Fast and Accurate Tensor Decomposition without a High Performance Computing Machine

Huan He
Department of Computer Science
Emory University
Atlanta, Georgia
Email: huan.he@emory.edu

Yuanzhe Xi
Department of Mathematics
Emory University
Atlanta, Georgia
Email: yuanzhe.xi@emory.edu

Joyce C Ho
Department of Computer Science
Emory University
Atlanta, Georgia
Email: joyce.c.ho@emory.edu

2020 IEEE International Conference on Big Data (Big Data) | 978-1-7281-6251-5/20/$31.00 ©2020 IEEE | DOI: 10.1109/BigData50022.2020.9378111

***Abstract*—The rapid growth in the collection of high-dimensional data has led to the emergence of tensor decomposition, a powerful analysis method for the exploration of such data. Since tensor decomposition can extract hidden structures and capture underlying relationships between variables, it has been used successfully across a broad range of applications. However, tensor decomposition is a computationally expensive task, and most existing methods developed to decompose large tensors require expensive computing hardware or high-performance computing environment. Moreover, existing approaches focus solely on numeric data, and may not yield desirable results for binary or count data. Therefore, we propose `FAST-CP`, a novel algorithm to accelerate the convergence of the stochastic gradient descent based CANDECOMP/PARAFAC (CP) decomposition model through a new extrapolation method. Our algorithm can model a variety of tensor data types, accelerates convergence in terms of speed and quality, and improves the learning stability of stochastic gradient descent. Our empirical results on three real-world datasets demonstrate that `FAST-CP` decreases the total computation time while providing accurate results without necessitating a high-performance computing platform or environment.**

***Index Terms*—Tensor Decomposition, gradient descent, stochastic gradient descent, extrapolation, acceleration**

I. Introduction

Tensor, or N-way array, is a powerful representation that encodes the relationship between N dimensions. These multi-way arrays can then be factored to identify the underlying patterns in the data. The CANDECOMP/PARAFAC (CP) decomposition [1] is a popular model due to its intuitive output structure and uniqueness property. Under CP decomposition, a dataset with three modes that is stored as an $I \times J \times K$ tensor $\mathcal{X}$ is factorized as a sum of multi-way outer (rank-one) products, $\mathcal{X} = \sum_{r=1}^{R} \mathbf{A}_r^{(1)} \circ \mathbf{A}_r^{(2)} \circ \mathbf{A}_r^{(3)}$, where $\mathbf{A}_r^{(1)}, \mathbf{A}_r^{(2)}, \mathbf{A}_r^{(3)}$ are column vectors of size I, J, K, respectively, that represent latent data concepts. Due to its simplicity and reliability, various domains have used CP decomposition to identify patterns including urban computing, network analysis, computer vision, healthcare, and criminology [2], [3].

The most popular approach for solving the sum of squared error formulation is the alternating least squares (ALS) method. Conceptually and numerically simple, it provides astonishingly good results in many cases if employed with care [4]. However, the ALS method requires a significant amount of memory. To address the memory limitations, scalable ALS-based algorithms have been proposed yet often require a high-performance computing environment (e.g., Spark, MapReduce). Such computational requirements make CP decomposition difficult for practitioners to readily adopt. For example, in the healthcare setting, there is limited access to such high-performance systems and data cannot be transferred due to patient privacy concerns.

Another limitation of CP-ALS is the inability to model non-numeric data (e.g., count or binary). Several studies show that non-standard choices of the scalar loss function yield more reasonable results and are more appropriate for different types of data [5]–[8]. For these loss functions, various optimization-based algorithms can be adapted including Conjugate Gradient, Gauss-Newton, Levenberg-Marquardt and limited-memory BFGS algorithms (see [4], [9] for a comparison of these methods). Unfortunately, as shown in [9], each gradient step is essentially as expensive as an ALS step.

Recently, several stochastic gradient descent (SGD) based algorithms have been proposed [10]–[13]. These methods randomly select samples from the tensor at each iteration and optimize the factor matrices based on these entries, thereby circumventing the formation of the dense tensor needed by the full gradient. This is particularly beneficial for performing large-scale tensor decomposition with limited computing resources. Moreover, a gradient-based approach allows for any arbitrary element-wise loss function that is summed across all tensor entries. This advantage over the ALS method is critical in many applications of CP decomposition as the data can be described by different distributions (e.g., Binary, Gaussian, Poisson). Although SGD-based algorithms have been proved to be a promising approach, they often require a significantly large number of iterations to converge to an optimal point. As a result, these algorithms often exhibit very poor convergence properties. Compared to ALS, SGD performs updates with less computational cost, but is less accurate. In addition to reducing the convergence speed, noise in the gradient makes SGD-based algorithms harder to tune. Indeed, it has been shown that the stochastic gradient method with a constant stepsize only converges to a ball around the optimum.

To address the limitations of existing SGD-based algorithms mentioned above, we propose `FAST-CP`, an accelerated SGD-based CP decomposition model for large-scale tensors on a personal computer. Our model accelerates the convergence speed by mixing past iterates in a systematic fashion and decreases variance of the stochastic gradients. FAST-CP is up to 4 times faster than the state-of-the-art tensor SGD algorithm [10] across three different datasets, and can execute in a reasonable time for large datasets. Our contribution can be summarized as follows:

- Efficient extrapolation step: We propose a computationally cheap technique that we call extrapolation to speed up the convergence of stochastic gradient updates for large-scale tensor CP decomposition with various loss functions.
- Improved convergence: While SGD is memory efficient, it usually performs poorly in terms of convergence rate and quality. We illustrate how extrapolation of gradient sequences can fix this issue and yield better factor matrices.
- Robustness: We show that FAST-CP improves the learning stability and lowers the variance of its base optimizer with negligible computation and memory cost.
- Generalizability: We empirically demonstrate FAST-CP can significantly improve the performance of various types of tensors and different variants of SGD algorithms including the standard (vanilla) version and Adam.

II. Background

This section briefly introduces CP tensor decomposition and related work. The list of operations and symbols used in this paper are listed in Table I.

A. Tensor and Tensor Operations

TABLE I: Symbols and their associated definitions

Symbol	Definition
$\mathcal{X}$, $\mathbf{X}$, $\mathbf{x}$, x	Tensor, Matrix, Column Vector, Scalar
$\mathbb{1}$	All one matrix
$\mathbf{X}_{(n)}$	n-mode matricization of a tensor $\mathcal{X}$
$\mathbf{X}(\mathbf{r},:)$	rth row of $\mathbf{X}$
$\mathbf{X}(:,\mathbf{r})$	rth column of $\mathbf{X}$
$\mathbf{A}^{(n)}$	nth factor matrix
$\mathbf{x}_n$	nth element of vector $\mathbf{x}$
$\|\cdot\|_2, \|\cdot\|_F$	Matrix 2 norm, Frobenius norm
$*$	Hadamard (elementwise) product
$\oslash$	Hadamard (elementwise) division
$\circ$	outer product
$\otimes$	Kronecker product
$\odot$	Khatri-Rao product (column-wise $\otimes$)

Tensors are generalizations of matrices and vectors to higher dimensions. An N-way tensor is denoted as $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and each element of the tensor represents the interactions between N types of data. Each dimension of the tensor is referred to as a *mode*. Tensors can be unfolded or flattened as a matrix, which is called *matricization*. $\mathbf{X}_{(n)}$ denotes the matricization of $\mathcal{X}$ along mode-n.

Definition 1. *A rank-one N-way tensor is the outer product of N vectors: $\mathcal{X} = a^{(1)} \circ a^{(2)} \circ \cdots \circ a^{(N)}$. Each element of a rank-one tensor is the product of the corresponding vector elements (i.e., $x_{i_1 i_2 \cdots i_N} = a^{(1)}_{i_1} a^{(2)}_{i_2} \cdots a^{(N)}_{i_N}$).*

Definition 2. *The Khatri-Rao product of two real-valued matrices $\mathbf{A} \odot \mathbf{B}$ of sizes $I_{\mathbf{A}} \times R$ and $I_{\mathbf{B}} \times R$, respectively, produces a matrix $\mathbf{Z}$ of size $I_{\mathbf{A}} I_{\mathbf{B}} \times R$ such that $\mathbf{Z} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \cdots & \mathbf{a}_R \otimes \mathbf{b}_R \end{bmatrix}$, where $\otimes$ is the Kronecker product.*

In this paper, the matrix $\mathbf{Z}_n$ represents the Khatri-Rao product of all the factor matrices except $\mathbf{A}^{(n)}$ such that $\mathbf{Z}_n = \mathbf{A}^{(1)} \odot \cdots \odot \mathbf{A}^{(n-1)} \odot \mathbf{A}^{(n+1)} \odot \cdots \odot \mathbf{A}^{(N)}$.

B. CP decomposition

The CANDECOMP / PARAFAC (CP) model [1] is one of the most popular and well-studied tensor decomposition methods. In CP decomposition, the observed tensor, $\mathcal{X}$, is approximated using a sum of rank-one tensors (or $\mathcal{M}$):

$$\mathcal{X} \approx \mathcal{M} = \sum_{r=1}^{R} \mathbf{A}^{(1)}(:,r) \circ \mathbf{A}^{(2)}(:,r) \circ \cdots \circ \mathbf{A}^{(N)}(:,r). \quad (1)$$

Figure 1 provides an example of the CP decomposition for a sparse tensor, where each rank-one tensor represents a latent factor. Fitting a CP decomposition involves minimizing an objective function between the tensor $\mathcal{X}$ and a model tensor $\mathcal{M}$. In general, it takes the form of summation of element-wise loss functions over all entries, and is chosen based on assumptions about the underlying distribution of the data.

$$\text{minimize}\, F(\mathcal{X}, \mathcal{M}) \equiv \sum_{i_1 i_2 \cdots i_N} f\left(x_{i_1 \dots i_N}, m_{i_1 \dots i_N}\right) \quad (2)$$

For the numeric data, it is common to assume that the tensor elements follow a Gaussian distribution, which corresponds to the least squares approximation. The objective function f associated with the least squares approximation is defined as:

$$F(\mathcal{X}, \mathcal{M}) = \sum_{i_1 \cdots i_N} (x_{i_1 \cdots i_N} - m_{i_1 \cdots i_N})^2. \quad (3)$$

For count data, an appropriate assumption about the underlying distribution of the data is Poisson [6], [7] and the following KL-divergence fitting function f is used:

$$F(\mathcal{X}, \mathcal{M}) = \sum_{i_1 \cdots i_N} (m_{i_1 \cdots i_N} - x_{i_1 \cdots i_N} \log m_{i_1 \cdots i_N}). \quad (4)$$

The CP decomposition with the loss function (4) is often referred to as *CP-APR* [6], [7]. Using a Poisson model leads to a much better explanation for the zero observations encountered in sparse data, where these zeros correspond to events that are unlikely to be observed.

For binary data, it is natural to assume the tensor elements follow a Bernoulli distribution and the objective function f takes the following form:

$$F(\mathcal{X}, \mathcal{M}) = \sum_{i_1 \cdots i_N} (\log(1 + m_{i_1 \cdots i_N}) - x_{i_1 \cdots i_N} \log m_{i_1 \cdots i_N}). \quad (5)$$

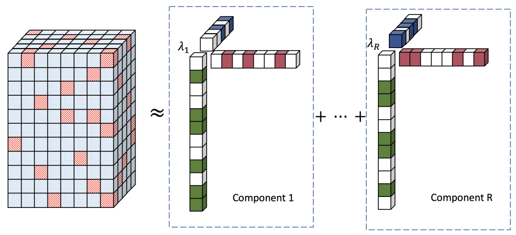

Fig. 1: An illustration of CP decomposition for a sparse tensor. Shaded squares stand for nonzeros. The original tensor is approximated by the weighted sum of R rank-one tensors.

Both (4) and (5) require a non-negative constraint.

C. Nonlinear Acceleration Techniques

When a sequence of numbers, vectors, matrices or tensors converge slowly, or even diverge, extrapolation techniques can be used to transform the current sequence into a new sequence, which, under certain assumptions, converges faster. There exist many such sequence transformations which range across a wide range of disciplines with varying goals and various degree of success. For a review, see [14].

Classical acceleration techniques take a sequence $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n$, e.g., vectors in R^d, and produce an accelerated sequence $\{\mathbf{t}_n^{(k)}\}$ of the form

$$\mathbf{t}_n^{(k)} = a_0\mathbf{x}_n + a_1\mathbf{x}_{n+1} + \cdots + a_k\mathbf{x}_{n+k}, \tag{6}$$

where the a_i's usually depend on k and n but satisfy the constraint $\sum_{i=0}^{k} a_i = 1$.

Most existing extrapolation techniques are based on the assumption that $\mathbf{x}_n$ satisfies a k-term kernel of the form:

$$a_0(\mathbf{x}_n - \mathbf{x}) + a_1(\mathbf{x}_{n+1} - \mathbf{x}) + \cdots + a_k(\mathbf{x}_{n+k} - \mathbf{x}) = \mathbf{0}, \forall n, \tag{7}$$

where $\mathbf{x}$ is the exact limit for the original sequence and the scalars $a_0, \ldots, a_k$ and $\mathbf{x}$ are unknowns with $a_0 a_k \neq 0$ and satisfy the constraint. (7) is called *Shanks kernel* in the literature.

SGD methods with this acceleration techniques have been shown effective for deep learning training [15], [16]. These methods compute an extrapolated set of parameters in neural networks by mixing the information from the past few iterates and can dramatically reduce the number of epochs. However, simple adoption of such techniques fails to speed up tensor CP decomposition.

III. Method

Although there has been a significant recent interest on developing scalable tensor factorization methods, most of these methods treat data as real-valued, and are therefore inappropriate for handling binary and count data. Motivated by the prevalence of different types of tensors, we present `FAST-CP`, a scalable and accelerated tensor decomposition framework which can handle numeric, binary and count-valued tensors. We begin by explaining how SGD works for variety of CP objective functions. Next we present our novel extrapolation technique that improves the convergence of SGD methods and obtains a more optimal convergence point. Then we provide some theoretical convergence analyses to support our method.

A. Generalized CP decomposition using SGD

Gradient descent is a common technique used to deal with CP decomposition associated with various loss functions f. If we define $\mathcal{Y} = \mathcal{X} - \mathcal{M}$, the partial derivative of F in Equation (3) with respect to $\mathbf{A}^{(n)}$ can be written as

$$\frac{\partial F}{\partial \mathbf{A}^{(n)}} = -2\mathbf{Y}_{(n)}\mathbf{Z}_n, \tag{8}$$

where $\mathbf{Y}_{(n)}$ is the matricization of $\mathcal{Y}$ along mode-n and $\mathbf{Z}_n$ is the Khatri-Rao product of all factor matrices except $\mathbf{A}^{(n)}$. The operation on the right hand size of (8) is called the *matricized tensor times Khatri-Rao product* (MTTKRP).

Similarly, the partial derivative of F in Equation (4) with respect to $\mathbf{A}^{(n)}$ can be written as

$$\frac{\partial F}{\partial \mathbf{A}^{(n)}} = -(\mathbb{1} - \mathbf{X}_n \oslash \mathbf{Y}_{(n)})\mathbf{Z}_n, \tag{9}$$

and the partial derivative of F in Equation (5) with respect to $\mathbf{A}^{(n)}$ can be written as

$$\frac{\partial F}{\partial \mathbf{A}^{(n)}} = -(\mathbb{1} \oslash \mathbf{Y}_{(n)} - \mathbf{X}_n \oslash \mathbf{Y}_{(n)})\mathbf{Z}_n, \tag{10}$$

For Equations (8)–(10), even when $\mathcal{X}$ is sparse, $\mathcal{Y}$ is usually a fully dense tensor. If $S = \prod_{i=1}^{N} I_i$, then the calculation of gradients for CP decomposition involves an intermediate sequence of N matricized-tensor times Khatri-Rao products (MTTKRPs) with a dense tensor of size S. These operations cost $O(RS)$, even when $\mathcal{X}$ is sparse. Thus, for large-scale tensor problems, the computational and storage costs of computing the exact gradient may be infeasible.

In recent years, SGD algorithms have been proposed to alleviate the difficulty of applying standard gradient descent in tensor decomposition problems. To create a random sparse instance $\tilde{\mathbf{Y}}_{(n)}$ of $\mathbf{Y}_{(n)}$, one can sample K indices uniformly with replacement. This uniform sampling is one of the most common strategies used for fitting dense tensors. However it may not be appropriate for sparse tensors since nonzeros will rarely be sampled. A stratified sampling-based SGD algorithm has recently been proposed in [10] to fix this issue. Different from previously proposed tensor SGD algorithms [17], [18], this algorithm samples both zero and non-zero elements of $\mathcal{X}$ and constructs $\tilde{\mathcal{Y}}$ as an unbiased estimation of $\mathcal{Y}$.

While SGD-based algorithms are memory efficient, both uniform sampling and stratified sampling-based approaches often require too many iterations to converge in practice. Moreover, the SGD-based CP algorithm can oscillate around the minimal. Thus, we are interested in designing an algorithm that can further accelerate SGD algorithms and yield more accurate results.

B. Extrapolated Stochastic Gradient Descent

Classical extrapolation methods reviewed in Section II-C can be quite effective but they all rely heavily on some intrinsic *smoothness* characteristics of the sequence. This smoothness is expressed by either the Shanks kernel or the differential of the function f. Preliminary experiments show that extrapolation on factor matrices did not yield any considerable speedup. This is because the factor matrices generated by SGD algorithms are both random and noisy, which violates the smoothness assumption required by classical extrapolation methods.

In the deep learning community, a similar idea called *smoothing* has been explored to avoid sharp minima and obtain better generalization performance [19], [20]. The basic idea is to uniformly average a sample of past gradients to obtain the so-called *extragradient*. In fact, these smoothing techniques correspond to a special case in tensor SGD algorithms when only the last gradient matrix is used to update the current factor matrix. Since the convergence of these smoothing methods depends on the magnitude of the extragradient and classical nonlinear acceleration techniques can produce a gradient with smaller norm than the extragradient, applying extrapolation techniques on gradient sequences can lead to faster convergence. In this paper, we adopt the Vector Epsilon Algorithm (VEA) [21] framework to implement our gradient sequence extrapolation. This framework uses the generalized matrix inverse to extend the scalar ϵ-algorithm to sequences of matrices. The algorithm is summarized in the following formula:

$$\begin{aligned} \epsilon_{-1}^{(i)} &= 0 \\ \epsilon_{0}^{(i)} &= \Delta\mathbf{A}_i^{(n)} \\ \epsilon_{k+1}^{(i)} &= \epsilon_{k-1}^{(i+1)} + \frac{(\epsilon_k^{(i+1)} - \epsilon_k^{(i)})}{\left\|\epsilon_k^{(i+1)} - \epsilon_k^{(i)}\right\|_F^2} \quad \text{for } k > 0. \end{aligned} \tag{11}$$

The final output of those sequences defined in ϵ-algorithm represent the Shanks transforms of the original sequence $\Delta\mathbf{A}_i^{(n)}$. As soon as a new iterate $\Delta\mathbf{A}_i^{(n)}$ becomes available we can immediately compute $\epsilon_1^{(i-1)}, \epsilon_2^{(i-2)}, \ldots$. The VEA implementation is detailed in Algorithm 1.

After we obtain the extrapolated gradient matrix, we use it to update the current factor matrix with the same stepsize as the baseline optimization algorithm. Note that this is different from VEA where the original sequence is not interlaced with the extrapolated one. We call this extrapolated SGD method as `FAST-CP` and detail its major operations in Algorithm 2. We further extend the extrapolation idea of `FAST-CP-SGD`to `FAST-CP-Adam`and validate its effectiveness (compared with Adam) on CP decomposition in the experiments. When Adam is used as the baseline optimizer, we still pass the stochastic gradient sequence in Algorithm 1.

C. Theoretical Analysis

In this section, we provide some theoretical justifications for the proposed method shown in Algorithm 1.

Following most convergence analyses for SGD methods [22], [23], we make the following three assumptions for the tensor gradient sequence:

Algorithm 1 Extrapolation of gradient sequence for mode n

```
1: For the first time: Initialize a table $\mathbf{T}^{(n)}$ with $2k+1$
   columns. Set a window size $k$.
2: **Input**: The latest gradient matrix $\Delta\mathbf{A}_i^{(n)}$ and factor matrix
   $\mathbf{A}_{i+1}^{(n)}$, current iteration number $i$, table $\mathbf{T}^{(n)}$
3: **Output:** Updated factor matrix $\mathbf{A}_{i+1}^{(n)}$, table $\mathbf{T}^{(n)}$
4: $j = 1, \mathbf{z} = 0$ and an empty array $\mathbf{Y}$
5: Set $\mathbf{Y}(:,1) = \Delta\mathbf{A}_i^{(n)}$
6: **while** $j < 2k+1$ and $j < i$ **do**
7:    Compute $\triangle\varepsilon = \mathbf{Y}(:,j) - \mathbf{T}^{(n)}(:,j)$
8:    Calculate $\mathbf{z} = \mathbf{z} + \triangle\varepsilon / \|\triangle\varepsilon\|_F^2$
9:    $\mathbf{Y}(:,j+1) = \mathbf{z}$
10:   Reset $\mathbf{z} = \mathbf{T}^{(n)}(:,j)$
11:   $j = j+1$
12: **end while**
13: Set $\mathbf{T}^{(n)} = \mathbf{Y}$ and $\mathbf{E}^{(n)} = \mathbf{T}^{(n)}(:,2k+1)$
14: **if** $i \le 2k$ and $j$ mod 2 == 1 **then** # when $i$ is too small
15:    $\mathbf{E}^{(n)} = \mathbf{E}^{(n)} / \left\|\mathbf{E}^{(n)}\right\|_F^2$
16: **end if**
17: Update $\mathbf{A}_{i+1}^{(n)} := \mathbf{A}_{i+1}^{(n)} - \gamma\mathbf{E}^{(n)}$
18: **return** $\mathbf{A}_{i+1}^{(n)}$, $\mathbf{T}^{(n)}$
```

Algorithm 2 `FAST-CP`

```
1: **Input**: $N$-way tensor $\mathcal{X}$, batch size $K$, learning rate $\gamma$,
   max epoch number $M$, number of SGD updates per epoch
   $I$, gradient sequence length $k$
2: **Output**: CP decomposition $\mathbf{A}^{(1)}, \cdots, \mathbf{A}^{(N)}$
3: Initialize factors $\mathbf{A}^{(1)}, \cdots, \mathbf{A}^{(N)}$
4: **while** Not converged or max epoch number not reached
   **do**
5:    Shuffle indices of tensor $\mathcal{X}$
6:    **for** $i = 1 : I$ **do**
7:       Get samples either using uniform sampling or strati-
         fied sampling
8:       **for all** modes **do**          # optimize all at once
9:          Calculate gradients $\Delta\mathbf{A}^{(n)}$ according to (8)–(10)
10:         Update $\mathbf{A}_{i+1}^{(n)} = \mathbf{A}_i^{(n)} - \gamma\Delta\mathbf{A}_i^{(n)}$
11:         Update $\mathbf{A}_{i+1}^{(n)}$ and $\mathbf{T}^{(n)}$ according to Algorithm 1
12:      **end for**
13:   **end for**
14:   Check convergence
15: **end while**
16: **return** $\mathbf{A}^{(1)}, \cdots, \mathbf{A}^{(N)}$
```

Assumption 1 The objective function F is continuously differentiable and the gradient of F is Lipschitz continuous with Lipschitz constant $L > 0$.

Assumption 2 The stochastic gradient computed by Equations (8)–(10) is an unbiased estimator of the true gradient.

Assumption 3 The variance of the stochastic gradients is bounded. That is there exists a constant $\sigma^2 > 0$ such that

$$\mathbb{E}\left[\|\frac{\partial F}{\partial \mathbf{A}^{(n)}} - \Delta\mathbf{A}^{(n)}\|^2\right] \le \sigma^2$$

for all modes n.

We first review the known convergence result for mini-batch SGD on non-convex functions from [24].

Theorem 1. *Under the assumptions 1-3, after T mini-batch gradient updates, each with K samples, the mini-batch SGD returns an iterate $\mathbf{x}$ which satisfies*

$$\mathbb{E}\left[\|\nabla F(\mathbf{x})\|^2\right] \leq \mathcal{O}\left(\frac{L(F(\mathbf{x}_0)-F^*)}{T} + \frac{\sigma\sqrt{L(F(\mathbf{x}_0)-F^*)}}{\sqrt{KT}}\right),$$

where $\mathbf{x}_0$ is the initial guess and F^* is a lower bound on the values of F.

Theorem 1 indicates that when the mini-batch size is small, the converge rate is dominated by the term $\frac{\sigma\sqrt{L(F(\mathbf{x}_0)-F^*)}}{\sqrt{KT}}$ which has possible speedup as K increases. On the other hand, when the mini-batch size becomes large, further increasing the mini-batch size has little effect on the convergence as the convergence rate will be dominated by the first term $\frac{L(F(\mathbf{x}_0)-F^*)}{T}$. In the numerical experiments, we carefully tune this hyperparameter and choose the optimal one for baseline methods.

In the next theorem, we review the asymptotic convergence for SGD with extragradients when only one worker and no momentum term are used.

Theorem 2 (Theorem 4.4 [23])**.** *Under the same assumptions as in Theorem 1, when the stepsize $\gamma \leq \frac{1}{L}$, the sequence $\mathbf{x}_t$ generated by extragradients satisfies*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1}\|\nabla F(\mathbf{x}_t)\|^2\right] \tag{12}$$

$$\leq \frac{2}{\gamma T}\mathbb{E}\left[F(\mathbf{x}_0)-F^*\right] + \left(\frac{4\gamma^2L^2}{K} + \frac{\gamma L}{K}\right)\sigma^2, \tag{13}$$

where $\mathbf{x}_0$ is the initial guess and F^* is a lower bound on the values of F.

Although Theorem 2 cannot show that SGD with extragradients achieves a speedup over mini-batch SGD, its superior performance in terms of faster convergence and better generalization has been demonstrated in various deep learning tasks [23], [25], [26].

Figure 2 shows that the proposed method can significantly accelerate the convergence. It shows that SGD with extragradient does not yield any improvement on the tensor CP decomposition while extrapolating on a gradient sequence with $k = 3$ can significantly accelerate the convergence.

IV. Experiments and Results

All experiments were run using Python on a Dual Socket Intel E5-2683v3 2.00GHz CPU with 64 GB memory. [1]

[1]Our implementation is available at https://github.com/hehuannb/fast-cp.

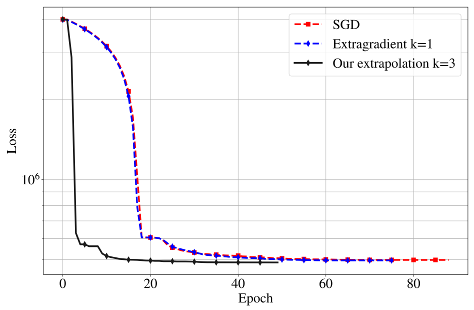

Fig. 2: Vector Epsilon Algorithm (VEA) is applied on the stochastic gradient sequence for HCP dataset. Extragradient $k = 1$ refers to the smoothing technique [23].

A. Datasets

We use the following three publicly available tensors that are from small to large and consist of continuous, count and binary data respectively.

- Human Connectome Project (HCP): Human Connectome Project collects measurements of structural and functional neural connections in vivo within and across individuals[2]. We use the constructed tensor from [27]. It is a $68 \times 68 \times 212$ binary tensor consisting of structural connectivity patterns among brain regions for 212 individuals. Each entry encodes the presence or absence of fiber connections between the brain regions.
- MNIST: A dataset of handwritten numbers from 250 different people, available from the National Institute of Standards and Technology (NIST) [28]. The MNIST dataset contains 60,000 images in the training set, each of size 28×28 pixels with 256 gray levels. We normalize each element by using the global mean (0.1307) and standard deviation (0.3081). After the normalization process, a $60000 \times 28 \times 28$ numeric tensor is obtained.
- Yelp: A dataset that contains 4M ratings from 1M users in Yelp across 149 months[3]. The tensor modes correspond to 1,029,432 Yelp users, 144,072 businesses and 149 months. Each entry represents the user rating (integer from 1 to 5) for the business. We use this dataset to demonstrate the scalability of `FAST-CP`.

B. Baseline Methods

In this experiment, four baselines have been selected to evaluate the performance. The baseline methods contain alternating minimization including CP-ALS [29] and CP-APR [6], [7] and SGD-based methods including GCP-SGD and Adam [10].

- CP-ALS [29]: The standard method for fitting the CP model to numeric data. The algorithm alternates among the modes, fixing every factor matrix but $\mathbf{A}^{(i)}$. CP-ALS has a closed-form solution for each mode but requires significant memory.

[2]http://www.humanconnectomeproject.org/data/hcp-project/
[3]http://www.yelp.com/datasetchallenge

- CP-APR [6], [7]: An algorithm proposed for modeling sparse count data using a Poisson distribution. This algorithm employs an alternating optimization scheme that sequentially optimizes one factor matrix while holding the others fixed. We use the state-of-the-art CP-APR [7] as a baseline model which uses limited-memory quasi-Newton approximations.
- GCP-SGD [10]: An algorithm for fitting the generalized CP decomposition using SGD. It adopts both uniform sampling and stratified sampling strategies. When the tensor is dense, we use the uniform sampling. When the tensor is sparse, we choose stratified sampling since it is more efficient and converges faster. It uses SGD as the base optimizer.
- GCP-Adam [10]: The only difference is that GCP-Adam uses Adam [30] as the base optimizer, which usually converges faster than SGD in deep learning applications.

C. Experimental Specifications

1) Evaluation metrics: We use the evaluation of loss functions as the measure for tensor reconstruction error, which are defined in Equations (3), (4), and (5). We also report CPU time (sec) to show the cost-benefit trade-off of extrapolation step.

2) Experimental setup: The loss function is selected to reflect the appropriate distribution for each dataset. In other words, the Gaussian (Equation (1)), Poisson (Equation (4)), and Bernoulli (Equation (5)) distributions are used for MNIST, Yelp, and HCP datasets, respectively. We fix the rank, R, for each dataset and set it to reflect the size of the tensor. The rank is set to 60, 100, and 10, for MNIST, Yelp, and HCP datasets respectively.

For CP-ALS and CP-APR, we set the max iteration number as 100. For CP-ALS, the iteration stops when either the max iteration number is reached or the change is under $1e-4$. For CP-APR, the iteration stops when either the max iteration number is reached or it violates the KKT condition [7]. For SGD-based methods, whenever the loss function value fails to decrease, we decay the learning rate by 0.1. If the loss function value remains the same after three consecutive learning rate decreases, the algorithm stops. Finally, uniform sampling is used for MNIST since it is dense, while stratified sampling is used for Yelp and HCP, which are both sparse.

D. Acceleration of CP-SGD

In this section, we evaluate the effectiveness of our extrapolation framework and compare it with baseline methods on the three datasets. All the methods use the same initialization, and the learning rate γ is carefully tuned to yield the best results in terms of the lowest loss value. Figure 3 summarizes the convergence rate for the two SGD-based methods (vanilla SGD and Adam) with and without our extrapolation method. As can be seen, `FAST-CP` significantly accelerates the convergence in terms of number of epochs for both SGD variants.

To better understand the computation time of the different methods, Figure 4 plots the loss as a function of time. This figure demonstrates that the extrapolation step is computationally efficient, as it adds little time in terms of computation. `FAST-CP` (with vanilla SGD) converges faster than the standard CP decomposition methods, CP-ALS and CP-APR, for MNIST and Yelp, respectively. Moreover, the accuracy of the solution from `FAST-CP` is comparable to that obtained from the traditional methods.

We can observe that Adam can sometimes converge faster than SGD in terms of number of epochs, however, the convergence point is often less desirable than vanilla SGD. In addition, the updates of the first moment and second moment are not computationally efficient, and thus Adam requires much more time. On the other hand, `FAST-CP-Adam` can avoid some of the bad local minima as it smoothens the loss surface (when compared to Adam without extrapolation). However, `FAST-CP-SGD` still yields a comparable or even better local minima.

To investigate the stability of `FAST-CP`, we run 10 experiments for each dataset and method and summarize results in Table II. We make two observations from the table. First, `FAST-CP` significantly outperforms its base optimizer, both in terms of convergence and time, demonstrating the effectiveness of the extrapolation. Second, we can see that although `FAST-CP-Adam`is less accurate than `FAST-CP-SGD`, the convergence point is still much better for `FAST-CP-Adam`than for Adam. It indicates that the extrapolation step helps smoothen the loss surface of Adam. Third, `FAST-CP` under different learning rates converge to similar points for each dataset. This improvement is important for better anytime performance in new datasets where the learning rate is not well-calibrated.

E. Effects of hyperparameters

To better understand the optimization behaviors of `FAST-CP` under different learning rates and hyperparameter settings, we perform additional experiments varying these values in this section.

1) Convergence using different learning rates: In the previous section, we show that `FAST-CP` can accelerate both SGD and Adam under a fine-tuned learning rate. We also explored the performance of `FAST-CP` under different learning rates. We only present results of HCP here due to the space limitation. Figure 5 presents different learning rates for the vanilla SGD and Adam with and without extrapolation on the HCP tensor. From the figure, we notice there is little discrepancy in the convergence for `FAST-CP`. The impact of learning rate is much more noticeable in the SGD-based algorithms without extrapolation. Thus, our method improves the stability of the decomposition and lowers the variance of its base optimizer without incurring significant computation and memory cost. Experiments on MNIST and Yelp exhibit similar performance.

2) Understanding the effect of extrapolation sequence length k: One key hyperparameter used in `FAST-CP` is k, the length of the extrapolation sequence. We investigate its effect on `FAST-CP` in this section. Previously, we observed

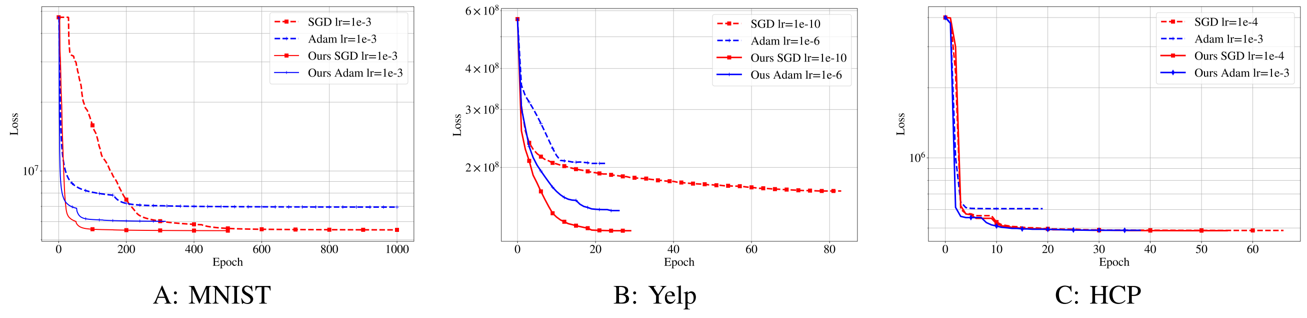

Fig. 3: Convergence plots for the SGD-based methods for the 3 datasets. In A, $R = 60$, batch size=2000, each epoch contains 3000 iterations. In B, $R = 100$, batch size=5000, each epoch contains 100 iterations. In C, $R = 10$, batch size=500, each epoch contains 300 iterations.

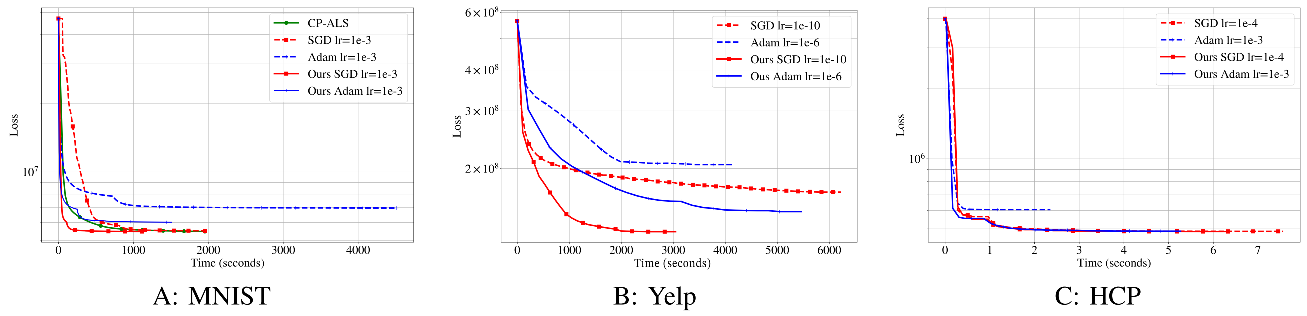

Fig. 4: Time comparison among baseline models on three datasets. It shows that extrapolation can converge to a more optimal point with cost of limited extra time while Adam takes much more time.

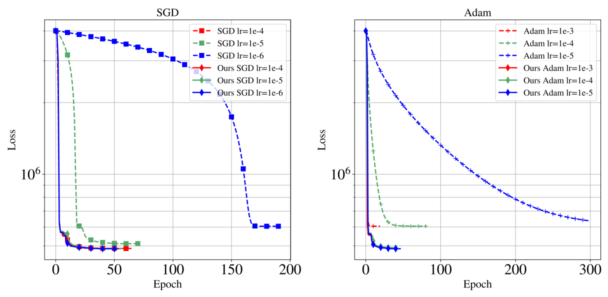

Fig. 5: Convergence under different learning rate on HCP

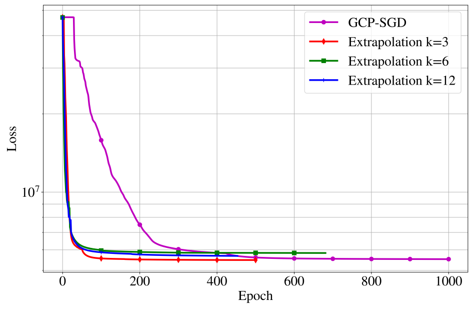

Fig. 6: MNIST: Effect of the hyperparameter k on convergence of extrapolated SGD.

that extragradient ($k = 1$) does not offer any improvement and using extrapolation on a length-3 gradient sequence can. Does a larger k yield further improvement? The answer is no. In Figure 6, we can see that a length of 3 already provides the majority of the benefits. Not only does a larger k incur more memory, but it even converges at a worse point. This is reasonable because extrapolation using a larger k leads to numerically ill-conditioned problems and generates an unstable extrapolation. Moreover, a larger k also requires additional computation. Therefore, we suggest $k = 3$ as a robust hyperparameter for `FAST-CP`.

V. Conclusion

In this paper, we present `FAST-CP`, a fast and accurate SGD tensor decomposition framework that scales to large tensors. We proposed a simple extrapolation technique which, by reusing past gradients and transporting them, offers excellent performance on a variety of tensor decomposition problems. By providing a higher quality gradient estimate that can be plugged in the existing optimizer, we demonstrate that `FAST-CP` can significantly improve the performance of SGD and Adam, even with different learning rate.

VI. Acknowledgement

This work was supported by the National Science Foundation awards IIS-#1838200 and OAC-#2003720.

TABLE II: The performance comparison of different methods on three datasets averaged over 10 different seeds

(a) MNIST

Method.	Lr	Loss±std (1e6)	Time±std (sec.)
CP-ALS	-	5.4839 ±0.0512	2016 ±41.83
GCP-SGD	1e-2	5.6819 ±0.0728	1924 ±60.84
GCP-SGD	1e-3	5.5243 ±0.0772	1956 ±47.98
GCP-Adam	1e-3	6.849 ±0.0693	4515 ±115.47
GCP-Adam	1e-4	7.1932 ±0.0713	4492 ±105.38
`FAST-CP` SGD	1e-2	**5.5025 ±0.042**	**796 ±56.59**
`FAST-CP` SGD	1e-3	**5.4815 ±0.061**	**993 ±48.96**
`FAST-CP` Adam	1e-3	**5.6226 ±0.082**	**1511 ±98.45**
`FAST-CP` Adam	1e-4	**5.595 ±0.079**	1813 ±104.26

(b) Yelp

Method.	Lr	Loss±std (1e8)	Time±std (sec.)
CP-APR	-	1.324 ±0.0103	9476 ±228.86
GCP-SGD	1e-10	1.657 ±0.0828	6325 ±74.52
GCP-SGD	1e-11	1.683 ±0.0659	7056 ±68.43
GCP-Adam	1e-6	1.931 ±0.2574	4159 ±180.36
GCP-Adam	5e-7	2.124 ±0.1789	4636 ±160.23
`FAST-CP` SGD	1e-10	**1.3183 ±0.0168**	**3045 ±40.08**
`FAST-CP` SGD	1e-11	**1.3346 ±0.0211**	**3213 ±45.59**
`FAST-CP` Adam	1e-6	**1.4432 ±0.0162**	5360 ±126.54
`FAST-CP` Adam	5e-7	**1.4648 ±0.0142**	4987 ±106.76

(c) HCP

Method.	Lr	Loss±std (1e5)	Time±std (sec.)
GCP-SGD	1e-4	4.8977 ±0.0228	8.05 ±3.38
GCP-SGD	1e-5	5.5514 ±0.4972	13.30 ±5.8
GCP-SGD	1e-6	5.9177 ±0.4384	21.05 ±4.21
GCP-Adam	1e-3	5.7180 ±0.4693	9.89 ±5.84
GCP-Adam	1e-4	5.8014 ±0.4313	9.75 ±5.3
GCP-Adam	1e-5	5.8414 ±0.4513	31.27 ±7.3
`FAST-CP` SGD	1e-4	**4.8724 ±0.0078**	**4.73 ±0.98**
`FAST-CP` SGD	1e-5	**4.8716 ±0.0081**	**4.68 ±0.96**
`FAST-CP` SGD	1e-6	**4.8719 ±0.0061**	**4.75 ±0.94**
`FAST-CP` Adam	1e-3	**4.8736 ±0.0082**	**4.98 ±1.01**
`FAST-CP` Adam	1e-4	**4.8728 ±0.0081**	**4.98 ±0.89**
`FAST-CP` Adam	1e-5	**4.8724 ±0.0079**	**4.96 ± 0.83**

REFERENCES

[1] R. A. Harshman, P. Ladefoged, H. G. von Reichenbach, R. I. Jennrich, D. Terbeek, L. Cooper, A. L. Comrey, P. M. Bentler, J. Yamane, and D. Vaughan, "Foundations of the parafac procedure: Models and conditions for an "explanatory" multimodal factor analysis," 1970.

[2] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "Tensors for Data Mining and Data Fusion - Models, Applications, and Scalable Algorithms." *ACM TIST*, 2017.

[3] J. C. Ho, J. Ghosh, and J. Sun, "Marble - high-throughput phenotyping from electronic health records via sparse nonnegative tensor factorization." *KDD*, 2014.

[4] G. Tomasi and R. Bro, "A comparison of algorithms for fitting the parafac model," *Computational Statistics Data Analysis*.

[5] P. Miettinen, "Boolean tensor factorizations," in *2011 IEEE 11th International Conference on Data Mining*, 2011, pp. 447–456.

[6] E. C. Chi and T. G. Kolda, "On tensors, sparsity, and nonnegative factorizations," *SIAM Journal on Matrix Analysis and Applications*, vol. 33, no. 4, pp. 1272–1299, 2012.

[7] S. Hansen, T. Plantenga, and T. G. Kolda, "Newton-based optimization for kullback–leibler nonnegative tensor factorizations," *Optimization Methods and Software*, vol. 30, no. 5, pp. 1002–1029, 2015.

[8] D. Hong, T. G. Kolda, and J. A. Duersch, "Generalized canonical polyadic tensor decomposition," *SIAM Review*, vol. 62, no. 1, pp. 133–163, 2020.

[9] L. Sorber, M. Van Barel, and L. De Lathauwer, "Optimization-based algorithms for tensor decompositions: Canonical polyadic decomposition, decomposition in rank-$(l_r,l_r,1)$ terms, and a new generalization," *SIAM Journal on Optimization*, 2013.

[10] T. G. Kolda and D. Hong, "Stochastic gradients for large-scale tensor decomposition," arXiv, June 2019, submitted for publication.

[11] D. Choi, J.-G. Jang, and U. Kang, "S3cmtf: Fast, accurate, and scalable method for incomplete coupled matrix-tensor factorization," *PLOS ONE*, 06 2019.

[12] H. He, J. Henderson, and J. C. Ho, "Distributed tensor decomposition for large scale health analytics," in *The World Wide Web Conference*, ser. WWW '19, 2019.

[13] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "ParCube - Sparse Parallelizable CANDECOMP-PARAFAC Tensor Decomposition." *TKDD*, vol. 10, no. 1, pp. 1–25, 2015.

[14] C. Brezinski, M. Redivo-Zaglia, and Y. Saad, "Shanks sequence transformations and anderson acceleration," *SIAM Review*, vol. 60, no. 3, pp. 646–669, 2018.

[15] D. Scieur, A. d'Aspremont, and F. Bach, "Regularized nonlinear acceleration," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. USA: Curran Associates Inc., 2016, pp. 712–720.

[16] D. Scieur, E. Oyallon, A. d'Aspremont, and F. Bach, "Nonlinear acceleration of deep neural networks," *arXiv preprint arXiv:1805.09639*, 2018.

[17] S. Smith, J. Park, and G. Karypis, "HPC formulations of optimization algorithms for tensor completion," *Parallel Computing*, vol. 74, no. Complex Syst. 3 4 1989, pp. 99–117, 2018.

[18] 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." *SDM*, pp. 109–117, 2014.

[19] W. Wen, Y. Wang, F. Yan, C. Xu, C. Wu, Y. Chen, and H. Li, "Smoothout: Smoothing out sharp minima to improve generalization in deep learning," *arXiv: Learning*, 2018.

[20] K. Haruki, T. Suzuki, Y. Hamakawa, T. Toda, R. Sakai, M. Ozawa, and M. Kimura, "Gradient noise convolution (gnc): Smoothing loss function for distributed large-batch sgd," *ArXiv*, vol. abs/1906.10822, 2019.

[21] P. Wynn, "Acceleration techniques for iterated vector and matrix," vol. 16, p. 301–322, 1962.

[22] F. Zhou and G. Cong, "On the convergence properties of a k-step averaging stochastic gradient descent algorithm for nonconvex optimization," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, J. Lang, Ed. ijcai.org, 2018, pp. 3219–3227.

[23] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Extrapolation for large-batch training in deep learning," *CoRR*, vol. abs/2006.05720, 2020. [Online]. Available: https://arxiv.org/abs/2006.05720

[24] S. Ghadimi and G. Lan, "Accelerated gradient methods for nonconvex nonlinear and stochastic programming," *Math. Program.*, vol. 156, no. 1-2, pp. 59–99, 2016. [Online]. Available: https://doi.org/10.1007/s10107-015-0871-8

[25] G. Gidel, H. Berard, P. Vincent, and S. Lacoste-Julien, "A variational inequality perspective on generative adversarial nets," *CoRR*, vol. abs/1802.10551, 2018. [Online]. Available: http://arxiv.org/abs/1802.10551

[26] C. Daskalakis, A. Ilyas, V. Syrgkanis, and H. Zeng, "Training gans with optimism," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[27] L. Wang, Z. Zhang, and D. Dunson, "Common and individual structure of brain networks," *The Annals of Applied Statistics*, vol. 13, pp. 85–112, 03 2019.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," 1998.

[29] R. Harshman, "Foundations of the parafac procedure: Models and conditions for an "explanatory" multi-model factor analysis," 1970.

[30] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.