Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

DinTucker: Scaling Up Gaussian Process Models on Large Multidimensional Arrays

Shandian Zhe,[1] **Yuan Qi,**[1] **Youngja Park,**[2] **Zenglin Xu,**[3] **Ian Molloy,**[2] and **Suresh Chari**[2]
[1]Department of Computer Science, Purdue University
[2]IBM Thomas J. Watson Research Center
[3]School of Comp. Sci. & Eng., Big Data Res. Center, University of Electronic Science and Technology of China

Abstract

Tensor decomposition methods are effective tools for modelling multidimensional array data (i.e., tensors). Among them, nonparametric Bayesian models, such as Infinite Tucker Decomposition (InfTucker), are more powerful than multilinear factorization approaches, including Tucker and PARAFAC, and usually achieve better predictive performance. However, they are difficult to handle massive data due to a prohibitively high training cost. To address this limitation, we propose **D**istributed **in**finite **Tucker** (DINTUCKER), a new hierarchical Bayesian model that enables local learning of InfTucker on subarrays and global information integration from local results. We further develop a distributed stochastic gradient descent algorithm, coupled with variational inference for model estimation. In addition, the connection between DINTUCKER and InfTucker is revealed in terms of model evidence. Experiments demonstrate that DINTUCKER maintains the predictive accuracy of InfTucker and is scalable on massive data: On multidimensional arrays with billions of elements from two real-world applications, DINTUCKER achieves significantly higher prediction accuracy with less training time, compared with the state-of-the-art large-scale tensor decomposition method, GigaTensor.

Introduction

Real-world datasets with multiple aspects are usually described by multidimensional arrays, i.e., tensors. For example, a file access log can be depicted by a three-mode array (*user, file, action*) and patient-medicine responses by a four mode array (*person, medicine, biomarker, time*). Given tensor-valued data, tensor analysis aims to capture complex interactions embedded in data and to predict missing entries (*e.g.,* unknown medicine responses in patient-medicine responses data).

Despite the success of traditional tensor analysis methods, such as Tucker (Tucker 1966), CANDECOMP/PARAFAC (CP) (Harshman 1970) and their generalizations (Chu and Ghahramani 2009), they are mostly multilinear methods and are difficult to handle complex interactions among tensor modes. Therefore, nonlinear tensor decomposition methods based on nonparametric Bayesian, such as InfTucker (Xu, Yan, and Qi 2012; 2015) and its generalization, random function prior models (Lloyd et al. 2012), have been proposed. Theoretically justified by the generalization of de Finetti's theorem for RCE arrays (Aldous 1981; Lauritzen 2006), these models are able to capture the nonlinear relationship between array elements and often leads to a superior predictive performance. However, a critical bottleneck of InfTucker and its generalization is that they mainly operate on data that can fit in the memory of a singe computer and cannot employ the parallel computing power from computer clusters or graphics processing units (GPUs). This largely stems from a global Gaussian process (GP) assumption, which treats the vectorized whole array as one GP projection, and makes them infeasible for real-world large data.

To address this issue, we propose **D**istributed **in**finite **Tucker** (DINTUCKER) decomposition, a new nonlinear tensor decomposition model based on local GP assumption. It splits the whole array into multiple small subarrays, each of which is sampled from a local tensor-variate GP. By conducting infinite Tucker decomposition on the subarrays in a distributed manner, DINTUCKER is scalable to massive multidimensional array data, while keeps the nonlinear modelling power. The major contributions of this work are summarized as follows:

1. Model and Algorithm. A new hierarchical Bayesian model for tensor analysis is proposed, which enables distributed training of InfTucker on subarrays and information integration from all local training results. For fast inference, a distributed variational inference algorithm based on stochastic gradient descent is developed.
2. Scalability. The local training scheme enables DINTUCKER scale to array data with more than 50 billion elements (as demonstrated in Table 1). Furthermore, DINTUCKER enjoys almost linear scalability on the number of computational nodes.
3. Analysis. We show a close connection between DINTUCKER and InfTucker: Under certain conditions, the model evidence of DINTUCKER associates with an evidence lower bound of InfTucker. The conclusions can be generalized to GP and local GP models.
4. Applications. DINTUCKER has been applied to analyze knowledge bases (Carlson et al. 2010) and user access log from an Internet company. On both cases, DINTUCKER achieves significantly higher prediction accuracy with less training time compared with GigaTensor, the state-of-the-

art large-scale tensor decomposition algorithm.

Background

Tensor Decomposition

We denote a K-mode multidimensional array or tensor by $\mathcal{M} \in \mathbb{R}^{m_1 \times m_2 \ldots \times m_K}$, where the k-th mode has m_k dimensions. We use $m_{\mathbf{i}}$ ($\mathbf{i} = (i_1, \ldots, i_K)$) to denote $\mathcal{M}$'s entry at location $\mathbf{i}$. Using the vectorization operation, we can stack all of $\mathcal{M}$'s entries in a vector, $\text{vec}(\mathcal{M})$, with size $\prod_{k=1}^{K} m_k$ by 1. In $\text{vec}(\mathcal{M})$, the entry $\mathbf{i} = (i_1, \ldots, i_K)$ of $\mathcal{M}$ is mapped to the entry at position $j = i_K + \sum_{i=1}^{K-1}(i_k - 1)\prod_{k+1}^{K} m_k$. 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 multiplication 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 entry-wise definition is $(\mathcal{W} \times_k \mathbf{U})_{i_1 \ldots i_{k-1} j i_{k+1} \ldots i_K} = \sum_{i_k=1}^{r_k} w_{i_1 \ldots i_K} u_{j i_k}$. The Tucker decomposition of a K-mode tensor $\mathcal{M}$ is $\mathcal{M} = \mathcal{W} \times_1 \mathbf{U}^{(1)} \times_2 \ldots \times_K \mathbf{U}^{(K)} = [\![\mathcal{W}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(K)}]\!]$ 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. If we enforce $r_1 = \ldots = r_K$ and restrict the core tensor $\mathcal{W}$ to be diagonal (i.e., $W_{i_1 \ldots i_K} \neq 0$ only if $i_1 = \ldots = i_K$), it reduces to PARAFAC decomposition.

Infinite Tucker Decomposition

The infinite Tucker (InfTucker) decomposition (Xu, Yan, and Qi 2012) generalizes the Tucker decomposition in an infinite feature space based on a tensor-variate GP. The tensor-variate GP is a collection of random variables $\{m(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(K)})\}$, $\mathbf{u}^{(k)} \in \mathbb{R}^r$, whose finite joint probability over any set of input locations follows the tensor-variate normal distribution. Specifically, given $\mathcal{U} = \{\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(k)}\}$, the zero mean tensor-variate GP on $\mathcal{M}$ has the probability density function

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

where $\Sigma^{(k)} = k(\mathbf{U}^{(k)}, \mathbf{U}^{(k)})$ is the covariance matrix in mode k, and $\otimes$ is the Kronecker product.

The InfTucker model assumes the latent factors $\mathcal{U}$ are sampled from element-wise Laplace priors $p(\mathcal{U})$, which encourage sparse estimation for easy model interpretation. Given $\mathcal{U}$, a latent real-valued tensor $\mathcal{M}$ is sampled from the tensor variate GP. Then, 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 for continuous observations and probit models for binary observations. Thus the joint distribution is $p(\mathcal{Y}, \mathcal{M}, \mathcal{U}) = p(\mathcal{U})p(\mathcal{M}|\mathcal{U})p(\mathcal{Y}|\mathcal{M})$. By using nonlinear covariance function $k(\mathbf{u}^i, \mathbf{u}^i)$, InfTucker maps the latent factors in each mode into an infinite feature space and then performs the Tucker decomposition with the core tensor $\mathcal{W}$ of infinite size. Based on a feature mapping, InfTucker can capture nonlinear relationships between latent factors.

Hierachical Bayesian model for DinTucker

A major bottleneck of InfTucker is that it cannot scale up to large arrays. This stems from a global GP assumption used by InfTucker: It assumes all elements of the tensor $\mathcal{M}$ are sampled from a global Gaussian process given latent factors $\mathcal{U}$. As a result, computing the distribution for the global $\mathcal{M}$—$p(\mathcal{M}|\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(K)})$ in Equation (1)—requires computing the Kronecker-product of the covariance matrices and its inverse. This matrix inversion is prohibitively expensive. Although InfTucker (Xu, Yan, and Qi 2012) explores properties of the Kronecker product to avoid naive computation, it still needs to perform eigen-decomposition over the covariance matrix for each mode, which is infeasible for a large dimension m_k. Moreover, all the latent factors are coupled in $p(\mathcal{M}|\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(K)})$ so that we cannot distribute the computation over many computational units.

To overcome these limitations, we propose DINTUCKER that assumes the data are sampled from many, smaller GP models, and the latent variables for these GP models are coupled together in a hierarchical Bayesian model. The local GP enables fast computation over subarrays and the hierarchical Bayesian model allows information sharing across different subarrays—making distributed inference possible.

Specifically, we first break the observed multidimensional array $\mathcal{Y}$ into N subarrays $\{\mathcal{Y}_1, \ldots, \mathcal{Y}_N\}$ for multiple computational units (*e.g.,* one per MAPPER in MAPREDUCE). Each subarray is sampled from a GP based on latent factors $\tilde{\mathcal{U}}_n = \{\tilde{\mathbf{U}}_n^{(1)}, \ldots, \tilde{\mathbf{U}}_n^{(K)}\}$. Then we tie these latent factors to the common latent factors $\mathcal{U} = \{\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(K)}\}$ via a prior distribution $p(\tilde{\mathcal{U}}_n|\mathcal{U}) = \prod_{k=1}^{K} \mathcal{N}(\text{vec}(\tilde{\mathbf{U}}_n^{(k)})|\text{vec}(\mathbf{U}^{(k)}), \lambda\mathbf{I})$ where λ is a variance parameter that controls the similarity between $\mathcal{U}$ and $\tilde{\mathcal{U}}_n$.

Furthermore, we use stochastic gradient descent (SGD) to optimize $\{\tilde{\mathcal{U}}_n\}$ and $\mathcal{U}$ due to its computational efficiency and theoretical guarantees. The use of SGD naturally enables us to deal with dynamic array data with increasing sizes over time. To use SGD, we break each $\mathcal{Y}_n$ into T_n smaller subarrays $\mathcal{Y}_n = \{\mathcal{Y}_{n1}, \ldots, \mathcal{Y}_{nT_n}\}$. We allow the subarrays from each $\mathcal{Y}_n$ to share the same latent factors $\{\tilde{\mathcal{U}}_n\}$. The reason that we do not need to explicitly introduce another set of latent factors, say, $\{\tilde{\mathcal{U}}_{nt}\}_t$, for subarrays in each $\mathcal{Y}_n$ is the following: suppose we have a prior $p(\tilde{\mathcal{U}}_{nt}|\tilde{\mathcal{U}}_n)$ to couple these $\tilde{\mathcal{U}}_{nt}$, we can set $p(\tilde{\mathcal{U}}_{nt}|\tilde{\mathcal{U}}_n) = \delta(\tilde{\mathcal{U}}_{nt} - \tilde{\mathcal{U}}_n)$ ($\delta(a) = 1$ if and only if $a = 0$) without causing conflicts between updates over $\tilde{\mathcal{U}}_{nt}$—since they are updated sequentially. This situation is different from parallel inference over $\tilde{\mathcal{U}}_n$ for which, if we simply set $\tilde{\mathcal{U}}_n = \mathcal{U}$ for all n, we will have conflicts between inconsistent $\tilde{\mathcal{U}}_n$ estimated in parallel from different computational units.

Given $\tilde{\mathcal{U}}_n$, a latent real-valued subarray $\mathcal{M}_{nt}$ is sampled from the corresponding local GP. Then we sample the noisy observations $\mathcal{Y}_{nt}$ from the latent subarray $\mathcal{M}_{nt}$. Denoting $\{\mathcal{M}_{nt}\}_{t=1}^{T_n}$ by $\mathcal{M}_n$, we have the joint probability of our model: $p(\mathcal{U}, \{\tilde{\mathcal{U}}_n, \mathcal{M}_n, \mathcal{Y}_n\}_{n=1}^{N}) = \prod_{n=1}^{N} p(\tilde{\mathcal{U}}_n|\mathcal{U}) \prod_{t=1}^{T_n} p(\mathcal{M}_{nt}|\tilde{\mathcal{U}}_n)p(\mathcal{Y}_{nt}|\mathcal{M}_{nt})$. Note that $\mathcal{M}_{nt}$ depends only on its corresponding elements in $\tilde{\mathcal{U}}_n$, so that the computation of $p(\mathcal{M}_{nt}|\tilde{\mathcal{U}}_n)$ is efficient.

Compared with InfTucker, the joint probability of DIN-

TUCKER replaces the global term $p(\mathcal{M}|\mathbf{U}^{(1)},\ldots,\mathbf{U}^{(K)})$ (which couples all the latent factors and the whole latent multidimensional array $\mathcal{M}$) by smaller local terms. These local terms require much less memory and processing time than the global term. More important, the additive nature of these local terms in the log domain enables distributed inference.

Distributed inference on MAPREDUCE

Now we present our distributed inference algorithm in MAPREDUCE framework. We focus on binary tensor data, for which we use the probit model for $p(\mathcal{Y}_{nt}|\mathcal{M}_{nt})$. It is straightforward to modify the following presentation to handle continuous and count data.

First, we use data augmentation to decompose the probit model into $p(y_\mathbf{i}|m_\mathbf{i}) = \int p(y_\mathbf{i}|z_\mathbf{i})p(z_\mathbf{i}|m_\mathbf{i})dz_\mathbf{i}$, where $p(y_\mathbf{i}|z_\mathbf{i}) = \delta(y_\mathbf{i}=1)\delta(z_\mathbf{i}>0) + \delta(y_\mathbf{i}=0)\delta(z_\mathbf{i}\leq 0)$ and $p(z_\mathbf{i}|m_\mathbf{i}) = \mathcal{N}(z_\mathbf{i}|m_\mathbf{i},1)$. Here $\delta(\cdot)$ is the binary indicator function. For each $\mathcal{M}_{nt}\in\mathcal{M}_n$, we introduce an augmented $\mathcal{Z}_{nt}$. Let us denote $\mathcal{Z}_n = \{\mathcal{Z}_{nt}\}_{t=1}^{T_n}$. The joint probability of the augmented model is

$$p(\mathcal{U},\{\tilde{\mathcal{U}}_n,\mathcal{M}_n,\mathcal{Z}_n,\mathcal{Y}_n\}_{n=1}^N) = \prod_{n=1}^N p(\tilde{\mathcal{U}}_n|\mathcal{U})\prod_{t=1}^{T_n} p(\mathcal{M}_{nt}|\tilde{\mathcal{U}}_n)p(\mathcal{Z}_{nt}|\mathcal{M}_{nt})p(\mathcal{Y}_{nt}|\mathcal{Z}_{nt}).$$

Variational approximation

We then apply variational EM to optimize the latent factors $\mathcal{U},\{\tilde{\mathcal{U}}_n\}$: in the E-step, we use the variational approximation and, in the M-step, we apply SGD to maximize the variational lower bound over the latent factors. Specifically, in the E-step, we use a fully factorized distribution $q(\{\mathcal{Z}_n,\mathcal{M}_n\}_{n=1}^N) = \prod_{n=1}^N\prod_{t=1}^{T_n} q(\mathcal{Z}_{nt})q(\mathcal{M}_{nt})$ to approximate the posterior distribution $p(\{\mathcal{Z}_n,\mathcal{M}_n\}_{n=1}^N|\{\mathcal{Y}_n,\tilde{\mathcal{U}}_n\}_{n=1}^N,\mathcal{U})$. The variational inference minimizes the Kullback-Leibler (KL) divergence between the approximate and the exact posteriors by coordinate descent. The variational updates for $q(\mathcal{Z}_{nt})$ and $q(\mathcal{M}_{nt})$ are the same as those for $q(\mathcal{Z})$ and $q(\mathcal{M})$ in (Xu, Yan, and Qi 2012).

Estimating latent factors

Given the variational distributions, we estimate the group-specific latent factors $\{\tilde{\mathcal{U}}_n\}_{n=1}^N$ and the common latent factors $\mathcal{U}$ by maximizing the expected log joint probability, $\mathbb{E}_q\left[\log p(\mathcal{U},\{\tilde{\mathcal{U}}_n,\mathcal{Y}_n,\mathcal{Z}_n,\mathcal{M}_n\}_{n=1}^N)\right]$. Specifically, we optimize the group-specific latent factors $\{\tilde{\mathcal{U}}_n\}_{n=1}^N$ via SGD in the MAP step and update the common latent factors $\mathcal{U}$ in the REDUCE step.

Estimating the group-specific latent factors $\{\tilde{\mathcal{U}}_n\}$ via MAPPER Given $\mathcal{U}$, the expected log likelihood function with respect to $\tilde{\mathcal{U}}_n$ is

$$f(\tilde{\mathcal{U}}_n) = \log(p(\tilde{\mathcal{U}}_n|\mathcal{U})) + \sum_{t=1}^{T_n}\mathbb{E}_q\left[\log(p(\mathcal{M}_{nt}|\tilde{\mathcal{U}}_n))\right] + \sum_{t=1}^{T_n}\mathbb{E}_q\left[\log(p(\mathcal{Z}_{nt}|\mathcal{M}_{nt}))\right]. \quad (2)$$

We have investigated L-BFGS to maximize Equation (2) over $\tilde{\mathbf{U}}_n^k$ but SGD turns out to give better performance. To perform SGD, we first rearrange the objective function in Equation (2) as a summation form $f_n(\tilde{\mathcal{U}}_n) = \sum_{t=1}^{T_n} g_{nt}(\tilde{\mathcal{U}}_n)$, and

$$\begin{aligned} g_{nt}(\tilde{\mathcal{U}}_n) &= \frac{1}{T_n}\log(p(\tilde{\mathcal{U}}_n|\mathcal{U})) + \mathbb{E}_q\left[\log(p(\mathcal{M}_{nt}|\tilde{\mathcal{U}}_n))\right] \\ &\quad + \mathbb{E}_q\left[\log(p(\mathcal{Z}_{nt}|\mathcal{M}_{nt}))\right] \\ &= -\frac{1}{2T_n\lambda}\sum_{j=1}^K \|\mathrm{vec}(\mathbf{U}^{(j)}) - \mathrm{vec}(\tilde{\mathbf{U}}_n^{(j)})\|^2 \\ &\quad + \|[\![\mathbb{E}_q[\mathcal{M}_{nt}]; (\mathbf{\Sigma}_{nt}^{(1)})^{-\frac{1}{2}},\ldots,(\mathbf{\Sigma}_{nt}^{(K)})^{-\frac{1}{2}}]\!]\|^2 \\ &\quad + \sum_{k=1}^K \frac{m_{nt}}{m_{nt,k}}\log|\mathbf{\Sigma}_{nt}^{(k)}| + \mathrm{tr}\left({\mathbf{\Lambda}_{nt}}^{-1}\mathbf{\Upsilon}_{nt}\right) \end{aligned} \quad (3)$$

where $m_{nt,k}$ is the dimension of k-th mode in $\mathcal{Y}_{nt}$, $m_{nt} = \prod_{k=1}^K m_{nt,k}$, $\mathbf{\Lambda}_{nt} = \mathbf{\Sigma}_{nt}^{(1)}\otimes\ldots\otimes\mathbf{\Sigma}_{nt}^{(K)}$, $\mathbf{\Sigma}_{nt}^{(k)} = k(\tilde{\mathbf{U}}_{nt}^{(k)},\tilde{\mathbf{U}}_{nt}^{(k)})$ is the k-th mode covariance matrix over the sub-factors of $\tilde{\mathcal{U}}_n$, and $\mathbf{\Upsilon}_{nt}$ is the statistics computed in the variational E-step.

We randomly shuffle the subarrays in $\mathcal{Y}_n$ and sequentially process each subarray. For each subarray $\mathcal{Y}_{nt}$, we have the update: $\tilde{\mathcal{U}}_n = \tilde{\mathcal{U}}_n + \eta\partial g_{nt}(\tilde{\mathcal{U}}_n)$. The gradient $\partial g_{nt}(\tilde{\mathcal{U}}_n)$ has a form similar to that of the expected log joint probability with respect to global latent factors $\mathcal{U}$ in InfTucker. We omit the detailed equation here and refer the detail to the paper of InfTucker (Xu, Yan, and Qi 2012). The SGD optimization for each $\tilde{\mathcal{U}}_n$ is implemented by a MAP task in the MAPREDUCE system.

Estimating the parent latent factors $\mathcal{U}$ via REDUCER Given $\{\tilde{\mathcal{U}}_1,\ldots,\tilde{\mathcal{U}}_N\}$, the expected log joint probability as a function of $\mathcal{U}$ is $f(\mathcal{U}) = \sum_{n=1}^N\sum_{k=1}^K \log\mathcal{N}(\tilde{\mathbf{U}}_n^{(k)}|\mathbf{U}^{(k)},\lambda\mathbf{I})$. Setting this gradient to zero, we have the simple update for $\mathcal{U}$: $\mathbf{U}^{(k)} = \frac{1}{N}\tilde{\mathbf{U}}_n^{(k)}$. This is implemented in the REDUCE step of MAPREDUCE.

Algorithm complexity

The time complexity of InfTucker is $O(\sum_{k=1}^K m_k^3 + m_k m)$ where m_k is the dimension of mode k and $m = \prod_{k=1}^K m_k$. If any m_k is large, InfTucker is computationally too expensive to be practical. For DINTUCKER, if the dimension of a subarray in mode k is $\overline{m}_k$, the time complexity of analyzing it is $O(\sum_{k=1}^K \overline{m}_k^3 + \overline{m}_k\overline{m})$ where $\overline{m} = \prod_{k=1}^K\overline{m}_k$ is the total number of entries in a subarray. When we set identical $\overline{m}_k$ for any k, the time complexity becomes $O(\overline{m}^{(1+\frac{1}{K})})$. Given

L subarrays and N MAPPER nodes, the time complexity for each MAPPER node is $O(\frac{L}{N}\overline{m}^{(1+\frac{1}{K})})$, nearly linear in the number of elements in each small subarray.

The space complexity of InfTucker is $O(m+\sum_{k=1}^{K} m_k^2 + m_k r_k)$ because it needs to store the whole array and the covariance matrices for all modes in the memory of a computer, in addition to latent factors. This is obviously infeasible for large data. By contrast, DINTUCKER only needs to store one small subarray and its covariance matrices in each MAPPER node via streaming, and the space complexity is $O(\overline{m} + \sum_{k=1}^{K} \overline{m}_k^2 + m_k r_k)$.

Prediction via bagging of local GPs

To predict unknown entries, InfTucker needs to compute the posterior of the whole latent array, which is infeasible for large arrays. To address this issue, we use a collection local GP models on subarrays to predict missing entries and aggregate the predictions by bagging. Specifically, we first sample subarrays, find their corresponding latent factors, and then use them to calculate predictive means of the unknown elements with GP prediction, and finally aggregate the predictive means by averaging. Because DINTUCKER can quickly provide predictions on the small subarrays, it achieves fast final predictions. Note that Bagging (Hastie, Tibshirani, and Friedman 2001) has been widely used to improve many machine learning methods and recently has also shown effective on GP models (Bratieres et al. 2014).

Connection to InfTucker

We now investigate the connection between DINTUCKER and InfTucker in terms of model evidence.

First, for simplicity, we consider real-valued arrays and use a Gaussian noise model for $p(\mathcal{Y}|\mathcal{M})$ in InfTucker and DINTUCKER, i.e., $p(\mathcal{Y}|\mathcal{M}) = \mathcal{N}(\mathrm{vec}(\mathcal{Y})|\mathrm{vec}(\mathcal{M}), \tau\mathbf{I})$; according to (1), the marginal probability of InfTucker is given by $p_I(\mathcal{Y}|\mathcal{U}) = \mathcal{N}\big(\mathrm{vec}(\mathcal{Y})|\mathbf{0}, \mathbf{\Sigma}(\mathcal{U},\mathcal{U})\big)$, where $\mathbf{\Sigma}(\mathcal{U},\mathcal{U}) = k(\mathbf{U}^{(1)},\mathbf{U}^{(1)}) \otimes \ldots \otimes k(\mathbf{U}^{(K)},\mathbf{U}^{(K)}) + \tau\mathbf{I}$. Now we partition the indices of each mode into multiple segments with the same size. Each segment corresponds to a set of latent factors (in that mode). Using the segments from different modes, we form a grid partition of the whole array, $\mathcal{Y} = \{\mathcal{Y}_1, \ldots, \mathcal{Y}_n\}$, where each subarray contains d elements. Let us index i-th subarray by $(i_{s_1}, \ldots, i_{s_K})$ where i_{s_k} represents the segment in mode k. The corresponding latent factors are denoted by $\{\mathbf{U}^{(1)}_{i_{s_1}}, \ldots, \mathbf{U}^{(K)}_{i_{s_K}}\}$. Then we can obtain a block form for the covariance of InfTucker: $\mathbf{\Sigma}(\mathcal{U},\mathcal{U}) = \{\mathbf{\Sigma}_{ij}\}_{1\le i,j\le n}$, where $\mathbf{\Sigma}_{ij} = k(\mathbf{U}^{(1)}_{i_{s_1}}, \mathbf{U}^{(1)}_{j_{s_1}}) \otimes \ldots \otimes k(\mathbf{U}^{(K)}_{i_{s_K}}, \mathbf{U}^{(K)}_{j_{s_K}}) + \delta(i-j)\tau\mathbf{I}$ and, it describes the similarity between the elements of subarrays i and j.

Next, we derive a variational evidence lower bound for InfTucker. Specifically, we first decompose $\mathbf{\Sigma}(\mathcal{U},\mathcal{U})$ into the summation of two block matrices using the following theorem.

Theorem 1. Given a block matrix $\mathbf{\Sigma} = \{\mathbf{\Sigma}_{ij}\} \succ 0$, there exists $\alpha \in [0,1]$, such that $\mathbf{\Sigma} = \mathbf{\Sigma}^{(1)} + \mathbf{\Sigma}^{(2)}$ where $\mathbf{\Sigma}^{(1)}_{ii} = (1-\alpha)\mathbf{\Sigma}_{ii}$ and $\mathbf{\Sigma}^{(1)}_{ij} = \mathbf{\Sigma}_{ij} (i \neq j)$, $\mathbf{\Sigma}^{(2)}$ is block diagonal: $\mathbf{\Sigma}^{(2)} = \mathrm{diag}(\{\alpha\mathbf{\Sigma}_{ii}\})$, and $\mathbf{\Sigma}^{(1)} \succ 0, \mathbf{\Sigma}^{(2)} \succ 0$.

Based on the decomposition, we can construct an equivalent augmented model for InfTucker, by introducing a latent variable $\mathbf{f}$:

$$\mathbf{f} \sim \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{\Sigma}^{(1)}), \quad \mathcal{Y}|\mathbf{f} \sim \mathcal{N}(\mathrm{vec}(\mathcal{Y})|\mathbf{f}, \mathbf{\Sigma}^{(2)}). \tag{4}$$

The log marginal likelihood (i.e., log evidence) of InfTucker can be written as $\log(p_I(\mathcal{Y}|\mathcal{U})) = \log\big(\int \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{\Sigma}^{(1)})\mathcal{N}(\mathrm{vec}(\mathcal{Y})|\mathbf{f}, \mathbf{\Sigma}^{(2)})\mathrm{d}\mathbf{f}\big)$.

Using Jensen's inequality, we have $\log(p_I(\mathcal{Y}|\mathcal{U})) \ge \sum_{i=1}^{n} \int \log\big(\mathcal{N}(\mathrm{vec}(\mathcal{Y}_i)|\mathbf{f}_i, \alpha\mathbf{\Sigma}_{ii})\big)\mathcal{N}(\mathbf{f}_i|0, (1-\alpha)\mathbf{\Sigma}_{ii})\mathrm{d}\mathbf{f}$. Note that we use the property that $\mathbf{\Sigma}^{(2)}$ is block diagonal. The resulted bound is a variational lower bound and the equality is achieved by setting the partition number n to 1 and α to 0. When $\alpha > 0$, we obtain $\log(p_I(\mathcal{Y}|\mathcal{U})) \ge L(\alpha)$ where

$$L(\alpha) = \sum_{i=1}^{n} -\frac{1}{2}\log|2\pi\mathbf{\Sigma}_{ii}| - \frac{1}{2\alpha}\mathrm{vec}(\mathcal{Y}_i)^\top \mathbf{\Sigma}_{ii}^{-1}\mathrm{vec}(\mathcal{Y}_i) - \frac{nd}{2}\log(\alpha) - \frac{nd}{2\alpha}(1-\alpha). \tag{5}$$

Now we consider DINTUCKER built on subarrays $\{\mathcal{Y}_1, \ldots, \mathcal{Y}_n\}$. For simplicity, we set $\lambda = 0$ in the prior of group-specific latent factors so that $p(\tilde{\mathcal{U}}_n|\mathcal{U}) = \delta(\mathcal{U} - \tilde{\mathcal{U}}_n)$. Then the log marginal likelihood of DINTUCKER becomes

$$\log(p_D(\mathcal{Y}|\mathcal{U})) = \sum_{i=1}^{m} \log(\mathcal{N}(\mathrm{vec}(\mathcal{Y}_i)|\mathbf{0}, \mathbf{\Sigma}_{ii})) = \sum_{i=1}^{n} -\frac{1}{2}\log|2\pi\mathbf{\Sigma}_{ii}| - \frac{1}{2}\mathrm{vec}(\mathcal{Y}_i)^\top \mathbf{\Sigma}_{ii}^{-1}\mathrm{vec}(\mathcal{Y}_i). \tag{6}$$

Comparing (5) and (6), we obtain the following conclusions.

Conclusion 1. The log evidence of DINTUCKER is identical to $L(\alpha)$ in (5) when $\alpha = 1$. However, $L(1)$ is not guaranteed to be the evidence lower bound of InfTucker, because $\mathbf{\Sigma}^{(1)}$ can be indefinite and the augmented model (4) can be invalid when $\alpha = 1$.

Theorem 2. When $\tau \ge \frac{1}{2\pi}$, DINTUCKER's evidence associates with an evidence lower bound of InfTucker. Specifically, there exits $\alpha \in (0,1)$ such that

$$\log(p_I(\mathcal{Y}|\mathcal{U})) \ge L(\alpha) \ge \frac{1}{\alpha}\log(p_D(\mathcal{Y}|\mathcal{U})) - \frac{nd}{2}\log(\alpha) - \frac{nd}{2\alpha}(1-\alpha).$$

Note that the above analysis is general for GP models: If we partition GP input $\mathbf{X}$ into $\{\mathbf{X}_1, \ldots, \mathbf{X}_n\}$, responses $\mathbf{y}$ into $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ and use local GP to model each $\mathbf{y}_i$ as an independent GP projection on $\mathbf{X}_i$, we obtain the same conclusions and theorems. Thereby, our analysis reveals the connection not only between InfTucker and DINTUCKER, but also between general GP and local GP models.

Related work

Many works have been proposed for multidimensional array decomposition, such as (Shashua and Hazan 2005a; Chu and Ghahramani 2009; Sutskever, Tenenbaum, and Salakhutdinov 2009; Acar et al. 2011; Hoff 2011; Yang and Dunson 2013; Rai et al. 2014; Sun et al. 2015; Hu, Rai, and Carin 2015; Rai et al. 2015). The majority of them are based on multilinear factorization schemes. Despite their success, they lack of flexibility to capture nonlinear interactions and complex patterns in data. To overcome this limit, nonlinear tensor decomposition models are developed, including InfTucker (Xu, Yan, and Qi 2012) based on global GP and (Zhe et al. 2015) based on Dirichlet process and local GP. However, InfTucker is severally restricted by scalability and impractical for real-world applications; (Zhe et al. 2015) describes an online VB-EM algorithm to decompose large arrays in a single computer, but it cannot use multiple computational resources in parallel and may take a long time for very large array analysis. By contrast, DINTUCKER is built with divide-and-conquer strategy and its hierarchical modelling nature enables efficient parallel inference and soft information sharing from different local GPs on subarrays. As a distributed extension of InfTucker, DINTUCKER maintains the modelling power of InfTucker and is more practical for real-world large array analysis.

The learning of local GPs is not new, for example, (Rasmussen and Ghahramani 2002) proposed an infinite GP mixture model; (Kim, Mallick, and Holmes 2005) used partitions of GP to analyze spatial data; (Gramacy and Lee 2008) proposed treed GP; and (Dunson and Fox 2012) proposed multiresolution GP coupling nestedly partitioned GPs for time-series analysis. However, all these works are based on GP models with known inputs, while in our model, the inputs (i.e., latent factors) are unknown and need to be estimated.

Distributed algorithms to scale up tensor decomposition to massive data becomes a recent research focus, such as GigaTensor (Kang et al. 2012) on MAPREDUCE framework, which exploits data sparseness and avoids the intermediate data explosion, and DFacTo (Choi and Vishwanathan 2014), which exploits the properties of the Khatri-Rao product to reduce the number of sparse matrix-vector products. These algorithms are very efficient, however, they mainly focus on improving the alternative least square algorithm (ALS) for PARAFAC (DFacTo also suits gradient descent algorithm for PARAFAC). It is not clear that how their ideas can be applied to the learning of nonlinear tensor decomposition models.

Experiment

We carried out our experiments on a HADOOP cluster. The cluster consists of 16 machines, each of which has a 4-quad Intel Xeon-E3 3.3 GHz CPU, 8 GB RAM, and a 4 Terabyes disk. We implemented DINTUCKER with PYTHON and used HADOOP streaming for training and prediction.

Small datasets

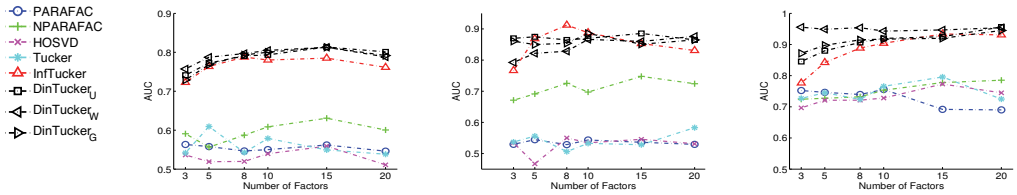
We first examined DINTUCKER on the following social network datasets, *Digg1*, *Digg2* and *Enron*. Both *Digg1* and *Digg2* datasets are extracted from a social news website digg.com. *Digg1* describes a three-way interaction (news, keyword ,topic), and *Digg2* a four-way interaction (user, news, keyword, topic). *Digg1* contains $581 \times 124 \times 48$ elements and 0.024% of them are non-zero. *Digg2* has $22 \times 109 \times 330 \times 30$ elements and 0.002% of them are nonzero. *Enron* is extracted from the Enron email dataset. It depicts a three-way relationship (sender, receiver, time). The dataset contains $203 \times 203 \times 200$ entries, of which 0.01% are nonzero.

We compared DINTUCKER with the following tensor decomposition methods: PARAFAC, nonnegative PARAFAC (NPARAFAC)(Shashua and Hazan 2005b), high order SVD (HOSVD) (Lathauwer, Moor, and Vandewalle 2000), Tucker decomposition and InfTucker. We chose the number of latent factors from the range $\{3,5,8,10,15,20\}$. Since the data are binary, we evaluated all the approaches by the area-under-curve (AUC) based on a random 5-fold partition of the data. Specifically, we split the nonzero entries into 5 folds and used 4 folds for training. For the test set, we used all the ones in the remaining fold and randomly chose 0.1% zero entries (so that the evaluations will not be overwhelmed by zero elements). We repeated this procedure for 5 times with different training and test sets each time. For InfTucker, we used cross validation to tune the hyperparameter of its Laplace prior. For DINTUCKER, we set the subarray size to $40 \times 40 \times 40$ for *Digg1* and *Enron*, and $20 \times 20 \times 20 \times 20$ for *Digg2*. To generate subarrays, we used the three sampling strategies in (Zhe et al. 2015), namely uniform, weighted and grid sampling. For each strategy, we sampled $1,500$ subarrays for training. We ran our distributed online inference algorithm with 3 mappers, and set the number of iterations to 5.

We tuned the learning rate η from the range $\{0.0005, 0.001, 0.002, 0.005, 0.01\}$. We used another cross-validation to choose the kernel function from the RBF, linear, Polynomial and Matérn functions and tuned its hyperparameters. For the Matérn kernel, the order of its Bessel function is either $\frac{3}{2}$ or $\frac{5}{2}$. For our bagging prediction, we randomly sampled 10 subarrays, each with the same size as the training subarrays. The results are shown in Figure 1. As we can see, all versions of DINTUCKER are comparable to InfTucker on the three datasets. Furthermore, DINTUCKER significantly outperforms all the alternatives.

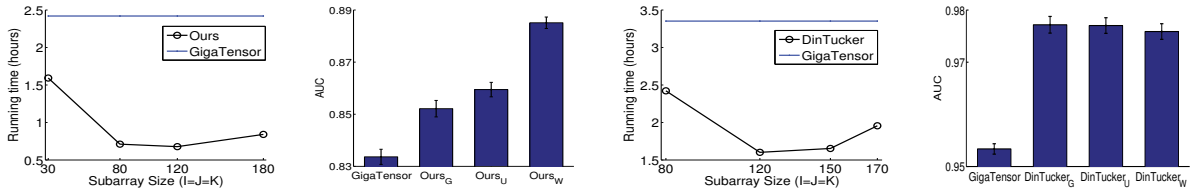
Scalability with regard to the number of machines

To examine the scalability and predictive performance of DINTUCKER, we used the large datasets in two real-world applications: The first dataset is NELL, a knowledge base containing triples (*e.g.,*'George Harrison', 'playsInstrument', 'Guitar') from the 'Read the Web' project (Carlson et al. 2010). We filtered out the triples with confidence less than 0.99 and then analyzed the triplets from 20,000 most frequent entities. The second dataset is ACC, an access log from a source code version control system in a large company. The log provides information such as user id,



(a) *digg1* (b) *digg2* (c) *enron*

Figure 1: The prediction results on small datasets. The results are averaged over 5 runs. DINTUCKER$_{\mathrm{U}}$, DINTUCKER$_{\mathrm{W}}$ and DINTUCKER$_{\mathrm{G}}$ refer to our method based on the uniform, weighted, and grid sampling strategies, respectively.



(a) NELL: running time (b) NELL: prediction (c) ACC: running time (d) ACC: prediction

Figure 2: The running time and AUC for the NELL and ACC datasets, averaged over 50 runs.

target resource (i.e., file name), action (i.e., "FileCheckIn" and "FileCheckOut"), the start time and end time of the action. We used the records from 2000 most active users and extracted triples (user, action, resource) for analysis.

The statistics of the datasets are summarized in Table 1. We examined the scalability of DINTUCKER with regard to the number of machines on the NELL dataset. We set the number of latent factors to 5 in each mode. We set the subarray size to $50 \times 50 \times 50$. We randomly sampled 590,400 subarrays, so that the number of array entries processed by DINTUCKER is roughly the same as the whole array: $50 \times 50 \times 50 \times 590400 / (20000 \times 12295 \times 280) = 1.07$.

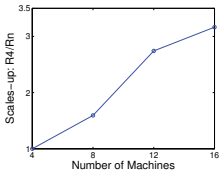


Figure 3: The scalability of DINTUCKER with regard to the number of machines on the NELL dataset.

The results are shown in Figure 3. The Y-axis shows R_4/R_n, where R_n is the running time for n machines. Note that the running speed scales up linearly.

Table 1: Statistics of multidimensional array data. B: billion, K: thousand.

Data	I	J	K	Number of entries
NELL	20K	12.3K	280	68.9B
ACC	2K	179	199.8K	71.5B

Running time and prediction accuracy

We compared DINTUCKER with GigaTensor, the state-of-the-art large scale tensor decomposition algorithm based on PARAFAC. We used the original GigaTensor implementation in JAVA on HADOOP and adopted its default setting. For DINTUCKER, we use the Matérn kernel and 5 MAPREDUCE iterations.

We set the number of latent factors for each mode to 5 for the NELL dataset and 10 for the ACC dataset. The NELL and ACC datasets contain 0.0001% and 0.003% nonzero entries, respectively. We randomly chose 80% of nonzero entries for training and then, from the remaining entries, we sampled 50 test datasets, each of which consists of 200 nonzero entries and $2,000$ zero entries. For DINTUCKER's prediction, we randomly sampled 10 subarrays of size $50 \times 50 \times 50$ for bagging.

To make a fair comparison, we trained DINTUCKER and GigaTensor using the same amount of data, which is the product of the sizes of the sampled subarrays and the number of the subarrays in the training. Also, to examine the trade-off between using fewer larger subarrays vs. using more smaller subarrays given the same computational cost, we varied the size of subarrays but kept the total number of entries for training to be the same as the number of entries in the whole array.

Figure 2 summarizes the running time and AUC of DINTUCKER and GigaTensor on the NELL and ACC datasets. The training time of DINTUCKER is given in Figures 2a and c. Note that since the training time only depends on the number and the size of subarrays, the subarray sampling strategies do no affect the training time. Figures 2a and c also demonstrate the trade-off between the communication cost and the training time over the subarrays: If we use smaller subarrays, it is faster to train the GP model over each subarray, but it incurs a larger communication/IO cost. As subarrays get smaller, the overall training time first decreases—due to less training time on each subarray—and then increases when the communication/IO cost is too large.

Figures 2b and d report the AUCs of GigaTensor and DINTUCKER based on different sampling strategies with subarray size $80 \times 80 \times 80$. They show that regardless the subarray sampling strategy, DINTUCKER outperforms GigaTensor consistently. Although GigaTensor explores data sparsity for fast computation, DINTUCKER achieves more accurate prediction in less training time.This confirms the advantage of nonlinear tensor decomposition on large array data.

Conclusion

We propose DINTUCKER, a nonparametric Bayesian learning algorithm that scales to large tensors. The core idea is to generate local GPs linked together via a hierarchical model. The connection to InfTucker in terms of model evidence is also revealed. The conclusions are general for GP and local GP models.

Acknowledgement

Shandian Zhe and Dr. Yuan Qi were supported by NSF CAREER award IIS-1054903 and IBM Research. Dr. Zenglin Xu was supported by two grants of NSF China under Nos. 61572111 and 61440036.

References

Acar, E.; Dunlavy, D. M.; Kolda, T. G.; and Mørup, M. 2011. Scalable tensor factorizations for incomplete data. *Chemometrics and Intelligent Laboratory Systems* 106(1):41–56.

Aldous, D. 1981. Representations for partially exchangeable arrays of random variables. *Journal of Multivariate Analysis* 11(4):581–598.

Bratieres, S.; Quadrianto, N.; Nowozin, S.; and Ghahramani, Z. 2014. Scalable gaussian process structured prediction for grid factor graph applications. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 334–342.

Carlson, A.; Betteridge, J.; Kisiel, B.; Settles, B.; Hruschka Jr, E. R.; and Mitchell, T. M. 2010. Toward an architecture for never-ending language learning. In *AAAI*.

Choi, J. H., and Vishwanathan, S. 2014. Dfacto: Distributed factorization of tensors. In *Advances in Neural Information Processing Systems*, 1296–1304.

Chu, W., and Ghahramani, Z. 2009. Probabilistic models for incomplete multi-dimensional arrays. *AISTATS*.

Dunson, D. B., and Fox, E. B. 2012. Multiresolution gaussian processes. In *Advances in Neural Information Processing Systems*, 737–745.

Gramacy, R. B., and Lee, H. K. 2008. Bayesian treed gaussian process models with an application to computer modeling. *Journal of the American Statistical Association* 103(483).

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

Hastie, T.; Tibshirani, R.; and Friedman, J. J. H. 2001. *The elements of statistical learning*, volume 1. Springer New York.

Hoff, P. 2011. Hierarchical multilinear models for multiway data. *Computational Statistics & Data Analysis* 55:530–543.

Hu, C.; Rai, P.; and Carin, L. 2015. Zero-truncated poisson tensor factorization for massive binary tensors. In *UAI*.

Kang, U.; Papalexakis, E.; Harpale, A.; and Faloutsos, C. 2012. 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*, 316–324. ACM.

Kim, H.-M.; Mallick, B. K.; and Holmes, C. 2005. Analyzing nonstationary spatial data using piecewise gaussian processes. *Journal of the American Statistical Association* 100(470):653–668.

Lathauwer, L. D.; Moor, B. D.; and Vandewalle, J. 2000. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl* 21:1253–1278.

Lauritzen, S. L. 2006. Exchangeable rasch matrices. In *Bruno de Finetti Centenary Conference*.

Lloyd, J. R.; Orbanz, P.; Ghahramani, Z.; and Roy, D. M. 2012. Random function priors for exchangeable arrays with applications to graphs and relational data. In *NIPS*, 1007–1015.

Rai, P.; Wang, Y.; Guo, S.; Chen, G.; Dunson, D.; and Carin, L. 2014. Scalable Bayesian low-rank decomposition of incomplete multiway tensors. In *Proceedings of the 31th International Conference on Machine Learning (ICML)*.

Rai, P.; Hu, C.; Harding, M.; and Carin, L. 2015. Scalable probabilistic tensor factorization for binary and count data. In *IJCAI*.

Rasmussen, C. E., and Ghahramani, Z. 2002. Infinite mixtures of gaussian process experts. *Advances in neural information processing systems* 2:881–888.

Shashua, A., and Hazan, T. 2005a. Non-negative tensor factorization with applications to statistics and computer vision. In *Proceedings of the 22nd ICML*.

Shashua, A., and Hazan, T. 2005b. Non-negative tensor factorization with applications to statistics and computer vision. In *Proceedings of the 22th International Conference on Machine Learning (ICML)*, 792–799.

Sun, W.; Lu, J.; Liu, H.; and Cheng, G. 2015. Provable sparse tensor decomposition. *arXiv preprint arXiv:1502.01425*.

Sutskever, I.; Tenenbaum, J. B.; and Salakhutdinov, R. R. 2009. Modelling relational data using bayesian clustered tensor factorization. In *Advances in neural information processing systems*, 1821–1828.

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

Xu, Z.; Yan, F.; and Qi, Y. 2012. Infinite Tucker decomposition: Nonparametric Bayesian models for multiway data analysis. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*.

Xu, Z.; Yan, F.; and Qi, Y. 2015. Bayesian nonparametric models for multiway data analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37(2):475–487.

Yang, Y., and Dunson, D. 2013. Bayesian conditional tensor factorizations for high-dimensional classification. *Journal of the Royal Statistical Society B, revision submitted*.

Zhe, S.; Xu, Z.; Chu, X.; Qi, Y.; and Park, Y. 2015. Scalable nonparametric multiway data analysis. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, 1125–1134.