the optimization variables, we showed that the algorithm admits an interesting connection to the classic single-block stochastic proximal gradient scheme—thereby enjoying the same convergence properties. Simulations and real experiments showed that the proposed algorithm outperforms deterministic BCD algorithms devised for the same problems in both runtime and accuracy.

5. REFERENCES

[1] M. Sørensen and L. De Lathauwer, "Multidimensional harmonic retrieval via coupled canonical polyadic decomposition-part i: Model and identifiability," *IEEE Trans. Signal Process.*, vol. 65, no. 2, pp. 517–527, Jan 2017.

[2] D. Choi, J. Jang, and U. Kang, "Fast, accurate, and scalable method for sparse coupled matrix-tensor factorization," *Computing Research Repository*, Aug 2017.

[3] N. Kargas, N. D. Sidiropoulos, and X. Fu, "Tensors, learning, and kolmogorov extension for finite-alphabet random vectors," *IEEE Trans. Signal Process.*, vol. 66, pp. 4854–4868, Jul 2018.

[4] P. Traganitis, A. Pages-Zamora, and G. B. Giannakis, "Blind multiclass ensemble classification," *IEEE Trans. Signal Process.*, vol. 66, pp. 4737–4752, Jul 2018.

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

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

[7] N. Vervliet and L. De Lathauwer, "A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors," *IEEE J. Sel. Topics Signal Process.*, vol. 10, no. 2, pp. 284–295, Jan.

[8] M. Sørensen and L. De Lathauwer, "Coupled tensor decompositions for applications in array signal processing," in *Proc CAMSAP 2013*, Dec 2013, pp. 228–231.

[9] A. Beutel, P. P. Talukdar, A. Kumar, C. Faloutsos, E. E. Papalexakis, and E. P. Xing, "Flexifact: Scalable flexible factorization of coupled tensors on hadoop," in *Proc. SIAM SDM 2014*, 2014, pp. 109–117.

[10] C. I. Kanatsoulis, X. Fu, N. D. Sidiropoulos, and W.-K. Ma, "Hyperspectral super-resolution: A coupled tensor factorization approach," *IEEE Trans. Signal Process.*, vol. 66, no. 24, pp. 6503–6517, 2018.

[11] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, pp. 223–311, 2018.

[12] A. P. Dawid and A. M. Skene, "Maximum likelihood estimation of observer error-rates using the em algorithm," *Applied Statistics*, pp. 20–28, 1979.

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

[14] X. Fu, C. Gao, H.-T. Wai, and K. Huang, "Block-randomized stochastic proximal gradient for low-rank tensor factorization," *arXiv:1901.05529*, Jan 2019.

[15] S. Ghadimi and G. Lan, "Stochastic first- and zeroth-order methods for nonconvex stochastic programming," *SIAM Journal on Optimization*, vol. 23, no. 4, pp. 2341–2368, 2013.

[16] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Proc. NIPS*, Jan 2013, pp. 315–323.

[17] L. Xiao and T. Zhang, "A proximal stochastic gradient method with progressive variance reduction," *SIAM Journal on Optimization*, vol. 24, pp. 2057–2075, 2014.

[18] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Proc. NIPS*, 2014, pp. 1646–1654.

[19] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[20] N. Kargas and N. D. Sidiropoulos, "Learning mixtures of smooth product distributions: Identifiability and algorithm," in *Proc. AISTATS*, 2019 [to appear].

[21] Y. Zhang, X. Chen, D. Zhou, and M. I. Jordan, "Spectral methods meet em: A provably optimal algorithm for crowdsourcing," *Journal of Machine Learning Research*, vol. 17, no. 102, pp. 1–44, 2016.