Distributed Flexible Nonlinear Tensor Factorization for Large Scale Multiway Data Analysis

Shandian Zhe
Purdue University
szhe@purdue.edu

Pengyuan Wang
Yahoo! Labs
pengyuan@yahoo-inc.com

Kuang-chih Lee
Yahoo! Labs
kclee@yahoo-inc.com

Zenglin Xu
University of Electronic Science and Technology of China
zlxu@uestc.edu.cn

Jian Yang
Yahoo! Labs
jianyang@yahoo-inc.com

Youngjia Park
IBM Thomas J. Watson Research Center
young_park@us.ibm.com

Yuan Qi
Purdue University
alanqi@cs.purdue.edu

Abstract

Tensor factorization is an important approach to multiway data analysis. However, real-world tensor data often encompass complex interactions among tensor elements, and are extremely sparse and of huge size. Despite the success of exiting approaches, they are either not powerful enough to model the complex interactions or extreme sparsity in data. To overcome these limits, we propose a new tensor factorization model in this paper. It employs Gaussian process (GP) to capture the complex nonlinear relationships between tensor modes, and employs random functions over any subset of tensor elements to avoid the expensive computation of Kronecker products and to flexibly incorporate meaningful tensor entries for training. To scale up the model to large tensor data, we propose a distributed variational inference algorithm on MAPREDUCE, which uses a tight variational evidence lower bound (ELBO) to improve inference quality and to prevent inefficient Expectation-Maximization procedure. In implementation, we design non-key-value MAP-REDUCE to prevent expensive data shuffling and to fully exploit the memory-cache mechanism in efficient MAPREDUCE systems, such as SPARK. Experimental results have demonstrated our method's advantages in both the predictive performance and computational efficiency over the existing approaches; moreover, our approach shows promising results in the application of Click-Through-Rate (CTR) prediction for online advertising.

1 Introduction

Multiway data, described by tensors, are common in real applications. For example, online advertising data can be represented by a tensor with three modes (*user*, *advertisement*, *context*) and patient-medicine response by a tensor with four modes (*person*, *medicine*, *biomarker*, *time*). Given tensor-valued data, the goal is to capture hidden relationships embedded in data and to predict unobserved entries (*e.g.,* unknown click behavior for online advertisement). This is usually done through tensor factorization. However, the design of a tensor factorization algorithm for real-world applications can be challenging, because i) the interaction between tensor modes may be complex and highly nonlinear; ii) tensor data are usually very sparse and consist of very few nonzero elements; and iii) tensor data are often of huge size, often containing millions of nonzero entries and billions of zeros entries.

While many tensor factorization algorithms have been proposed, they rarely address all these challenges. Standard multilinear methods, such as Tucker decomposition [15] and CANDECOMP/PARAFAC (CP) [4] are difficult to model complex and nonlinear interactions between tensor modes. Although nonlinear methods such as Infinite Tucker Decomposition(InfTucker) [16] and its extensions [17, 18], can overcome this limit by using tensor-variate Gaussian processes (GP) [16], they are computationally expensive since the Kronecker product between the covariances of all the modes forces the GP to rigidly model an entire tensor. What's more, they are incapable to model the extreme sparsity of data, i.e., the imbalanced amount of nonzero entries. As we know, in real data, most zero elements are meaningless—they are just missing or unobserved, thus employing all the zero entries in the modelling may affect the factorization quality and lead to biased prediction.

To address these issues, we propose a new and flexible nonlinear tensor factorization model. The model places Gaussian process priors over tensor entries, which means that the input is constructed by concatenating latent factors in each mode and the nonlinear relationship is captured by various kernel functions. In this way, the covariance function is free of the Kronecker product structure, and thus the model is more flexible to incorporate meaningful elements for training. Furthermore, to make the model scalable to large data, a distributed variational inference algorithm is developed based on MAPREDUCE: It employs a tight variational evidence lower bound (ELBO) on the optimal variational posteriors, which is derived by functional derivatives and nonlinear equations, to improve inference quality and to prevent inefficient EM procedure; it maintains a whole gradient vector in each MAPPER and remove (key,value) pairs to prevent expensive data shuffling, and to fully exploit the memory-cache mechanism in efficient MAPREDUCE systems, such as SPARK, for fast training.

For evaluation, our model is first examined on four small real world datasets where most of tensor factorization methods are feasible. It shows that our method obtains higher prediction accuracy than InfTucker and other multilinear methods. Then the examination on large tensor data with millions of nonzero elements has shown that our method can obtain much better predictive performance than GigaTensor, the state-of-the-art large-scale multilinear factorization algorithm, and significantly better than or comparable to scalable nonlinear methods based on tensor-variate GP. Moreover, our method achieves faster training speed and enjoys almost linear scalability on the number of computational nodes. Finally, our model is applied to CTR prediction for online advertising and achieves a 20% improvement over the broadly used logistic regression and linear SVM.

2 Background

A K-mode tensor is denoted by $\mathcal{M} \in \mathbb{R}^{m_1 \times \ldots \times m_K}$, where the k-th mode has m_k dimensions. The tensor entry at location $\mathbf{i}$ ($\mathbf{i} = (i_1, \ldots, i_K)$) is denoted by $m_{\mathbf{i}}$. The tensor $\mathcal{M}$ can be flatten into a vector, denoted by $\mathrm{vec}(\mathcal{M})$, where the entry $\mathbf{i} = (i_1, \ldots, i_K)$ in $\mathcal{M}$ is mapped to the entry at position $j = i_K + \sum_{k=1}^{K-1}(i_k - 1)\prod_{t=k+1}^{K} m_t$ in $\mathrm{vec}(\mathcal{M})$.

Given a tensor $\mathcal{W} \in \mathbb{R}^{r_1 \times \ldots \times r_K}$ and a matrix $\mathbf{U} \in \mathbb{R}^{s \times r_k}$, a mode-$k$ tensor-matrix product between $\mathcal{W}$ and $\mathbf{U}$ is denoted by $\mathcal{W} \times_k \mathbf{U}$, which is a tensor of size $r_1 \times \ldots \times r_{k-1} \times s \times r_{k+1} \times \ldots \times r_K$. The element-wise calculation is $(\mathcal{W} \times_k \mathbf{U})_{i_1 \ldots i_{k-1} j i_{k+1} \ldots i_K} = \sum_{i_k=1}^{r_k} w_{i_1 \ldots i_K} u_{j i_k}$.

Typical multilinear tensor factorization methods include Tucker and CANDECOMP/PARAFAC (CP). The Tucker decomposition is defined by

$$\mathcal{M} = [\![\mathcal{W}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(K)}]\!] = \mathcal{W} \times_1 \mathbf{U}^{(1)} \times_2 \ldots \times_K \mathbf{U}^{(K)} \quad (1)$$

where $\mathcal{W} \in \mathbb{R}^{r_1 \times \ldots \times r_K}$ is the core tensor, and $\mathbf{U}^{(k)} \in \mathbb{R}^{m_k \times r_k}$ is the k-th latent factor matrix. CP decomposition can be considered as a special instance of Tucker decomposition where the core tensor $\mathcal{W}$ is restricted to be diagonal, i.e., $r_1 = \ldots = r_K$ and $w_{i_1 \ldots i_K} \neq 0$ only if $i_1 = \ldots = i_K$.

The infinite Tucker decomposition (InfTucker) is a nonlinear tensor factorization model based on tensor-variate Gaussian process [16]. It can be derived from Tucker decomposition: First, we assign element-wise standard normal prior over the core tensor $\mathcal{W}$ in Equation (1); by marginalizing out $\mathcal{W}$, we can obtain the marginal distribution of tensor $\mathcal{M}$ given the latent factors:

$$p(\mathcal{M}|\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(K)}) = \mathcal{N}(\mathrm{vec}(\mathcal{M}); \mathbf{0}, \Sigma^{(1)} \otimes \ldots \otimes \Sigma^{(K)}) \quad (2)$$

where $\Sigma^{(k)} = \mathbf{U}^{(k)}\mathbf{U}^{(k)\top}$ and $\otimes$ is the Kronecker product. The kernel trick is then used to model nonlinear relationships: Each row $\mathbf{u}_t^k$ of the latent factors $\mathbf{U}^{(k)}$ is replaced with a nonlinear feature transformation $\phi(\mathbf{u}_t^k)$ and then an equivalent nonlinear covariance matrix $\Sigma^{(k)} = k(\mathbf{U}^{(k)}, \mathbf{U}^{(k)})$

is used to replace $\mathbf{U}^{(k)}\mathbf{U}^{(k)\top}$, where $k(\cdot,\cdot)$ is the covariance function. After feature mapping, the (invisible) core tensor $\mathcal{W}$ actually obtains the dimension of mapped feature vector in each mode, which could be infinite. Since the covariance of $\mathrm{vec}(\mathcal{M})$ is a function of the latent factors $\mathcal{U} = \{\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(K)}\}$, Equation (2) actually defines a Gaussian process (GP) on tensor, namely tensor-variate GP [16], where the input are based on the latent factors $\mathcal{U}$. Further, the standard Laplace prior is assigned over $\mathcal{U}$ to encourage sparse estimation for easy interpretation. Given $\mathcal{M}$, the observed tensor $\mathcal{Y}$ is sampled from a noisy model $p(\mathcal{Y}|\mathcal{M})$. For example, we can use Gaussian models and probit models for continuous and binary observations respectively. Thus the joint probability of InfTucker is $p(\mathcal{Y}, \mathcal{M}, \mathcal{U}) = p(\mathcal{U})p(\mathcal{M}|\mathcal{U})p(\mathcal{Y}|\mathcal{M})$.

3 Model

Although InfTucker and its extensions [17, 18] can capture nonlinear interactions in data via tensor-variate GP, they can suffer from the extreme sparsity in real data. The reason is that the covariance is a Kronecker product between the covariances over all the modes, i.e., $\{\Sigma^{(1)}, \ldots, \Sigma^{(K)}\}$ (see Equation (2)). Each $\Sigma^{(k)}$ is of size $m_k \times m_k$ and the full covariance is of size $\prod_k m_k \times \prod_k m_k$. Thus the GP is enforced to model the entire tensor associated with the latent factors $\mathcal{U}$, including all zeros and nonzero elements; there is no way to model a (meaningful) subset of entries instead. However, the real-world tensors are usually extremely sparse, with huge number of zero elements and a tiny portion of nonzero elements. On one hand, because most zero elements are meaningless—they are actually missing or unobserved, using them may affect the tensor factorization quality and lead to biased prediction; on the other hand, incorporating massive zero elements into GP models can result in large covariance matrix and high computational cost. Although [17, 18] improved the scalability by modelling subtensors instead, the sampled subtensors can still be very sparse. Even worse, because their size has to be small for fast computation, it is often easy to sample ones not containing any nonzero elements. This may further incur numerical instability in model estimation.

To address these issues, we propose a new, flexible Gaussian process tensor factorization model. While keeping the nonlinear modelling power, the model disposes of the Kronecker product structure in covariance and can use any set of tensor entries for training, rather than use an entire tensor exclusively. Specifically, given tensor $\mathcal{M} \in \mathbb{R}^{m_1 \times \cdots \times m_K}$, for each tensor entry $m_{\mathbf{i}}$ ($\mathbf{i} = (i_1, \ldots, i_K)$), we construct an input $\mathbf{x}_i$ by concatenating the corresponding latent factors in all the modes: $\mathbf{x}_{\mathbf{i}} = [\mathbf{u}^{(1)}_{i_1}, \ldots, \mathbf{u}^{(K)}_{i_K}]$, where $\mathbf{u}^{(k)}_{i_k}$ is the i_k-th row in latent factor matrix $\mathbf{U}^{(k)}$ for mode k. Note that $\mathbf{u}^{(k)}_{i_k}$ is a $1 \times m_k$ vector and $\mathbf{x}_{\mathbf{i}}$ is a $1 \times \sum_{j=1}^{K} m_j$ vector. We assume that there is an underlying function $f : \mathbb{R}^{\sum_{j=1}^{K} m_j} \rightarrow \mathbb{R}$ such that $m_{\mathbf{i}} = f(\mathbf{x}_{\mathbf{i}}) = f([\mathbf{u}^{(1)}_{i_1}, \ldots, \mathbf{u}^{(K)}_{i_K}])$. This function is unknown and can be complex and nonlinear. To learn the function, we assign a Gaussian process prior over f: For any set of tensor entries $S = \{\mathbf{i}_1, \ldots, \mathbf{i}_N\}$, the function values $\mathbf{f}_S = \{f(\mathbf{x}_{\mathbf{i}_1}), \ldots, f(\mathbf{x}_{\mathbf{i}_N})\}$ are distributed according to a multivariate Gaussian distribution with mean $\mathbf{0}$ and covariance decided by $\mathbf{X}_S = \{\mathbf{x}_{\mathbf{i}_1}, \ldots, \mathbf{x}_{\mathbf{i}_N}\}$: $p(\mathbf{f}_S|\mathcal{U}) = \mathcal{N}(\mathbf{f}_S|\mathbf{0}, k(\mathbf{X}_S, \mathbf{X}_S))$ where $k(\cdot,\cdot)$ is a (nonlinear) covariance function. Note that $k(\mathbf{x}_{\mathbf{i}}, \mathbf{x}_{\mathbf{j}}) = k([\mathbf{u}^{(1)}_{i_1}, \ldots, \mathbf{u}^{(K)}_{i_K}], [\mathbf{u}^{(1)}_{j_1}, \ldots, \mathbf{u}^{(K)}_{j_K}])$ and there is no Kronecker product structure, and any set of tensor entries can be used for training. To prevent from learning bias toward zero, we can use a set of entries with balanced zeros and nonzeros; further, domain knowledge can be incorporated to choose meaningful entries for training.

We further assign a standard normal prior over the latent factors $\mathcal{U}$; given tensor entries $\mathbf{m} = [m_{\mathbf{i}_1}, \ldots, m_{\mathbf{i}_N}]$, the observed tensor entries $\mathbf{y} = [y_{\mathbf{i}_1}, \ldots, y_{\mathbf{i}_N}]$ are sampled from a noisy model $p(\mathbf{y}|\mathbf{m})$. In this paper, we deal with both continuous and binary observations. For continuous data, we use Gaussian model, $p(\mathbf{y}|\mathbf{m}) = \mathcal{N}(\mathbf{y}|\mathbf{m}, \beta\mathbf{I})$ and the joint probability is

$$p(\mathbf{y}, \mathbf{m}, \mathcal{U}) = \prod_{t=1}^{K} \mathcal{N}(\mathrm{vec}(\mathbf{U}^{(t)})|\mathbf{0}, \mathbf{I})\mathcal{N}(\mathbf{m}|\mathbf{0}, k(\mathbf{X}_S, \mathbf{X}_S))\mathcal{N}(\mathbf{y}|\mathbf{m}, \beta\mathbf{I}) \tag{3}$$

where $S = [\mathbf{i}_1, \ldots, \mathbf{i}_N]$. For binary data, we use probit model: We first introduce augmented variables $\mathbf{z} = [z_1, \ldots, z_N]$ and then decompose the probit model into $p(z_j|m_{\mathbf{i}_j}) = \mathcal{N}(z_j|m_{\mathbf{i}_j}, 1)$ and $p(y_{\mathbf{i}_j}|z_j) = \mathbb{1}(y_{\mathbf{i}_j} = 0)\mathbb{1}(z_j \le 0) + \mathbb{1}(y_{\mathbf{i}_j} = 1)\mathbb{1}(z_j > 0)$ where $\mathbb{1}(\cdot)$ is an indicator function. Then the joint probability is

$$\begin{aligned} p(\mathbf{y}, \mathbf{z}, \mathbf{m}, \mathcal{U}) = &\prod_{t=1}^{K} \mathcal{N}(\mathrm{vec}(\mathbf{U}^{(t)})|\mathbf{0}, \mathbf{I})\mathcal{N}(\mathbf{m}|\mathbf{0}, k(\mathbf{X}_S, \mathbf{X}_S))\mathcal{N}(\mathbf{z}|\mathbf{m}, \mathbf{I}) \\ &\cdot \prod_{j} \mathbb{1}(y_{\mathbf{i}_j} = 0)\mathbb{1}(z_j \le 0) + \mathbb{1}(y_{\mathbf{i}_j} = 1)\mathbb{1}(z_j > 0). \end{aligned} \tag{4}$$

4 Distributed Variational Inference

We now present our distributed variational inference algorithm in MAPREDUCE framework.

4.1 Variational Evidence Lower Bound Based on Sparse Gaussian Process

First, because the GP prior term $p(\mathbf{m}|\mathcal{U})$ couples all the latent factors together and makes parallel inference very difficult, we derive a tractable variational evidence lower bound (ELBO) for distributed inference, following the sparse Gaussian process framework [14].

Specifically, by introducing a set of inducing points $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_p\}$ and latent targets $\mathbf{v}$ and applying Jensen's inequality twice, a tractable ELBO of the proposed model for real-valued tensor is given by $\log(p(\mathbf{y},\mathcal{U}|\mathbf{B})) \geq L_1$ where

$$L_1 = \log(p(\mathcal{U})) + \int q(\mathbf{v})(\log(p(\mathbf{v}|\mathbf{B})) - \log(q(\mathbf{v}))) + \sum\nolimits_j \int q(\mathbf{v}) F_{\mathbf{v}}(y_{\mathbf{i}_j}, \beta)\mathrm{d}\mathbf{v} \tag{5}$$

where $q(\mathbf{v})$ is the variational posterior of the latent targets $\mathbf{v}$, $F_{\mathbf{v}}(y_{\mathbf{i}_j}, \beta) = \int \log\big(\mathcal{N}(y_{\mathbf{i}_j}|m_{\mathbf{i}_j}, \beta)\big)\mathcal{N}(m_{\mathbf{i}_j}|\mu_j, \sigma_j)\mathrm{d}m_{\mathbf{i}_j}$, $\mu_j = k(\mathbf{x}_{\mathbf{i}_j}, \mathbf{B})\mathbf{K}_{BB}^{-1}\mathbf{v}$ and $\sigma_j = \mathbf{\Sigma}(j,j) = k(\mathbf{x}_{\mathbf{i}_j}, \mathbf{x}_{\mathbf{i}_j}) - k(\mathbf{x}_{\mathbf{i}_j}, \mathbf{B})\mathbf{K}_{BB}^{-1}k(\mathbf{B}, \mathbf{x}_{\mathbf{i}_j})$. The ELBO for the binary tensor has a similar form to Equation (5) except that an extra variational posterior $q(\mathbf{z}) = \prod_j q(z_j)$ is introduced for the probit likelihood (See Equation (4)): $\log(p(\mathbf{y},\mathcal{U}|\mathbf{B})) \geq L_2$ where

$$\begin{aligned} L_2 = {} & \log(p(\mathcal{U})) + \int q(\mathbf{v})(\log(p(\mathbf{v}|\mathbf{B})) - \log(q(\mathbf{v}))) \\ & + \sum\nolimits_j \int q(\mathbf{v}) \int q(z_j) F_{\mathbf{v}}(z_j, 1)\mathrm{d}z_j \mathrm{d}\mathbf{v} + \sum\nolimits_j q(z_j)\log(\frac{p(y_{\mathbf{i}_j}|z_j)}{q(z_j)}) \end{aligned} \tag{6}$$

4.2 Tight Bound with Optimal Variational Posteriors

Variational inference maximizes the ELBO to estimate the latent factors $\mathcal{U}$, the pseudo input $\mathbf{B}$, and the posteriors $q(\mathbf{v})$ and $q(\mathbf{z})$. Typically, an Expectation-Maximization (EM) algorithm is used: In the E-step, we update the variational posteriors $q(\mathbf{v})$ and $q(\mathbf{z})$ given $\mathcal{U}$ and $\mathbf{B}$; in the M-step, we optimize $\mathcal{U}$ and $\mathbf{B}$ with $q(\mathbf{v})$ and $q(\mathbf{z})$ fixed. Although widely used, this algorithm may be low efficient and not suitable for parallelization, because the updating is sequential: Every step (E or M) depends on the previous step and it may take long time to converge.

To enable efficient distributed inference, we remove the E step, by using a tighter bound only depending on $\mathcal{U}$ and $\mathbf{B}$:

$$L_1^*(\mathcal{U}, \mathbf{B}) = \max_{q(\mathbf{v})} L_1(q(\mathbf{v}), \mathcal{U}, \mathbf{B}), \quad L_2^*(\mathcal{U}, \mathbf{B}) = \max_{q(\mathbf{v}), q(\mathbf{z})} L_2(q(\mathbf{v}), q(\mathbf{z}), \mathcal{U}, \mathbf{B}).$$

Then we only need to estimate $\mathcal{U}$ and $\mathbf{B}$ via distributed optimization. After the optimal $\mathcal{U}$ and $\mathbf{B}$ are obtained, we can calculate $q(\mathbf{v})$ and $q(\mathbf{z})$ accordingly. This not only avoids the sequential EM procedure, but also improves the inference quality, because a tighter ELBO is used.

Specifically, we calculate the functional derivatives with respect to $q(\mathbf{v})$ and $q(\mathbf{z})$ and solve the optimal $q(\mathbf{v})$ and $q(\mathbf{z})$ by setting the functional derivatives to 0. Substituting them into L_1 and L_2, we obtain the tight ELBO. For real-valued tensor, we can derive that

$$\begin{aligned} L_1^* = {} & \frac{n}{2}\log(\beta) + \frac{1}{2}\log|\mathbf{K}_{BB}| - \frac{1}{2}\log|\mathbf{K}_{BB} + \beta\mathbf{A}_1| - \frac{1}{2}\beta A_2 - \frac{1}{2}\beta A_3 + \frac{\beta}{2}\mathrm{tr}(\mathbf{K}_{BB}^{-1}\mathbf{A}_1) \\ & - \frac{1}{2}\sum\nolimits_{k=1}^{K} \|\mathbf{U}^{(k)}\|_F^2 + \frac{1}{2}\beta^2\mathrm{tr}(\mathbf{a}_4^\top(\mathbf{K}_{BB} + \beta\mathbf{A}_1)^{-1})\mathbf{a}_4) + \mathrm{const}, \end{aligned} \tag{7}$$

where $\mathbf{A}_1 = \sum_j k(\mathbf{B}, \mathbf{x}_{\mathbf{i}_j})k(\mathbf{x}_{\mathbf{i}_j}, \mathbf{B})$, $A_2 = \sum_j y_{\mathbf{i}_j}^2$, $A_3 = \sum_j k(\mathbf{x}_{\mathbf{i}_j}, \mathbf{x}_{\mathbf{i}_j})$, $\mathbf{a}_4 = \sum_j k(\mathbf{B}, \mathbf{x}_{\mathbf{i}_j})y_{\mathbf{i}_j}$, and $\|\cdot\|_F$ is Frobenius norm.

The tight ELBO for binary tensor is more complex, because $q(\mathbf{v})$ and $q(\mathbf{z})$ are coupled in the original ELBO (Equation (6)). To obtain the tight ELBO, we first fix $q(\mathbf{z})$ and follows the same way as in the continuous case to obtain an intermediate bound, $\hat{L}_2(q(\mathbf{z}), \mathcal{U}, \mathbf{B}) = \max_{q(\mathbf{v})} L_2(q(\mathbf{v}), q(\mathbf{z}), \mathcal{U}, \mathbf{B})$. Then we use functional derivatives to calculate the optimal $q(\mathbf{z})$. It turns out that each optimal $q(z_i)$ is a truncated Gaussian:$q^*(z_j) \propto \mathcal{N}(z_j|c_j\langle z_j\rangle, 1)\mathbb{1}\big((2y_{\mathbf{i}_j} - 1)z_j \geq 0\big)$, and $\langle z_j\rangle$ can be obtained by solving the following nonlinear equation: $\langle z_j\rangle = c_j\langle z_j\rangle + \frac{(2y_{\mathbf{i}_j}-1)\mathcal{N}(c_j\langle z_j\rangle;0,1)}{\Phi\big((2y_{\mathbf{i}_j}-1)c_j\langle z_j\rangle\big)}$ where $\langle\cdot\rangle$ denotes the expectation under the variational posterior, $c_j = k(\mathbf{x}_{\mathbf{i}_j}, \mathbf{B})(\mathbf{K}_{BB} + \mathbf{K}_{BS}\mathbf{K}_{SB})^{-1}k(\mathbf{B}, \mathbf{x}_{\mathbf{i}_j})$ and $\Phi(\cdot)$ is the cumulative function for standard Gaussian. The nonlinear equation is guaranteed to have a solution. Substituting the optimal $q(\mathbf{z})$ in $\hat{L}_2$, we can obtain a tight ELBO L_2^* that has a form very similar to L_1^* for the real-valued case.

4.3 Distributed Inference on Tight Bound

Given the tight ELBO, we develop distributed algorithm to optimize the latent factors $\mathcal{U}$ and the inducing points $\mathbf{B}$. First, for continuous data, we can see from (7) that the major calculation is on terms $\mathbf{A}_1$, A_2, A_3 and $\mathbf{a}_4$. Due to their additive form over tensor entries, their gradient with respect to $\mathcal{U}$ and $\mathbf{B}$ (as well as the hyperparameters) also have additive forms over tensor entries. The derivation of the full gradient is long and tedious and we omit it here.We distribute the computation over multiple computational nodes (MAP step) and then collect the results to calculate L_1^* and the gradient ∇L_1^* (REDUCE step). Then a standard optimization framework is used, such as gradient descent and L-BFGS. For binary data, although we do not have an analytical form of L_2^*, the gradient ∇L_2^* can be calculated by first solving the nonlinear equations to obtain $q^*(\mathbf{z})$ and then substituting it into $\hat{L}_2$ and taking gradient as if $q^*(\mathbf{z})$ is constant: $\frac{\partial \hat{L}_2(q^*(\mathbf{z}),\phi)}{\partial \phi}$ where $\phi = \{\mathcal{U}, \mathbf{B}\}$, because $q^*(\mathbf{z})$'s parameters ψ has $\frac{\partial \psi}{\partial \phi} = 0$. Compared with the continuous case, the extra computation is to solve the nonlinear equation for each tensor entry $\mathbf{i}_j$ used in the model. This can also be distributed in MAP step.

4.3.1 Efficient Non-Key-Value Map-Reduce

To avoid expensive data shuffling, we discard the standard key-value Map-Reduce design. Instead, on each MAPPER, a whole gradient vector is maintained for all the parameters, including $\mathcal{U}$, $\mathbf{B}$ and the hyperparameters. Then the gradients are calculated according to the allocated tensor entries and stored in the whole gradient vector. After calculation, each MAPPER sends out the whole gradient vector; the REDUCER simply sums them all to obtain the full gradient; there is no extra data sorting. Efficient MAPREDUCE systems, such as SPARK, can highly optimize non-shuffling MAP and REDUCE, where most of the data are buffered in memory and massive disk I/Os are prevented; however, with data shuffling, the performance degrades severely [2]. Therefore, our algorithm can fully use the memory-cache mechanism to achieve fast inference.

5 Related work

There are many excellent works for tensor factorization, such as [11, 1, 10, 12, 13]. However, they are mainly based on multilinear factorization schemes, like CP, and are difficult to model nonlinear relationships and to discover complex patterns in data. Infinite Tucker decomposition [16], and its distributed extension, DinTucker [17] address this issue by modelling tensors or subtensors via tensor-variate GP. Further in [18], the Dirichlet process mixture prior is combined with the tensor-variate GP to find unknown latent clusters in tensor modes. Despite the nonlinear modelling power, these methods may suffer from the extreme sparsity in real-world tensor data; this stems from the kronekcer product structure in the covariance of tensor-variate GP; it requires to model an entire tensor, no matter whether an element is meaningful or not. By contrast, our flexible GP factorization method removes the Kronecker product restriction and can model any set of tensor entries; it can use a balanced zero and nonzero elements to prevent biased learning; it allows users to select meaningful entries with domain knowledge. In theory, all these nonlinear factorization models are instances of random function prior models [8] in exchangeable multidimensional arrays.

Our distributed variational inference algorithm is based on sparse GP, an efficient approximation framework to scale up GP models. Many works have been proposed for sparse GP. An excellent review of these works are given in [9]. Recently, in [14] a variational learning framework for sparse GP is proposed, based on which a tight variational lower bound for distributed inference of GP regression and GPLVM [7] is developed in [3]. While using similar variational framework, our works aims for tensor factorization; moreover, we obtain the tight ELBO not only for continuous data, but also for binary data; the latter is more challenging, because the tight ELBO does not have an analytical form; we address this problem by solving a nonlinear equation, with solution guaranteed; further, we develop an efficient, accurate method to calculate the ELBO and its gradient; our methods can also be used for distributed GP classification.

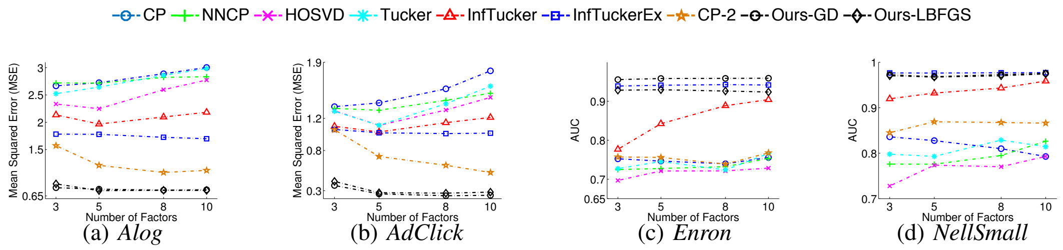

Figure 1: The prediction results on small datasets. The results are averaged over 5 runs.

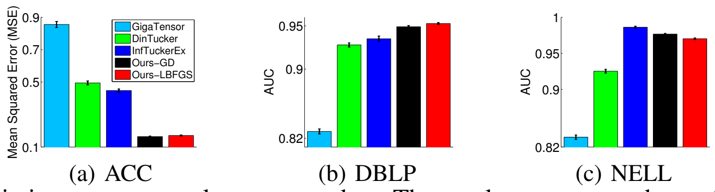

Figure 2: Prediction accuracy on large tensor data. The results are averaged over 50 test datasets.

6 Experiment

6.1 Missing Value Prediction

Datasets. First, four small datasets were used to examine the predictive performance of the proposed model: (1) *Alog*, a real-valued tensor of size $200 \times 100 \times 200$, was extracted from an access log from a file management system. It describes a three-way interaction (user, action, resource) and contains 0.33% nonzero entries.(2) *AdClick*, a real-valued tensor of size $80 \times 100 \times 100$, describing (user, publisher, advertisement) clicks. It was extracted from an online advertisement click log and contains 2.39% nonzero entries. (3) *Enron*, a binary tensor extracted from the Enron email dataset (`www.cs.cmu.edu/~./enron/`) and depicting three-way relationships (sender, receiver, time). It contains $203 \times 203 \times 200$ elements, of which 0.01% are nonzero. (4) *NellSmall*, a binary tensor extracted from the NELL knowledge base (`rtw.ml.cmu.edu/rtw/resources`), of size $295 \times 170 \times 94$. It depicts the knowledge predicts (entity, relationship, entity). The data contains 0.05% nonzero elements.

Competing methods. We compared our approach with the following tensor factorization methods: CP, nonnegative CP (NN-CP) [11], high order SVD (HOSVD) [6], Tucker, infinite Tucker decomposition (InfTucker) [16] and its extension (InfTuckerEx) which uses Dirichlet process mixture (DPM) prior to model latent clusters and local tensor-variate GP to perform scalable, online factorization [18]. Note that InfTucker and InfTuckerEx are nonlinear factorization approaches.

Parameter settings. We used the same test setting in [18]: The number of the latent factors was chosen from $\{3, 5, 8, 10\}$. All the methods were evaluated by a 5-fold cross validation: The nonzero entries were randomly split into 5 folds and 4 folds were used for training; the remaining non-zero entries and 0.1% zero entries were used for test so that the evaluation will not be dominated by the large portion of zero entries. We followed [18] to select parameters for InfTucker and InfTuckerEx except that the kernel form was also selected in another cross-validation; for our model, we consistently used 100 inducing points and a balanced training set: In addition to the nonzero entries, we randomly sampled the same number of zero entries; we made sure they do not overlap with the test zero elements. Our model used ARD kernel and the kernel parameters were estimated jointly with the latent factors. We examined our distributed inference algorithm with two optimization frameworks, gradient descent and L-BFGS (denoted by Ours-GD and Ours-LBFGS respectively). For comprehensive evaluation, we also examined CP on balanced training entries generated in the same way as for our model, denoted by CP-2. The mean squared error (MSE) is used to evaluate predictive performance on *Alog* and *Click* and area-under-curve (AUC) on *Enron* and *Nell*. The average results from the 5-fold cross validation are reported.

Results. As shown on Figure 1, our model achieves higher prediction accuracy than InfTucker, and a better or comparable accuracy than InfTuckerEx. A t-test shows that our model outperforms InfTucker significanlty ($p < 0.05$) in almost all the cases. Although InfTuckerEx uses DPM prior to improve factorization, our model still obtains significantly better prediction on *Alog* and *AdClick* and comparable or better performance on *Enron* and *NellSmall*. This might be attributed to our model's flexibility of using balanced training entries to prevent learning bias (toward massive zeros). Similar improvement can be observed from CP to CP-2. Finally, our model largely outperforms all the remaining methods, showing the advantage of nonlinear factorization.

6.2 Scalability with regard to the Number of Machines

To examine the scalability of the proposed distributed inference algorithm, we used the following real-world large datasets: (1) ACC, A real-valued tensor describing three-way interactions (user, action, resource) in a code repository management system [18]. The tensor is of size $3K \times 150 \times 30K$, where 0.009% are nonzero. (2) DBLP: a binary tensor depicting a three-way bibliography relationship (author, conference, keyword) [18]. The tensor was extracted from DBLP database and contains $10K \times 200 \times 10K$ elements, where 0.001% are nonzero entries. (3) NELL: a binary tensor representing the knowledge predicts, in the form of (entity, entity, relationship) [17]. The tensor size is $20K \times 12.3K \times 280$ and 0.0001% are nonzero.

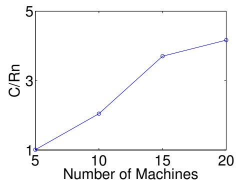

Figure 3: The scalability of our distributed inference algorithm.

The scalability of our distributed inference algorithm was examined with regard to the number of machines on ACC dataset. The number of latent factors was set to 3. We ran our algorithm based on gradient descent. The results are shown in Figure 3. The Y-axis shows the reciprocal of the running time multiplied by a constant—which corresponds to the running speed. As we can see, the speed of our algorithm scales up linearly to the number of machines.

6.3 Large Tensor Analysis

We then compared our approach with three state-of-the-art large scale tensor factorization methods: GigaTensor [5], Distributed infinite Tucker decomposition (DinTucker) [17], and InfTuckerEx. Both GigaTensor and DinTucker are developed on HADOOP, while InfTuckerEx uses online learning. Our model was implemented on SPARK. We ran Gigatensor, DinTucker and our approach on a large YARN cluster and InfTuckerEx on a single computer.

The number of latent factors were set to 3 for ACC and DBLP, and 5 for NELL. We randomly chose 80% of nonzero entries for training, and then sampled 50 test datasets from the remaining entries. For ACC and DBLP, each test dataset comprises 200 nonzero elements and $1,800$ zero elements; for NELL, each test dataset contains 200 nonzero elements and $2,000$ zero elements.

Figure 2 shows the predictive performance. It turns out that our approach consistently outperforms GigaTensor and DinTucker on all the three datasets; our approach outperforms InfTuckerEx on ACC and DBLP and is slightly worse than InfTuckerEx on NELL; note again that InfTuckerEx uses DPM prior to enhance factorization while our model does not; finally, all the nonlinear factorization methods improve GigaTensor, a distributed CP factorization algorithm by a large margin, confirming the advantages of nonlinear factorization on large data. In terms of speed, our algorithm is much faster than GigaTensor and DinTucker. For example, on DBLP dataset, the average per-iteration running time were 1.04, 15.4 and 20.5 minutes for our model, GigaTensor and DinTucker respectively.

6.4 Application on Click-Through-Rate Prediction

Finally, we applied our tensor factorization approach on Click-Through-Rate (CTR) prediction for online advertising. We used the online ads click log from a major Internet company, from which we extracted a four mode tensor (*user*, *advertisement*, *publisher*, *page-section*). We used the first three days's log on May 2015, trained our model on one day's data and used it to predict the click behaviour on the next day. We extracted the same number of non-clicks as the clicks for training and testing. For comparison, we used popular Logistic regression and linear SVM models, where each tensor entry is represented by a set of binary features according to the indices of each entity in the entry. The results are reported in Table 1, in terms of AUC. It can be shown that our model improves Logistic regression and Linear SVM by a large margin, on average 20.7% and 20.4% respectively. Thus it shows a promising potential of applying our model on CTR prediction task.

Table 1: CTR prediction accuracy on the first three days of May 2015. "1-2" means using May 1st's data for training and May 2nd's data for testing; similar are "2-3" and "3-4".

Method	1-2	2-3	3-4
Logistic regression	0.7360	0.7337	0.7538
Linear SVM	0.7414	0.7332	0.7540
Our model	**0.8907**	**0.8897**	**0.9036**

7 Conclusion

We propose a new nonlinear tensor factorization model. By disposing of the Kronecker product covariance structure, the model can exploit data sparsity and is flexible to incorporate meaningful tensor elements for training. Moreover, an efficient distributed variational inference algorithm is developed based on MAPREDUCE framework. The algorithm optimizes a tight ELBO, prevents inefficient EM and expensive data shuffling procedures, and fully exploit the memory-cache mechanism in efficient MAPREDUCE systems, such as SPARK.

References

[1] Evrim Acar, Daniel M Dunlavy, Tamara G Kolda, and Morten Morup. Scalable tensor factorizations for incomplete data. *Chemometrics and Intelligent Laboratory Systems*, 106(1):41–56, 2011.

[2] Aaron Davidson and Andrew Or. Optimizing shuffle performance in spark. *University of California, Berkeley-Department of Electrical Engineering and Computer Sciences, Tech. Rep*, 2013.

[3] Yarin Gal, Mark van der Wilk, and Carl Rasmussen. Distributed variational inference in sparse gaussian process regression and latent variable models. In *Advances in Neural Information Processing Systems*, pages 3257–3265, 2014.

[4] R. A. Harshman. Foundations of the PARAFAC procedure: Model and conditions for an"explanatory"multi-mode factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.

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

[6] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl*, 21:1253–1278, 2000.

[7] Neil D Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. *Advances in neural information processing systems*, 16(3):329–336, 2004.

[8] James Robert Lloyd, Peter Orbanz, Zoubin Ghahramani, and Daniel M. Roy. Random function priors for exchangeable arrays with applications to graphs and relational data. In *Advances in Neural Information Processing Systems 24*, pages 1007–1015, 2012.

[9] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *The Journal of Machine Learning Research*, 6:1939–1959, 2005.

[10] Piyush Rai, Yingjian Wang, Shengbo Guo, Gary Chen, David Dunson, and Lawrence Carin. Scalable Bayesian low-rank decomposition of incomplete multiway tensors. In *Proceedings of the 31th International Conference on Machine Learning (ICML)*, 2014.

[11] Amnon Shashua and Tamir Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *Proceedings of the 22th International Conference on Machine Learning (ICML)*, pages 792–799, 2005.

[12] Wei Sun, Junwei Lu, Han Liu, and Guang Cheng. Provable sparse tensor decomposition. *arXiv preprint arXiv:1502.01425*, 2015.

[13] Wei Sun, Zhaoran Wang, Han Liu, and Guang Cheng. Non-convex statistical optimization for sparse tensor graphical model. In *Advances in Neural Information Processing Systems*, 2015.

[14] Michalis K Titsias. Variational learning of inducing variables in sparse gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 567–574, 2009.

[15] Ledyard Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966.

[16] Zenglin Xu, Feng Yan, and Yuan Qi. Infinite Tucker decomposition: Nonparametric Bayesian models for multiway data analysis. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012.

[17] Shandian Zhe, Yuan Qi, Youngja Park, Ian Molloy, and Suresh Chari. Dintucker: Scaling up gaussian process models on multidimensional arrays with billions of elements. *arXiv preprint arXiv:1311.2663*, 2013.

[18] Shandian Zhe, Zenglin Xu, Xinqi Chu, Yuan Qi, and Youngja Park. Scalable nonparametric multiway data analysis. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 1125–1134, 2015.