Parallel and Distributed Inference in Coupled Tensor Factorization Models

Umut Şimşekli
Department of Computer Engineering
Boğaziçi University, İstanbul, Turkey
umut.simsekli@boun.edu.tr

Beyza Ermiş
Department of Computer Engineering
Boğaziçi University, İstanbul, Turkey
beyza.ermis@boun.edu.tr

Ali Taylan Cemgil
Department of Computer Engineering
Boğaziçi University, İstanbul, Turkey
taylan.cemgil@boun.edu.tr

Figen Öztoprak
Department of Industrial Engineering
Bilgi University, İstanbul, Turkey
figen.oztoprak@bilgi.edu.tr

Ş. İlker Birbil
Faculty of Engineering and Natural Sciences
Sabancı University, İstanbul, Turkey
sibirbil@sabanciuniv.edu

Abstract

Coupled tensor factorization methods are useful for sensor fusion, combining information from several related datasets by simultaneously approximating them by products of latent tensors. These methods are naturally adopted for the modern computing infrastructure that comprises of systems with hybrid architectures where the data might be very large and be distributed across several computers where each computer has multiple processors such as a multicore system or a graphics processing unit (GPU). In this paper, we present a parallel and distributed algorithmic framework for coupled tensor factorization to simultaneously estimate latent factors, specific divergences for each dataset as well as the relative weights in an overall additive cost function. We illustrate the proposed method on synthetic data experiments, where we show that the speed of the proposed algorithm is nearly proportional to the number of processing nodes.

1 Introduction

Coupled tensor factorization methods are useful in various application areas such as audio processing [1], computational psychology [2], bioinformatics [3] or collaborative filtering [4], where information from diverse sources are available and need to be combined for arriving at useful predictions. Examples of such situations are abound: for example for product recommendation, a product-buyer rating matrix can be enhanced with demographic information from the customer and connectivity information from a social network. In musical audio processing, one example is having a large collection of annotated audio data and symbolic music score information. The common theme in all such applications is the data fusion problem.

As a warm up, let us consider a simple example of a coupled matrix factorization model where two observed data matrices X_1 and X_2 are collectively decomposed as

$$\begin{aligned} X_1(i_1,i_3) &\approx \hat{X}_1(i_1,i_3) = \sum_{i_4} Z_1(i_1,i_4)Z_3(i_4,i_3), \\ X_2(i_2,i_3) &\approx \hat{X}_2(i_2,i_3) = \sum_{i_4} Z_2(i_2,i_4)Z_3(i_4,i_3) \end{aligned} \tag{1}$$

The factor Z_3 is the *shared factor* in both decompositions, making the overall model coupled. We call the factors Z_1 and Z_2 as *local factors* as they are only related to specific observations. The aim in this model is to estimate the latent factors Z_1, Z_2, and Z_3 given X_1 and X_2, where we need to solve the following optimization problem:

$$(Z_1^\star, Z_2^\star, Z_3^\star) = \underset{Z_1,Z_2,Z_3}{\arg\min} \left[\frac{1}{\phi_1} D_1(X_1||\hat{X}_1) + \frac{1}{\phi_2} D_2(X_2||\hat{X}_2)\right] \tag{2}$$

where D_1 and D_2 are *divergence functions* measuring the approximation error and the *dispersion parameters* ϕ_1 and ϕ_2 are the relative weights for the error in the approximation to each observed tensor. In practice, separable divergences are used, e.g., $D_1(X_1||\hat{X}_1) = \sum_{i_1,i_3} d_{p_1}(X_1(i_1,i_3)||\hat{X}_1(i_1,i_3))$. Some popular divergence (i.e., cost) functions are special cases of the β-divergence, defined as: ($p = 2 - \beta$)

$$d_p(x;\hat{x}) = \frac{x^{2-p}}{(1-p)(2-p)} - \frac{x\hat{x}^{1-p}}{1-p} + \frac{\hat{x}^{2-p}}{2-p} \tag{3}$$

where p is a power parameter. By taking appropriate limits it is easy to verify that d_p is the Euclidean distance square, information divergence or Itakura-Saito divergence [5] for $p = 0, 1$ and 2 respectively.

A strong and flexible modeling strategy for coupled tensor factorizations is based on exploiting the close connection between β-divergences and a particular exponential family, the so-called Tweedie models [6]. It turns out that Tweedie densities, to be described in more detail in the supplementary document, can be written in the following moment form

$$\mathbb{P}(x;\hat{x},\phi,p) = \frac{1}{K(x,\phi,p)} \exp\left(-\frac{1}{\phi} d_p(x;\hat{x})\right) \tag{4}$$

where $K(\cdot)$ is the normalizing constant, $\hat{x}$ is the mean, ϕ is the dispersion and p is the power parameter of the β-divergence defined in Eq.3. An important property is that the normalization constant K does not depend on $\hat{x}$; hence it is easy to see that for fixed p and ϕ, solving a maximum likelihood problem for $\hat{x}$ is indeed equivalent to minimization of the β-divergence. Here, different choices of p yield well-known important distributions such as Gaussian ($p = 0$), Poisson ($p = 1$), compound Poisson ($1 < p < 2$), gamma ($p = 2$) and inverse Gaussian ($p = 3$) distributions. Excluding the interval $0 < p < 1$ for which no exponential dispersion model exists, for all other values of p, one obtains Tweedie stable distributions [7]. To be described in more detail in the following section, in this probabilistic setting, the power parameter p will determine the divergence function (e.g., D_1 or D_2 in Eq.2), the dispersion ϕ will determine the relative weight of the cost, and the mean parameter $\hat{x}$ will be the output of the desired factorization.

Apart from estimation of the latent factors (e.g., Z_1, Z_2, Z_3), which is the primary task in coupled factorization models, further problems arise within these models that can be summarized as follows:

- *Estimation of the dispersions:* The dispersion parameters ϕ_1 and ϕ_2 play a key role in coupled factorizations as they form the balance between the approximation error to X_1 and X_2, for example observations may have been recorded using different and unknown scales. Typically, such weight parameters are selected manually [8, 9] and data is assumed to be suitably preprocessed. In a statistical setting, these relative weights are directly proportional to the observation noise variances and can be estimated directly from data. Regarding this problem, in [10] we have developed a method for maximum a-posteriori (MAP) inference for ϕ for the cases where $p \in \{0, 1, 2, 3\}$. In [11], we have presented two variational methods for estimating ϕ for $1 < p < 2$. Recently, we have developed a generalized method for dispersion estimation that covers the whole Tweedie family [12].

- *Automatic selection of the divergences:* Euclidean divergence is commonly used in tensor models, implicitly related to a conditionally Gaussian noise assumption. However, heavy-tailed noise distributions are often needed for robust estimation and more specific noise models are needed for sparse data, where Gaussian assumptions fall short. Choosing suitable divergence functions D_1 and D_2 (i.e., estimating p_1 and p_2) becomes even more critical in coupled models due to the data heterogeneity, where X_1 and X_2 may have different statistical characteristics. In such cases, it is useful to choose a specific divergence for each observed matrix, where we call total cost functions such as Eq.2 as *mixed divergences*. The state-of-the-art for estimating the power parameters in Tweedie models is based on numerical methods such as a grid search procedure [11, 12, 13, 14].

- *Handling arbitrary model topologies:* So far, we have motivated the tasks on the example model of Eq.1. However, in applications, one often needs to develop custom model topologies, where either the observed objects or the latent factors have multiple entities and cannot be represented without loss of structure using a matrix. To have this modeling flexibility for real world data sets that may consist of several tensors and require custom models, we would like to develop an algorithmic framework to handle a broad variety of model topologies. In this study, we make use of the Generalized Coupled Tensor Factorization (GCTF) framework [15] that aims to cover all possible model topologies and coupled factorization models.

In this paper, we will be concerned with distributed and parallel inference in coupled tensor factorization models. We present a parallel and distributed algorithmic framework for coupled tensor factorizations where we address all the aforementioned problems in a large-scale setting. Here, we envision a distributed-data/distributed-processing scenario, where each observed tensor (e.g., X_1, X_2) may reside at a different site (for example a cluster or a multicore machine) and each site has multiple processors with parallel, whilst limited, computational capacity. Our approach to optimization is inspired by the distributed and parallel stochastic gradient descent method for matrix factorizations (MF) in [16, 17]. These methods make use of the conditional independence structure of matrix factorizations, where the data is carefully segmented into mutually disjoint blocks that can be processed in parallel. Recently, [18] adapted this method to coupled tensor factorization (a Parafac-MF decomposition). However, this algorithm is focused only on estimating the factors of this specific model topology, without divergence or dispersion estimation. This algorithm also does not respect data locality and requires that blocks of observed tensors are distributed a-priori to each processor, possibly resulting in a heavy communication overhead. Also, for certain privacy critical applications, sending data across sites may be undesirable. In contrast, our approach is geared towards the distributed data scenario and we never need to transmit any data across sites, yet still being a valid incremental gradient method. We illustrate the proposed approach on synthetic data, where we focus on computationally less intensive; yet very widely used cases of the Tweedie distribution. Our experiments on a coupled matrix factorization problem show that the speed of the proposed algorithm is nearly proportional the number of processing nodes.

2 Probabilistic Modeling of Coupled Tensor Factorizations

In this study, we would like to deal with a broad variety of models and we rigorously develop our non-standard tensor notation [15], that aims to cover all possible model topologies and coupled factorization models. In this notation, a tensor is an N-way array.[1] We will denote tensors with capital letters, such as A, with its elements denoted by $A(i_1, i_2, \ldots, i_N)$. Here, A has N distinct indices $i_1, i_2, \ldots, i_N$. We let $I = [N]$ to be the index set of A, where $[N]$ denotes the set $\{1, 2, \ldots, N\}$. Each index i_k for $k \in I$ runs from 1 to its cardinality, denoted by s_k; i.e., we have $i_k \in \{1 \ldots s_k\}$, alternatively, $i_k \in [s_k]$. Each element of A is a real number and we write $A \in \mathbb{R}^{s_1 \times \cdots \times s_N}$.

Example 1. Suppose we have a 3-way tensor $A(i_1, i_2, i_3)$ where $i_1 \in \{1, 2\}$, $i_2 \in \{1, 2, 3\}$ and $i_3 \in \{1, 2\}$. Then, the index set is $I = \{1, 2, 3\}$, $N = 3$, and the index sizes are $s_1 = 2$, $s_2 = 3$, $s_3 = 2$; hence $A \in \mathbb{R}^{2\times3\times2}$.

In order to be able to handle a broad variety of tensor models and avoid unnecessary model specific notation, we index the elements of tensors with *index configurations*. An index configuration v is an

[1]We refer vectors as tensors with $N = 1$ index and matrices as tensors with $N = 2$ indices.

N-tuple from the product space of the domains of all indices defined as:

$$v \in \mathcal{C}_I(I) \equiv \prod_{k \in I} [s_k] = [s_1] \times [s_2] \times \cdots \times [s_N].$$

We will call the set $\mathcal{C}$ as the set of index configurations. Given an index configuration v, we will write $v(k)$ to refer specifically to the value of the index i_k, i.e., $v(k) = i_k$. Given the index set I, a tensor element $A(i_1, i_2, \ldots, i_N)$ is denoted more compactly by $A(v)$.

Often, we need to iterate over configurations on a particular subset $I_\alpha \subset I$. Given a particular $v \in \mathcal{C}_I(I)$, we define v_α as the index configuration such that $v_\alpha(k) = v(k) = i_k$ for all $k \in I_\alpha$ and, $v_\alpha(k) = 1$ for all $k \notin I_\alpha$. That is,

$$v_\alpha \in \mathcal{C}_I(I_\alpha) = \prod_{k \in I} [s_k^{\mathbb{1}(k \in I_\alpha)}] = [s_1^{\mathbb{1}(1 \in I_\alpha)}] \times [s_2^{\mathbb{1}(2 \in I_\alpha)}] \times \cdots \times [s_N^{\mathbb{1}(N \in I_\alpha)}] \qquad (5)$$

where $\mathbb{1}(x) = 1$ if x is true, and $\mathbb{1}(x) = 0$ otherwise. We will call $\mathcal{C}_I(I_\alpha)$ as the set of index configurations of I_α with respect to the domain I where $I_\alpha \subset I$.

We define the *tensor contraction* as the operation of summing a tensor over a subset of its indices. Often, we are required to contract a tensor F with index set J over a subset of its indices to obtain a tensor $\hat{X}$ with index set $I_0 \subset J$. We define the index configurations $v \in \mathcal{C}_J(J)$, $v_0 \in \mathcal{C}_J(I_0)$, and $\bar{v}_0 \in \mathcal{C}_J(\bar{I}_0)$ where $\bar{I}_0 = J \setminus I_0$ denotes the indices that are not present in $\hat{X}$. We write

$$\hat{X}(v_0) = \sum_{\bar{v}_0} F(v_0 \cup \bar{v}_0) = \sum_{\bar{v}_0} F(v),$$

where the union is defined as $(v_0 \cup \bar{v}_0) \in \mathcal{C}_J(I_0 \cup \bar{I}_0)$ such that $(v_0 \cup \bar{v}_0)(k) = v(k)$ for all k such that $i_k \in I_0 \cup \bar{I}_0$ and $(v_0 \cup \bar{v}_0)(k) = 1$ otherwise.

Example 2. Following Example 1, suppose we wish to sum the tensor A over i_2 to obtain the tensor $\hat{X}$. Here, the result $\hat{X}$ would be a tensor with two indices i_1 and i_3, with the index set as $I_0 = \{1, 3\}$. In traditional notation, the contraction would be denoted as $\hat{X}(i_1, i_3) = \sum_{i_2} A(i_1, i_2, i_3)$. Instead of dropping index i_2, we write the result of the contraction as $\hat{X}(i_1, 1, i_3) = \sum_{i_2} A(i_1, i_2, i_3)$, adopting the convention that $\hat{X}(i_1, 1, i_3)$ and $\hat{X}(i_1, i_3)$ refer to the same object. Hence the summation over the index i_2 is equivalent to computing $\hat{X}(i_1, 1, i_3) = A(i_1, 1, i_3) + A(i_1, 2, i_3) + A(i_1, 3, i_3)$ for each possible configurations of the pair i_1, i_3. This is exactly the elements of the sets of index configurations of I_0 given as $\mathcal{C}_I(I_0) = \{(1,1,1), (1,1,2), (2,1,1), (2,1,2)\}$. We have $\bar{I}_0 = I \setminus I_0 = \{2\}$ and the set of index configurations of $\bar{I}_0$ is: $\mathcal{C}_I(\bar{I}_0) = \{(1,1,1), (1,2,1), (1,3,1)\}$.

Similarly, we define the *tensor product* as the operation of multiplying two tensors. Two tensors Z_1 and Z_2 with index sets I_1 and I_2 can be mutiplied to obtain the tensor product G on the index set $J = I_1 \cup I_2$. We write $G(w) = Z_1(v_1) Z_2(v_2)$ where $w = v_1 \cup v_2$. More generally, for index sets $I_\alpha \subset J$ where $J = \cup_\alpha I_\alpha$ for $\alpha \in [N_z]$, we let $w \in \mathcal{C}_J(J)$ and $v_\alpha \in \mathcal{C}_J(I_\alpha)$. The product is a tensor G with the index set J, and Z_α are a collection of tensors, each with the index set I_α. When $w = v_1 \cup v_2 \cup \cdots \cup v_{N_z}$, we write

$$G(w) = \prod_{\alpha \in [N_z]} Z_\alpha(v_\alpha) = Z_1(v_1) Z_2(v_2) \ldots Z_{N_z}(v_{N_z}).$$

The Generalized Coupled Tensor Factorization (GCTF) framework is a statistical model for multiple observed tensors X_ν for $\nu \in [N_x]$. Each observed tensor X_ν is approximated by a model output tensor $\hat{X}_\nu$ that is obtained by the product of some latent tensors Z_α for $\alpha \in [N_z]$. The model is defined as follows:

$$X_\nu(u_\nu) \approx \hat{X}_\nu(u_\nu) = \sum_{\bar{u}_\nu} \prod_\alpha Z_\alpha(v_\alpha)^{R(\nu,\alpha)} \qquad (6)$$

Here, $u_\nu \in \mathcal{C}_I(I_{0,\nu})$ denotes the index configuration of X_ν and $v_\alpha \in \mathcal{C}_I(I_\alpha)$ denotes the index configuration of Z_α, where $I_{0,\nu}$ and I_α denote the index sets of an observed tensor X_ν and a latent factor Z_α, respectively. The set of all indices of a GCTF model is denoted by the index set $I = \{1, 2, \ldots, N\}$; it is the union of all indices used in the model, both the indices of observed and latent tensors: $I = (\cup_\nu I_{0,\nu}) \cup (\cup_\alpha I_\alpha)$. R is an $N_x \times N_z$ matrix with binary entries (0 or 1) that describes the coupling structure of a GCTF model, where each entry $R_{\nu\alpha}$ specifies if the model output tensor $\hat{X}_\nu$ is a function of Z_α. Finally, $\bar{u}_\nu \in \mathcal{C}_I(\bar{I}_{0,\nu})$ denote the index configurations that $\hat{X}_\nu$ is contracted on, where $\bar{I}_{0,\nu} = (\cup_{\alpha, R_{\nu\alpha}=1} I_\alpha) \setminus I_{0,\nu}$. The factors Z_α are called *local* if they are connected to a single observed tensor ($\sum_\nu R_{\nu\alpha} = 1$) and *shared* if they are connected to multiple observed tensors ($\sum_\nu R_{\nu\alpha} > 1$).

Example 3. Let us illustrate the model defined in Eq.1 in the GCTF notation. There are $N_x = 2$ observed tensors in the model (X_1 and X_2). There are $N_z = 3$ latent tensors (Z_1, Z_2, Z_3). The set of all indices I is defined as $I = \{1, 2, 3, 4\}$, the index sets of the observed tensors $I_{0,\nu}$ are defined as $I_{0,1} = \{1, 3\}$ and $I_{0,2} = \{2, 3\}$, and the index sets of the latent factors I_α are defined as $I_1 = \{1, 4\}$, $I_2 = \{2, 4\}$, and $I_3 = \{3, 4\}$. The sets of contraction indices are given as $\bar{I}_{0,1} = \bar{I}_{0,2} = \{4\}$. The coupling matrix for this model is given as follows: $R = [1\ 0\ 1\ ; 0\ 1\ 1]$.

The GCTF framework assumes the following probabilistic model on the observed tensors:

$$X_\nu(u_\nu) \sim \mathcal{TW}_{p_\nu}\big(X_\nu(u_\nu); \hat{X}_\nu(u_\nu), \phi_\nu\big), \qquad \nu = 1, \dots, N_x \tag{7}$$

where $\mathcal{TW}$ denotes Tweedie distribution and $\hat{X}_{1:N_x}$ are the model output tensors that are defined in Eq.6. The key contribution of this study is to simultaneously estimate latent factors $Z_{1:N_z}$, specific divergences for each dataset $p_{1:N_x}$ as well as the dispersions $\phi_{1:N_x}$ in a parallel and distributed manner, where the number of observed tensors, the number of latent factors, and the model topologies can be arbitrary.

3 Inference

In this study we propose a parallel and distributed framework for maximum a-posteriori estimation, where the aim is to solve the following optimization problem:

$$\max_{\substack{Z_{1:N_z} \\ \phi_{1:N_x}, p_{1:N_x}}} \left[-\sum_{\nu, u_\nu} \Big(\log K(X_\nu(u_\nu), \phi_\nu, p_\nu) + \frac{1}{\phi_\nu} d_p(X_\nu(u_\nu) || \hat{X}_\nu(u_\nu)) \Big) + \sum_\nu \log \mathbb{P}(\phi_\nu) \right] \tag{8}$$

where $\mathbb{P}(\phi_\nu)$ is the conjugate prior distribution of the dispersions, that is an inverse gamma distribution: $\phi_\nu \sim \mathcal{IG}(\phi_\nu; \tau_\phi, \kappa_\phi)$. Note that, when the dispersion and power parameters are known, the normalizing constant $K(\cdot)$ becomes irrelevant and the problem reduces to divergence minimization. However, when these parameters are not known, we need to deal with the normalizer $K(\cdot)$, an expression without a simple closed form apart from the cases $p = 0, 1, 2, 3$.

In order to estimate the latent factors, dispersions, and power parameters jointly, we utilize an iterative schema where we divide the optimization problem of Eq.8 into simpler subproblems. The ultimate method is a coordinate descent algorithm, where each parameter is updated at each iteration given the up-to-date values of the remaining parameters. Here, each iteration i consists of three estimation steps, stated as follows:

$$Z_\alpha^{(i+1)} = \arg\max_{Z_\alpha} \sum_{\nu=1}^{N_x} \log \mathbb{P}(X_\nu | Z_{1:N_z}, \phi_\nu^{(i)}, p_\nu^{(i)}), \qquad \alpha = 1, \dots, N_z \tag{9}$$

$$\phi_\nu^{(i+1)} = \arg\max_{\phi_\nu} \log\big(\mathbb{P}(X_\nu | \phi_\nu, \hat{X}_\nu^{(i+1)}, p^{(i)}) \mathbb{P}(\phi_\nu)\big), \qquad \nu = 1, \dots, N_x \tag{10}$$

$$p_\nu^{(i+1)} = \arg\max_{p_\nu} \log \mathbb{P}(X_\nu | \hat{X}_\nu^{(i+1)}, \phi_\nu^{(i+1)}, p_\nu), \qquad \nu = 1, \dots, N_x \tag{11}$$

In [11, 12], we focused on making inference in the challenging cases of the Tweedie distribution, where we evaluated our methods on regular-sized data due to computational requirements. Here, we will develop parallel and distributed methods for solving each of these subproblems in large-scale settings.

3.1 Distributed Incremental Gradient Descent for Learning the Factors

Given the dispersion and power parameters, finding the optimal factors $Z_{1:N_z}$ reduces to the problem of minimizing the β-divergence between the observations and the product of the latent factors, given

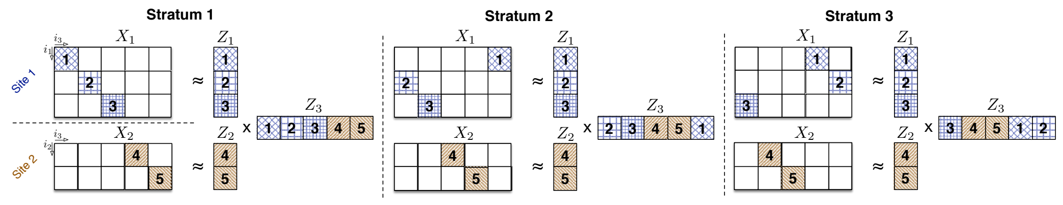


Figure 1: Illustration of DIGD on the simple model. The blocks are obtained by partitioning the observed indices i_1, i_2, i_3 into $N_1 = 3$, $N_2 = 2$, and $N_3 = 5$ pieces, respectively. There are 2 different sites and 5 nodes in total; the first site has 3 nodes and the second site has 2 nodes. The nodes are represented with different textures. The numbers inside the blocks indicate the nodes in which the corresponding blocks are located. In this example, at each sub-iteration, node i transfers its shared factor block to node $(i \mod 5) + 1$. This strategy implicitly determines the stratum at each sub-iteration.

as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{\nu} \sum_{u_\nu} \frac{1}{\phi_\nu} d_{p_\nu}\big(X_\nu(u_\nu) || \sum_{\bar{u}_\nu} \prod_{\alpha} Z_\alpha(v_\alpha)^{R(\nu,\alpha)}\big) \\ \text{subject to} \quad & Z_\alpha(v_\alpha) \in C, \quad \forall \alpha \in [N_z], v_\alpha \in \mathcal{C}_I(I_\alpha) \end{aligned}$$

where C is a constraint set. For simplicity, we define the iterate u that covers all (ν, u_ν) pairs and rewrite the cost function as follows:

$$\text{minimize} \quad \sum_{u=1}^{U} f_u(Z_1, \dots, Z_{N_z}),$$

subject to the same constraints and $f_u(\cdot)$ is the appropriate cost function. If U is sufficiently small, standard algorithms [15, 19] that minimize the β-divergence can be used here. However, when U is large, operating on the entire cost function becomes impractical. Besides, in large-scale applications the data is usually distributed among many processing units, which brings motivation to utilize distributed incremental methods.

Incremental gradient methods are powerful algorithms that operate on a single sample u at each iteration, rather than the entire cost function [20]. The idea is to take a small step at each iteration in the opposite direction of the gradient computed on the sample u. The order in which the samples are selected is an important choice and might result in different computational requirements depending on the problem structure. Here, we exploit the conditional independence structure of coupled tensor factorizations and construct a specific ordering by rearranging the cost function as follows:

$$\text{minimize} \quad \sum_{s} \sum_{b \in \Omega_s} \sum_{u \in \Omega_b} f_u(Z_1^b, \dots, Z_{N_z}^b)$$

where b is called as a block, Ω_b denotes the data points in block b, s is called as a stratum, and Ω_s denotes the blocks in stratum s. In large-scale applications, these blocks and the corresponding factors Z_α^b will be fully distributed. The key point here is to form each stratum in such a way that the parameter spaces of the blocks in a stratum become disjoint; enabling these blocks to be processed in parallel. In other words, we would like to iterate sequentially over the index s and process the summation over b in parallel. We now describe how to form such blocks and strata for coupled tensor factorization that supports data locality.

We start by defining a partition for each index i_k. For the observed indices ($k \in \cup_\nu I_{0,\nu}$), we define the partition $P_k = \{(b_k)_1, \dots, (b_k)_{N_k}\}$ on the set $[s_k]$, where $(b_k)_i$ are non-empty, disjoint subsets of $[s_k]$ and N_k denotes the number of subsets in P_k. We do not need partitions for the hidden indices ($k \notin \cup_\nu I_{0,\nu}$), however for notational consistency, we define a partition with a single element for each hidden index: $P_k = \{(b_k)_1\}$, where $(b_k)_1 = [s_k]$. Then a *block* of an observed tensor $X_\nu(u_\nu)$ is specified by a set of index configurations defined as follows:

$$u_\nu \in \mathcal{B}_I(I_{0,\nu}, \gamma) = \prod_{k \in I} (b_k)_{\gamma_k}^{\mathbb{1}(k \in I_{0,\nu})} = (b_1)_{\gamma_1}^{\mathbb{1}(1 \in I_{0,\nu})} \times (b_2)_{\gamma_2}^{\mathbb{1}(2 \in I_{0,\nu})} \times \cdots \times (b_N)_{\gamma_N}^{\mathbb{1}(N \in I_{0,\nu})}$$

where we define $S^1 = S$ and $S^0 = \{1\}$ for the set S. Here, γ is an array of integers that fully determine the structure of a block, where each $\gamma_k \in [N_k]$ determines the element that will be selected from P_k, so that $(b_k)_{\gamma_k}$ is the γ_k^{th} element of P_k. The corresponding blocks for the factors are defined similarly; these blocks consist of all $Z_\alpha(v_\alpha)$ with $v_\alpha \in \mathcal{B}_I(I_\alpha, \gamma)$. In practice, the data is usually sparse and partitioning the data into balanced blocks can be an important but a highly nontrivial task. Here, we utilize a simple heuristic approach for data-dependent index partitioning where each observed index is partitioned separately.

A stratum $\sigma = \{\gamma^1, \gamma^2, \ldots, \gamma^{N_s}\}$ is a collection of N_s non-overlapping blocks, where for each pair of blocks we have $\gamma_k^i \neq \gamma_k^j$ for all k in the index set I and $i \neq j$. Since the elements of the partitions are disjoint by definition and there are no common elements in two different blocks, the parameter spaces of the blocks in a stratum will also be disjoint by construction. This enables the blocks in a stratum to be processed in parallel. Moreover, we always choose strata such that the blocks in any stratum obey the data locality as illustrated in Fig 1.

Here, we propose a distributed incremental gradient descent (DIGD) algorithm for coupled tensor factorizations, where we sequentially iterate over the strata and process the blocks in a strata in parallel. One iteration of DIGD consists of several sub-iterations, where a stratum σ is implicitly determined at each sub-iteration by the schedule that the blocks of the shared factors are transferred among the nodes. After σ is formed, we run the IGD updates in parallel for each block in σ. The sub-iterations are continued until all the entries of the observed tensors are visited. Here, first we assign the blocks of local factors to the processors, then we circulate the blocks of the shared factors among the processors (see Fig 1). The pseudo-code for the overall algorithm and the required partial derivatives are provided in the supplementary document. Note that, we run DIGD until convergence at each iteration of the overall method defined in Eqs 9-11, therefore each iteration of the overall algorithm consists of several iterations of DIGD. The proposed algorithm is a valid IGD algorithm; provided all the elements of the observed tensors are processed at each iteration, the proof of convergence proceeds in the same lines as in [20].

3.2 Parallel and Distributed Estimation of Mixed Divergences

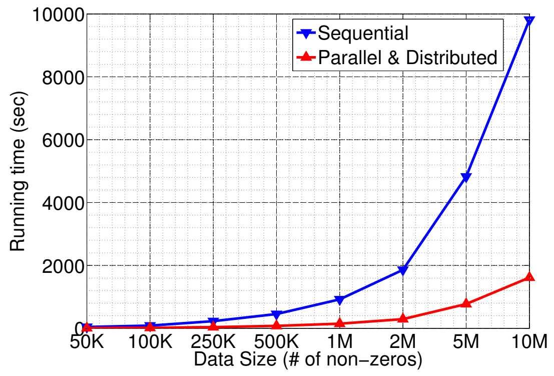
The MAP estimate of ϕ has analytical solution only for the Gaussian and the inverse Gaussian distributions. Inferring ϕ is intractable for the other cases. In our previous work [10], we showed that the inference becomes tractable for the Poisson and gamma distributions when the gamma functions in the probability mass and density functions are approximated with Stirling's approximation. The MAP estimate of ϕ for the cases $p \in \{0, 1, 2, 3\}$ is given as follows: [10]

$$\phi_\nu^\star = \frac{\left(\sum_{u_\nu} d_{p_\nu}(X_\nu(u_\nu) || \hat{X}_\nu(u_\nu))\right) + \kappa_\phi}{S_\nu/2 + \tau_\phi + 1}, \quad p \in \{0, 1, 2, 3\} \tag{12}$$

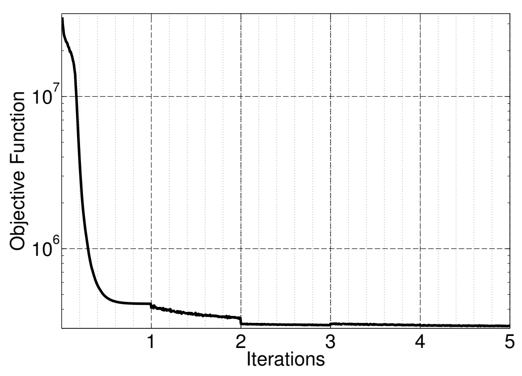
where S_ν is the number of elements in X_ν and $d_p(\cdot)$ is the β-divergence, defined in Eq.3. In this study, we focus on computationally less intensive; yet very widely used cases of the Tweedie distribution: $p \in \{0, 1, 2\}$. It is also straightforward to extend the proposed method for the remaining cases of p [12].

The last step of the proposed method (Eq.11) is to compute the maximum likelihood estimate of the power parameter p. Unfortunately, the optimal p does not have an analytical solution; the state-of-the-art is based on running numerical methods on this problem. In this study, we follow [11, 12] and utilize a grid search procedure in order to estimate the power parameter p given the other parameters.

Computing Eq.12 and the likelihood values for the grid search is inherently parallelizable since the problem is separable over the index configurations u_ν. In the distributed setting, after the DIGD procedure converges, each node starts to compute the relevant part of the computations (note that, these local computations can also be computed in a parallel fashion by using multicore CPUs) and passes the shared factor block to the next node. This procedure is continued until all the nodes complete their corresponding computation after passing over all the elements of the observed tensors. Finally, the nodes report the intermediate computations to a responsible node in their own site. The responsible nodes aggregate all the intermediate computations and compute the new ϕ_ν and p_ν values. These values are then broadcasted to the relevant nodes.



(a)



(b)

Figure 2: a) The running times of the algorithms for different data sizes b) The value of the objective function over the iterations for $s_1 = s_2 = s_3 = 320$ and $s_4 = 5$. These iterations correspond to the overall algorithm defined in Eqs 9-11, where at each iteration DIGD is run until convergence.

4 Preliminary Experiments

In this section, we illustrate the proposed method on the simple model defined in Eq.1. Here, we randomly generate the latent variables $Z_{1:3}$, $\phi_{1:2}$, power parameters $p_{1:2}$, and the observed tensors $X_{1:2}$. Our aim is to estimate all the latent variables given X_1 and X_2.

In this study, we implement the distributed inference algorithm by a message passing protocol in C using the OpenMPI[2] library. The algorithm is particularly suited for message passing and low level control on the distributed computations provide more insight then other platforms such as Hadoop MapReduce[3]. We conduct our experiments in a pseudo-distributed setting where we use an 8-core MacPro with 32 GB of memory. Thanks to the flexibility of OpenMPI library, it is possible to run the same code on any MPI cluster without any modifications.

In our experiments, we partition the observed indices i_1 and i_2 into $N_1 = N_2 = 4$ parts and i_3 into $N_3 = 8$ parts. In the DIGD procedure, we set the step size $\eta^{(i)} = (a/i)^b$, where i denotes the iteration number. An important observation is that, for the shared factors, the value of a should be smaller than the value that is chosen for the local factors, in order to have a faster convergence. Here, we set $a = 0.01$ for the local factors and $a = 0.001$ for the shared factors. The value of b is set to 0.51 for all factors.

Figure 2(a) shows the comparison between the proposed approach and the sequential implementation in terms of running times. The picture reveals that we are able to get a 6 fold performance increase on a 8 core machine when we have 10 million entries in the observed matrices. Figure 2(b) shows that the objective is decreased quickly during the iterations, with jumps corresponding to the estimation of divergences and dispersions.

5 Conclusion

We presented a parallel and distributed algorithmic framework for coupled tensor factorization to simultaneously estimate latent factors, specific divergences for each dataset as well as the relative weights in an overall additive cost function, where the number of observed tensors, the number of latent factors, and the model topologies can be arbitrary. We illustrated the proposed method on synthetic data experiments. Our experiments on a coupled matrix factorization problem showed that the speed of the proposed algorithm is nearly proportional to the number of processing nodes.

[2]`http://www.open-mpi.org`
[3]`http://hadoop.apache.org`

References

[1] L. Le Magoarou, A. Ozerov, and N. Q. K. Duong, "Text-informed audio source separation using nonnegative matrix partial co-factorization," in *MLSP*, 2013.

[2] T.F. Wilderjans, E. Ceulemans, I. Van Mechelen, and R.A. van den Berg, "Simultaneous analysis of coupled data matrices subject to different amounts of noise.," *Br J Math Stat Psychol*, vol. 64, pp. 277–90, 2011.

[3] E. Acar, G. Gurdeniz, M. A. Rasmussen, D. Rago, L. O. Dragsted, and R. Bro, "Coupled matrix factorization with sparse factors to identify potential biomarkers in metabolomics.," *IJKDB*, vol. 3, no. 3, pp. 22–43, 2012.

[4] V. W. Zheng, B. Cao, Y. Zheng, X. Xie, and Q. Yang, "Collaborative filtering meets mobile recommendation: A user-centered approach," in *AAAI'10*, 2010.

[5] C. Févotte, N. Bertin, and J. L. Durrieu, "Nonnegative matrix factorization with the Itakura-Saito divergence. with application to music analysis," *Neural Computation*, vol. 21, pp. 793–830, 2009.

[6] Y. K. Yılmaz and A. T. Cemgil, "Alpha/beta divergences and tweedie models," *arXiv:1209.4280 v1*, 2012.

[7] B. Jørgensen, *The Theory of Dispersion Models*, Chapman & Hall/CRC Monographs on Statistics & Applied Probability, 1997.

[8] M. Kim, J. Yoo, K. Kang, and S. Choi, "Nonnegative matrix partial co-factorization for spectral and temporal drum source separation," *J. Sel. Topics Signal Processing*, vol. 5, no. 6, pp. 1192–1204, 2011.

[9] T. Barker, T. Virtanen, and O. Delhomme, "Ultrasound-coupled semi-supervised nonnegative matrix factorisation for speech enhancement," in *ICASSP*, 2014.

[10] U. Şimşekli, B. Ermiş, A. T. Cemgil, and E. Acar, "Optimal weight learning for coupled tensor factorization with mixed divergences," in *EUSIPCO*, 2013.

[11] U. Şimşekli, A. T. Cemgil, and Yılmaz K., "Learning the beta-divergence in tweedie compound poisson matrix factorization models," in *ICML*, 2013.

[12] Umut Simsekli, Ali Taylan Cemgil, and Beyza Ermis, "Learning mixed divergences in coupled matrix and tensor factorization models," in *submitted to ICASSP*, 2015.

[13] P. K. Dunn and G. S. Smyth, "Series evaluation of tweedie exponential dispersion model densities," *Stats. & Comp.*, vol. 15, pp. 267–280, 2005.

[14] Y. Zhang, "Likelihood-based and bayesian methods for tweedie compound poisson linear mixed models," *Statistics and Computing*, vol. accepted, 2012.

[15] Y. K. Yılmaz, A. T. Cemgil, and U. Şimşekli, "Generalised coupled tensor factorisation," in *NIPS*, 2011.

[16] Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in *ACM SIGKDD*, 2011.

[17] Benjamin Recht and Christopher Ré, "Parallel stochastic gradient algorithms for large-scale matrix completion," *Mathematical Programming Computation*, 2013.

[18] Alex Beutel, Abhimanu Kumar, Evangelos E. Papalexakis, Partha Pratim Talukdar, Christos Faloutsos, and Eric P. Xing, "Flexifact: Scalable flexible factorization of coupled tensors on hadoop," in *SDM*, 2014.

[19] A. Cichocki, R. Zdunek, A. H. Phan, and S. Amari, *Nonnegative Matrix and Tensor Factorization*, Wiley, 2009.

[20] D. P. Bertsekas, "Incremental Gradient, Subgradient, and Proximal Methods for Convex Optimization: A Survey," 2010.