Probabilistic Neural-Kernel Tensor Decomposition

Conor Tillinghast[†], Shikai Fang[†], Kai Zhang[‡], Shandian Zhe[†]
University of Utah[†], Temple University[‡]
ctilling@math.utah.edu[†], shikai@cs.utah.edu[†], zhang.kai@temple.edu[‡], zhe@cs.utah.edu[†]

***Abstract*—Tensor decomposition is a fundamental framework to model and analyze multiway data, which are ubiquitous in real-world applications. A critical challenge of tensor decomposition is to capture a variety of complex relationships/interactions while avoiding overfitting the data that are usually very sparse. Although numerous tensor decomposition methods have been proposed, they are mostly based on a multilinear form and hence are incapable of estimating more complex, nonlinear relationships.**

To address the challenge, we propose POND, PrObabilistic Neural-kernel tensor Decomposition that unifies the self-adaptation of Bayes nonparametric function learning and the expressive power of neural networks. POND uses Gaussian processes (GPs) to model the hidden relationships and can automatically detect their complexity in tensors, preventing both underfitting and overfitting. POND then incorporates convolutional neural networks to construct the GP kernel to greatly promote the capability of estimating highly nonlinear relationships. To scale POND to large data, we use the sparse variational GP framework and reparameterization trick to develop an efficient stochastic variational learning algorithm. On both synthetic and real-world benchmark datasets, POND often exhibits better predictive performance than the state-of-the-art nonlinear tensor decomposition methods. In addition, as a Bayesian approach, POND provides the posterior distribution of the latent factors, and hence can conveniently quantify their uncertainty and the confidence levels for predictions.

***Index Terms*—tensor decomposition, neural networks, kernel, Bayesian nonparametrics**

I. Introduction

Multiway data consist of interactions among multiple entities/nodes and are ubiquitous in real-world applications. These data are naturally represented by tensors. For example, from online shopping history logs, we can extract a four-mode tensor *(user, item, shopping site, time)*. As an important tool for multiway data analysis, tensor decomposition estimates a set of latent factors to represent the nodes in each mode and the relationship between the nodes and entry values. With the factor representations, we can uncover the hidden structures within the nodes, *e.g.,* communities and outliers, and make predictions for missing entries or downstream tasks.

However, tensor decomposition can be quite challenging. First, practical tensor data are usually extremely sparse — the majority of the entry values are not observed. Second, due to the diversity and complexity of real-world applications, among tensor nodes can be a variety of complicated, even highly nonlinear relationships and interactions. While numerous tensor decomposition algorithms have been proposed [1]–[5], they mainly rely on a multilinear decomposition form, and are inadequate in capturing more complex, nonlinear relationships. To overcome this limitation, recently a few Bayesian nonparametric decomposition methods have been proposed [6]–[9] and these methods can automatically detect the nonlinearity of the hidden relationships in data. However, they are limited by shallow and oversimplified kernels (*e.g.,* RBF), and are still not powerful enough to estimate exceedingly complicated relationships. One can consider further building a very expressive, deep neural network (DNN) for tensor decomposition. However, due to the extreme sparsity of the tensor data, a DNN with a massive number of parameters is likely to severely overfit the data and lead to suboptimal results [10].

To address these challenges, we propose POND, a probabilistic neural-kernel tensor decomposition model that inherits both the self-adaption of nonparametric Bayesian function learning to avoid overfitting and the expressive power of neural networks to capture arbitrarily complex relationships when needed. Specifically, we build POND with a Gaussian process (GP) [11] that models the relationship between the latent factors and entry values as a latent function. As a nonparametric function prior, GP never imposes a parametric form for the target function; instead GP only introduces a degree of smoothness via the definition of the covariance (or kernel) function and hence can automatically infer the complexity of the function (*e.g.,* linear and nonlinear) from data, preventing both underfitting and overfitting. Next, we use convolutional neural networks to construct a deep kernel for the GP modeling, which greatly improves upon the commonly used shallow kernels (*e.g.,* RBF) in the expressiveness and is powerful enough to estimate arbitrarily complicated relationships in data. Finally, to deal with large-scale tensor data, we combine the variational sparse GP framework [12] and the reparameterization trick [13] to develop an efficient, stochastic variational learning algorithm. Rather than point estimations, the algorithm produces the posterior distribution of the latent factors, based on which we can conveniently quantify the uncertainty of the factors and confidence levels for predictions, *e.g.,* missing entry values.

For evaluation, we compared POND with state-of-the-art multilinear and nonlinear tensor decomposition approaches. On small real data with only a few observations and synthetic data with simple multilinear relationships, POND effectively adapted to the actual data complexity and avoided overfitting the data (with over-complex estimations of the relationships). As a result, POND greatly outperforms the state-of-the-art nonlinear decomposition method, CoSTco [10] that is purely based on convolutional neural networks plus dense layers. We then examined all the methods on four real-world large, sparse, and complex datasets. In almost all the cases, POND improves upon the existing nonparametric decomposition methods with shallow kernels by a large margin, and significantly outperforms the other competing approaches in prediction accuracy. Finally, we applied POND in click-through-rate prediction for online advertising. We investigated the uncertainty information provided by POND. We showed the rational and discussed the potential applications, such as improving the display of advertisements and customer experience.

II. Preliminaries

A. Tensor Decomposition

Let us first introduce the notation and background knowledge. In this paper, we denote scalars by lowercase letters (*e.g.,* u) and occasionally uppercase letters, vectors by boldface lowercase letters (*e.g.,*$\mathbf{u}$), and matrices by boldface uppercase letters (*e.g.,*$\mathbf{U}$). We denote a K-mode tensor by $\mathcal{Y} \in \mathbb{R}^{d_1 \times \ldots \times d_K}$. Each mode k consists of d_k entities or nodes (*e.g.,* customers). We index each entry with a tuple $\mathbf{i} = (i_1, \ldots, i_K)$ where i_k is the index of the node in mode k ($1 \le k \le K$). The entry value is denoted by $y_{\mathbf{i}}$. We can flatten the tensor $\mathcal{Y}$ into a vector, which we denote by $\mathrm{vec}(\mathcal{Y})$. Each entry $\mathbf{i} = (i_1, \ldots, i_K)$ in $\mathcal{Y}$ is then mapped to the element at position $j = i_K + \sum_{t=1}^{K-1}(i_t - 1)\prod_{k=t+1}^{K} d_k$ of $\mathrm{vec}(\mathcal{Y})$.

To conduct tensor decomposition, we first introduce a set of latent factors to represent the nodes in each tensor mode. Specifically, each node j in mode k is represented by $\mathbf{u}_j^k$, an r_k dimensional vector that consists of r_k latent factors. We can then stack all the factors in mode k to construct a $d_k \times r_k$ factor matrix $\mathbf{U}^k = [\mathbf{u}_1^k, \ldots, \mathbf{u}_{d_k}^k]^\top$. We aim to use the K factor matrices $\mathcal{U} = \{\mathbf{U}^1, \ldots, \mathbf{U}^K\}$ to reconstruct the observed tensor $\mathcal{Y}$.

A classical method is Tucker decomposition [1], which assumes

$$\mathcal{Y} \approx \mathcal{W} \times_1 \mathbf{U}^1 \times_2 \ldots \times_K \mathbf{U}^K, \tag{1}$$

where $\mathcal{W} \in \mathbb{R}^{r_1 \times \ldots \times r_K}$ is a parametric (core) tensor and $\times_k$ is the mode-k tensor matrix product [14]. The result of $\mathcal{W} \times_k \mathbf{U}^k$ is a tensor of size $r_1 \times \ldots \times r_{k-1} \times d_k \times r_{k+1} \times \ldots \times r_K$. The element-wise calculation is done by $(\mathcal{W} \times_k \mathbf{U}^k)_{i_1 \ldots i_{k-1} j i_{k+1} \ldots i_K} = \sum_{i_k=1}^{r_k} w_{i_1 \ldots i_K} u^k_{j i_k}$.

If we set all $r_k = R$ and restrict $\mathcal{W}$ to be diagonal, Tucker decomposition becomes CANDECOMP/PARAFAC (CP) decomposition [2]. Denote the diagonal elements of $\mathcal{W}$ by $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_R]^\top$, we can write the CP decomposition as

$$\mathrm{vec}(\mathcal{Y}) \approx \sum_{r=1}^{R} \lambda_r \cdot \mathbf{U}^1(:, r) \otimes \ldots \otimes \mathbf{U}^K(:, r), \tag{2}$$

where $\otimes$ is the Kronecker product and $\mathbf{U}^k(:, r)$ is the r-th column of $\mathbf{U}^k$ ($1 \le k \le K$). The element-wise decomposition is then given by

$$y_{\mathbf{i}} \approx \sum_{r=1}^{R} \lambda_r \prod_{k=1}^{K} u^k_{i_k r} = \boldsymbol{\lambda}^\top (\mathbf{u}^1_{\mathbf{i}_1} \circ \ldots \circ \mathbf{u}^K_{\mathbf{i}_K}), \tag{3}$$

where $\circ$ is the Hadamard product that performs element-wise multiplication. To estimate the latent factors, Tucker and CP decomposition usually minimize a mean squared reconstruction error.

While there have been proposed many other tensor decomposition approaches, *e.g.,* [3]–[5], most of them are inherently based on the Tucker or CP decomposition forms ((1) and (2)). However, since both forms are mutilinear functions of the latent factors $\mathcal{U}$, they are incapable of estimating more complex, nonlinear relationships.

B. Nonparametric Function Learning

Many applications demand that we learn a function (mapping) from the observed input and output data. Gaussian process (GP) [11] is a powerful Bayesian nonparametric function learning model, which is not restricted by any specific function form and can self-adapt to the complexity (*e.g.,* linear or nonlinear) of the function hidden in the data, hence preventing both underfitting and overfitting. Specifically, suppose we are given a set of training inputs and outputs, $\mathcal{D} = (\mathbf{X}, \mathbf{y})$ where $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top$ and $\mathbf{y} = [y_1, \ldots, y_N]^\top$. To learn the underlying function $f(\cdot)$, we first place a GP prior over the function, $f \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot))$ where $m(\cdot)$ is the mean function, usually set to 0, and $k(\cdot, \cdot)$ is the covariance (or kernel) function. That is, each function value is considered as a random variable and the collection of all the values of $f(\cdot)$ (at all the possible inputs in the domain) are (jointly) sampled from a Gaussian process. Note that a random process can consist of infinitely many random variables. According to the definition of GP, any finite set of random variables in the process follow a multivariate Gaussian distribution. Therefore, the function values at the training inputs $\mathbf{X}$, namely, $\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)]^\top$ has a multivariate Gaussian prior distribution,

$$p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K}) \tag{4}$$

where $\mathbf{K}$ is an $N \times N$ kernel matrix on $\mathbf{X}$ — $[\mathbf{K}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. A commonly used kernel function is the RBF kernel, $k_{\mathrm{RBF}}(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{1}{\eta}\|\mathbf{x}_i - \mathbf{x}_j\|^2)$ where η is the kernel parameter. Given the function values $\mathbf{f}$, we then sample the observed outputs $\mathbf{y}$ from a noise model. For continuous observations, we can use a Gaussian noise model, $p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{y}|\mathbf{f}, \tau^{-1}\mathbf{I})$ where τ is the inverse variance of the noise. We can marginalize out $\mathbf{f}$ to obtain the marginal likelihood,

$$p(\mathbf{y}|\mathbf{X}, \tau) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K} + \tau^{-1}\mathbf{I}). \tag{5}$$

To learn the GP model, we can maximize the likelihood to estimate the kernel parameters and the inverse variance τ. Given a test input $\mathbf{x}^*$, since the function value $f(\mathbf{x}^*)$ and the training outputs $\mathbf{y}$ also jointly follow a multivariate Gaussian distribution, the posterior of $f(\mathbf{x}^*)$ is a conditional Gaussian,

$$p\big(f(\mathbf{x}^*)|\mathbf{x}^*, \mathbf{X}, \mathbf{y}\big) = \mathcal{N}\big(f(\mathbf{x}^*)|\mu^*, v^*\big), \tag{6}$$

where $\mu^* = \mathbf{k}_*^\top(\mathbf{K} + \tau^{-1}\mathbf{I})^{-1}\mathbf{y}$, $v^* = k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}_*^\top(\mathbf{K} + \tau^{-1}\mathbf{I})^{-1}\mathbf{k}_*$ and $\mathbf{k}_* = [k(\mathbf{x}^*, \mathbf{x}_1), \cdots, k(\mathbf{x}^*, \mathbf{x}_N)]^\top$.

We can see that GP never assumes a parametric form of the target function; instead, it uses the kernel function $k(\cdot, \cdot)$ to induce a degree of smoothness, namely, how close the function values are according to the similarity of their inputs. This is reflected in the correlation (covariance) matrix in the joint distribution (4) and (5). Therefore, GP can automatically adapt to the complexity of the function in the data. In addition, the posterior distribution of the function outputs are Gaussian, which is nice and convenient for uncertainty quantification.

III. Model

Tensor data can contain a variety of relationships due to the diversity of real-world applications. These relationships can be relatively simple (say, multilinear) to exceedingly complex. Despite the conciseness of the Tucker/CP based decomposition methods, they are not flexible enough to identify nonlinear relationships in data. On the other hand, tensor data are typically extremely sparse. Most entries are unobserved and unknown. If we use a complicated, parametric model, such as densely connected neural networks, the model can easily overfit the data and result in inferior performance. To address these challenges, we hybridize Bayesian nonparametric tensor decomposition and neural network kernels. In this way, our model can (1) self-adapt to the complexity of the data, avoiding both overfitting and underfitting, and (2) enjoy the expressive power of the neural networks, being capable of capturing very complicated relationships when present in data.

A. Nonparametric Tensor Decomposition

First, to automatically capture the complexity of the relationship in tensors, we will use the nonparametric function learning model, *i.e.,* Gaussian process. Specifically, we consider the relationship between the latent factors and entry values as an unknown function $f : \mathbb{R}^{\sum_k r_k} \rightarrow \mathbb{R}$. For each entry $\mathbf{i}$, we have $y_{\mathbf{i}} = f(\mathbf{x_i})$, where the input $\mathbf{x_i} = [(\mathbf{u}^1_{i_1})^\top, \ldots, (u^K_{i_K})^\top]^\top$ consist of all the latent factors associated with entry $\mathbf{i}$. We then place a GP prior over $f(\cdot)$. Given the collection of the observed tensor entries, $\mathcal{S} = \{\mathbf{i}_1, \ldots, \mathbf{i}_N\}$, the function values $\mathbf{f}_S = [f(\mathbf{x}_{\mathbf{i}_1}), \ldots, f(\mathbf{x}_{\mathbf{i}_N})]^\top$ follow a multivariate Gaussian distribution,

$$p(\mathbf{f}_S|\mathcal{U}) = \mathcal{N}(\mathbf{f}_S|\mathbf{0}, \mathbf{K}_{SS}) \tag{7}$$

where $\mathbf{K}_{SS}$ is the kernel matrix of the N inputs constructed from the latent factors $\mathcal{U}$, and each element $[\mathbf{K}_{SS}]_{m,n} = k(\mathbf{x}_{\mathbf{i}_m}, \mathbf{x}_{\mathbf{i}_n})$ is a covariance (kernel) function of the corresponding input vectors. Given the latent function values $\mathbf{f}$, we then use a noise model to generate the observed entry values $\mathbf{y}_S = [y_{\mathbf{i}_1}, \ldots y_{\mathbf{i}_N}]^\top$. In this paper, we mainly focus on continuous values and hence we use the Gaussian noise model, $p(\mathbf{y}_S|\mathbf{f}_S) = \mathcal{N}(\mathbf{y}_S|\mathbf{f}_S, \tau^{-1}\mathbf{I})$ where τ is the inverse noise variance. It is straightforward to modify our model and learning algorithm to support other types of data, *e.g.,* binary tensors. We assign a standard normal prior over the latent factors. The joint probability of our model is given by

$$\begin{aligned} p(\mathbf{y}_S, \mathbf{f}_S, \mathcal{U}|\tau) = \prod_{k=1}^{K}\prod_{j=1}^{d_k} \mathcal{N}(\mathbf{u}^k_j|\mathbf{0}, \mathbf{I}) \\ \cdot \mathcal{N}(\mathbf{f}_S|\mathbf{0}, \mathbf{K}_{SS})\mathcal{N}(\mathbf{y}_S|\mathbf{f}_S, \tau^{-1}\mathbf{I}). \end{aligned} \tag{8}$$

Note that a major difference from the standard GP (see Section II-B) is that the inputs in our model are unknown latent factors and need to be jointly estimated with the kernel parameters and inverse noise variance. Therefore, our nonparametric decomposition is essentially a latent-input GP model.

B. Convolutional Neural Network Kernels

The nonparametric decomposition enables a great flexibility in capturing different types of relationships in data, *e.g.,* linear, multilinear and nonlinear; its self-adaption can effectively avoid underfitting and overfitting the data. However, the expressiveness of our nonparametric decomposition can be severely limited by the commonly used shallow kernels, such as RBF. These kernels make oversimplified assumptions about the function smoothness (*e.g.,* RBF kernel assumes the function is infinitely differentiable) and can disable our model from learning highly complicated functions/relationships (when needed). To overcome this limitation, we use convolutional neural networks to construct a much more expressive kernel so as to accommodate arbitrarily complex functions.

Specifically, in our new kernel, we first feed every input into a convolutional neural network to perform a nonlinear feature transformation, and then use the output of the network (*i.e.,* transformed features) to compute the shallow kernel, *e.g.,* RBF. For each entry $\mathbf{i}$, the input $\mathbf{x_i}$ has $\sum_{k=1}^{K} r_k$ dimensions, obtained by concatenating the latent factors in each mode associated with $\mathbf{i}$. To perform convolution, we first organize $\mathbf{x_i}$ into an $R \times K$ matrix $\mathbf{X_i}$, where $R = \max(r_1, \ldots, r_K)$. Each column k consists of the associated factors in mode k, namely, $\mathbf{u}^k_{i_k}$, appended with extra zeros if $r_k < R$. In the first layer, we use C channels of square filters, each of which is 2 × 2 or 3 × 3. In this way, we extract and integrate the neighbouring information in $\mathbf{X_i}$. We add zero paddings to maintain the original size of the input matrix. Therefor, the output of the first layer is a $C \times R \times K$ tensor. In the second layer, we use C channels of $R \times 1$ filters. In this way, we aggregate the information along different modes and the output is $C \times 1 \times K$. In the third layer, we apply one $1 \times K$ filter to aggregate the information across the modes. Finally, we obtain a C dimensional feature vector. We use this vector to compute the RBF kernel function. After each convolution, we add a bias term and apply a nonlinear activation function. In our experiments, we chose $\tanh(\cdot)$ due to its excellent performance. The convolution process in our neural kernel is summarized as follows:

$$\begin{aligned} \mathcal{X}_{\mathbf{i}}^{(\text{conv1})} &= \sigma\big(\text{Conv}(\phi_1, \mathbf{X_i}) + \beta_1\big) \in \mathbb{R}^{C\times R\times K}, \\ \mathcal{X}_{\mathbf{i}}^{(\text{conv2})} &= \sigma\big(\text{Conv}(\phi_2, \mathcal{X}_{\mathbf{i}}^{(\text{conv1})}) + \beta_2\big) \in \mathbb{R}^{C\times 1\times K}, \\ \mathcal{X}_{\mathbf{i}}^{(\text{conv3})} &= \sigma\big(\text{Conv}(\phi_3, \mathcal{X}_{\mathbf{i}}^{(\text{conv2})}) + \beta_3\big) \in \mathbb{R}^{C\times 1\times 1} \end{aligned}$$

where $\sigma(\cdot)$ is the nonlinear activation function, ϕ_1, ϕ_2 and ϕ_3 are convolutional filters, and β_1, β_2 and β_3 are bias terms. For any two entries $\mathbf{i}$ and $\mathbf{j}$, we apply the above procedure to obtain $\mathcal{X}_{\mathbf{i}}^{\text{conv3}}$ and $\mathcal{X}_{\mathbf{j}}^{\text{conv3}}$, and then compute the kernel by

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp\big(-\frac{\|\text{vec}(\mathcal{X}_{\mathbf{i}}^{\text{conv3}}) - \text{vec}(\mathcal{X}_{\mathbf{j}}^{\text{conv3}})\|^2}{\eta}\big). \tag{9}$$

We use (9) as the kernel function to construct $\mathbf{K}_{SS}$ in our nonparametric tensor decomposition model (8). In this way, we unify the self-adaption of nonparametric function learning and the expressive power of neural networks.

IV. Algorithm

We now present our model estimation algorithm. Given the observed tensor entries, we aim to estimate the posterior distribution of the latent factors $\mathcal{U}$, which can be very useful for uncertainty quantification. Note that in practice the activities or degrees of the nodes/entities in each mode can vary much. Those popular or active nodes (*e.g.,* hub nodes) appear in many observed entries, and hence their factor estimations are much more reliable/confident than inactive nodes only observed in a few entries. These uncertainties in turn influence/determine the confidence levels of missing value predictions and of other downstream tasks, such as in online advertising and recommendation.

However, the exact inference of our model is infeasible. First, the latent factors are coupled in complex neural kernels and the posterior distribution does not have any closed form. Second, the joint probability in (8) requires us to calculate the $N \times N$ covariance matrix $\mathbf{K}_{SS}$ and its inverse; when N (#entries) is large, the computation is prohibitively costly ($\mathcal{O}(N^3)$ time complexity). To address these issues, we develop an efficient stochastic variational inference algorithm that provides analytical, approximate posterior estimations and scales up to a large number of observed entries.

A. Decomposed Variational Model Evidence Lower Bound

Specifically, we first use the sparse variational GP framework [12] to develop a tractable variational model evidence lower bound (ELBO) that dispenses with the huge covariance matrix and is fully additive over the tensor entries. We then develop an efficient stochastic optimize algorithm to maximize the ELBO to estimate the approximate posteriors and the other parameters. To this end, we first introduce a set of M pseudo inputs, $\mathbf{Z} = [\mathbf{z}_1, \dots \mathbf{z}_M]^\top$ where $M \ll N$ and each $\mathbf{z}_m$ $(1 \le m \le M)$ has the same length as each $\mathbf{x}_{\mathbf{i}_n}(1 \le n \le N)$. We denote the values of $f(\cdot)$ at $\mathbf{Z}$ by $\mathbf{b} = [f(\mathbf{z}_1), \dots, f(\mathbf{z}_M)]^\top$, which we refer to as the pseudo outputs. We then augment our decomposition model by jointly sampling $\mathbf{f}_S$ and $\mathbf{b}$. Due to the GP prior over $f(\cdot)$, $\mathbf{f}_S$ and $\mathbf{b}$ jointly follow a multivariate Gaussian distribution,

$$p(\mathbf{f}_S, \mathbf{b}) = \mathcal{N}\left(\begin{bmatrix} \mathbf{f}_S \\ \mathbf{b} \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{SS} & \mathbf{K}_{SZ} \\ \mathbf{K}_{ZS} & \mathbf{K}_{ZZ} \end{bmatrix}\right) \tag{10}$$

where $\mathbf{K}_{ZZ}$ is the kernel matrix on $\mathbf{Z}$, each $[\mathbf{K}_{ZZ}]_{mt} = k(\mathbf{z}_m, \mathbf{z}_t)$, $\mathbf{K}_{SZ}$ is the cross kernel matrix between $\mathbf{X}_S$ and $\mathbf{Z}$, each $[\mathbf{K}_{SZ}]_{nm} = k(\mathbf{x}_{\mathbf{i}_n}, \mathbf{z}_m)$, and $\mathbf{K}_{ZS} = \mathbf{K}_{SZ}^\top$. We further decompose

$$p(\mathbf{f}_S, \mathbf{b}) = p(\mathbf{b})p(\mathbf{f}_S|\mathbf{b})$$

where $p(\mathbf{b}) = \mathcal{N}(\mathbf{b}|\mathbf{0}, \mathbf{K}_{ZZ})$ and $p(\mathbf{f}_S|\mathbf{b}) = \mathcal{N}(\mathbf{f}_S|\mathbf{K}_{SZ}\mathbf{K}_{ZZ}^{-1}\mathbf{b}, \mathbf{K}_{SS} - \mathbf{K}_{SZ}\mathbf{K}_{ZZ}^{-1}\mathbf{K}_{ZS})$ is a conditional Gaussian distribution. Now the joint probability of the augmented model is given by

$$\begin{aligned} p(\mathbf{y}_S, \mathbf{b}, \mathbf{f}_S, \mathcal{U}|\mathbf{Z}, \tau) = & \prod_{k=1}^{K}\prod_{j=1}^{d_k} \mathcal{N}(\mathbf{u}_j^k|\mathbf{0}, \mathbf{I}) \\ & \cdot p(\mathbf{b})p(\mathbf{f}_S|\mathbf{b})\mathcal{N}(\mathbf{y}_S|\mathbf{f}_S, \tau^{-1}\mathbf{I}). \end{aligned} \tag{11}$$

Note that the augmented model is equivalent to the original model. When we marginalize out the pseudo outputs $\mathbf{b}$, the joint prior in (10) becomes (7) and accordingly we recover the original probability (8).

Now based on the augmented model (11), we construct a variational ELBO. We introduce an approximate posterior distribution of the latent factors $\mathcal{U}$, the latent function values $\mathbf{f}_S$ and pseudo outputs $\mathbf{b}$ in the following form,

$$q(\mathcal{U}, \mathbf{b}, \mathbf{f}_S) = \prod_{k=1}^{K}\prod_{t=1}^{d_k} q(\mathbf{u}_t^k)q(\mathbf{b})p(\mathbf{f}_S|\mathbf{b}) \tag{12}$$

where $q(\mathbf{b}) = \mathcal{N}(\mathbf{b}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and each $q(\mathbf{u}_t^k) = \mathcal{N}\big(\mathbf{u}_t^k|\boldsymbol{\alpha}_t^k, \operatorname{diag}(\mathbf{v}_t^k)\big)$ is a diagonal Gaussian distribution. Note that we intentionally introduce the full conditional Gaussian distribution $p(\mathbf{f}_S|\mathbf{b})$ to construct the approximate posterior so as to obtain a decomposed lower bound, which we will explain later. We further parameterize $\boldsymbol{\Sigma}$ by their Cholesky decomposition, $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^\top$ to ensure the positive definiteness, and each $\mathbf{v}_t^k = \exp(\boldsymbol{\gamma}_t^k)$ to ensure the positiveness. Now, we derive the variational ELBO [15] from

$$\mathcal{L} = \mathbb{E}_{q(\mathcal{U}, \mathbf{b}, \mathbf{f}_S)}\big[\log \frac{p(\mathbf{y}_S, \mathbf{b}, \mathbf{f}_S, \mathcal{U}|\mathbf{Z}, \tau)}{q(\mathcal{U}, \mathbf{b}, \mathbf{f}_S)}\big]. \tag{13}$$

It is known that (according to Jensen's inequality)

$$\mathcal{L} + \mathrm{KL}\big(q(\mathcal{U}, \mathbf{b}, \mathbf{f}_S)\|p(\mathcal{U}, \mathbf{b}, \mathbf{f}_S|\mathbf{y}_S)\big) = \log p(\mathbf{y}_S),$$

where $\mathrm{KL}(\cdot\|\cdot)$ is the Kullback Leibler divergence between two distributions and $\mathrm{KL} \ge 0$. Given $\log p(\mathbf{y}_S)$ — the fixed log probability of the observed data, *i.e.,* the model evidence, maximizing the ELBO $\mathcal{L}$ w.r.t to the approximate posterior q is equivalent to minimizing the KL divergence between q and the true posterior $p(\mathcal{U}, \mathbf{b}, \mathbf{f}_S|\mathbf{y}_S)$. In other words, we can find the best approximate posterior distribution (parameterized by (12)) by maximizing the ELBO, rather than directly calculating the KL divergence, which is infeasible.

Now, we substitute (11) and (12) into (13) and obtain

$$\begin{aligned} \mathcal{L} = & -\prod_{k=1}^{K}\prod_{j=1}^{d_k} \mathrm{KL}\big(q(\mathbf{u}_j^k)\|p(\mathbf{u}_j^k)\big) - \mathrm{KL}\big(q(\mathbf{b})\|p(\mathbf{b})\big) \\ & + \sum_{n=1}^{N} \mathbb{E}_{q(\mathcal{U}, \mathbf{b}, f_n)}[\log \mathcal{N}(y_{\mathbf{i}_n}|f_n, \tau^{-1})] \end{aligned} \tag{14}$$

where $p(\mathbf{u}_j^k) = \mathcal{N}(\mathbf{u}_j^k|\mathbf{0}, \mathbf{I})$ is the prior distribution of $\mathbf{u}_j^k$, and $f_n = [\mathbf{f}_S]_n = f(\mathbf{x}_{\mathbf{i}_n})$. Note that the full conditional Gaussian distribution $p(\mathbf{f}_S|\mathbf{b})$ in both (11) and (12) have been cancelled inside the log ratio of (13). Consequently, we do not need to explicitly compute the full covariance matrix $\mathbf{K}_{SS}$ and the computational cost is greatly reduced. In addition, according to (12), we have

$$q(\mathcal{U}, \mathbf{b}, f_n) = \prod_{k=1}^{K}\prod_{t=1}^{d_k} q(\mathbf{u}_t^k) q(\mathbf{b}) p(f_n|\mathbf{b}), \tag{15}$$

where

$$p(f_n|b) = \mathcal{N}(f_n|\mu_n, \sigma_n^2) \tag{16}$$

is a scalar conditional Gaussian distribution and quite easy to calculate, $\mu_n = \mathbf{k}_n^\top \mathbf{K}_{ZZ}^{-1}\mathbf{b}$, $\sigma_n^2 = k(\mathbf{x}_{\mathbf{i}_n}, \mathbf{x}_{\mathbf{i}_n}) - \mathbf{k}_n^\top \mathbf{K}_{ZZ}^{-1}\mathbf{k}_n$, and $\mathbf{k}_n = [k(\mathbf{x}_{\mathbf{i}_n}, \mathbf{z}_1), \ldots, k(\mathbf{x}_{\mathbf{i}_n}, \mathbf{z}_M)]^\top$. Now we can see that the computation of the ELBO (14) is decomposed over individual tensor entries. Rather than calculate the huge $N \times N$ covariance matrix $\mathbf{K}_{SS}$, we only need to compute a kernel matrix $\mathbf{K}_{ZZ}$ and its inverse on a small set of M pseudo inputs. The additive form further enables us to conduct efficient stochastic optimization, presented as follows.

B. Stochastic Optimization

We aim to maximize the ELBO in (14) to estimate the approximate posterior of the latent factors $\mathcal{U}$ and pseudo outputs $\mathbf{b}$, the pseudo inputs $\mathbf{Z}$, the inverse noise variance τ, and the parameters of our neural kernel, including those in the convolutional filters and the internal RBF kernel. Despite its decomposed form, $\mathcal{L}$ is not analytical, because each expectation is under the (approximate) posterior distribution of the latent factors $\mathcal{U}$; the latent factors are coupled in the complex kernel computation in $p(f_n|b)$ (see (16)) and so in $\log \mathcal{N}(y_{\mathbf{i}_n}|f_n, \tau^{-1})$ (see (14)); hence the expectation does not have any closed form. Moreover, when N is large, the full summation across all the entries is still quite expensive. To address these challenges, we develop an efficient stochastic optimization algorithm. We first partition the entries into mini-batches of size B, $\mathcal{Q} = \{Q_1, \ldots, Q_{N/B}\}$, and rearrange the ELBO as

$$\mathcal{L} = -\prod_{k=1}^{K}\prod_{j=1}^{d_k} \mathrm{KL}\big(q(\mathbf{u}_j^k)\|p(\mathbf{u}_j^k)\big) - \mathrm{KL}\big(q(\mathbf{b})\|p(\mathbf{b})\big) + \frac{B}{N}\sum_{j=1}^{N/B}\frac{N}{B}\sum_{\mathbf{i}_n \in \mathbf{Q}_j} \mathbb{E}_{q(\mathcal{U},\mathbf{b},f_n)}[\log \mathcal{N}(y_{\mathbf{i}_n}|f_n, \tau^{-1})]. \tag{17}$$

Then the ELBO can be viewed as an expectation of a stochastic objective,

$$\mathcal{L} = \mathbb{E}_{p(t)}[\tilde{\mathcal{L}}_t] \tag{18}$$

where $p(t) = \frac{B}{N}$, $t \in \{1, \ldots, \frac{N}{B}\}$, and

$$\tilde{\mathcal{L}}_t = -\prod_{k=1}^{K}\prod_{j=1}^{d_k} \mathrm{KL}\big(q(\mathbf{u}_j^k)\|p(\mathbf{u}_j^k)\big) - \mathrm{KL}\big(q(\mathbf{b})\|p(\mathbf{b})\big) + \frac{N}{B}\sum_{\mathbf{i}_n \in \mathbf{Q}_t} \mathbb{E}_{q(\mathcal{U},\mathbf{b},f_n)}[\log \mathcal{N}(y_{\mathbf{i}_n}|f_n, \tau^{-1})]. \tag{19}$$

Now we can develop a stochastic optimization algorithm based on (18). Each iteration, we first sample a mini-batch Q_t and then compute the gradient of the stochastic objective $\tilde{\mathcal{L}}_t$ as an unbiased stochastic gradient of $\mathcal{L}$. However, in $\tilde{\mathcal{L}}_t$ each expectation term is intractable. To address this problem, we use the reparameterization trick [13]. For each $\mathbf{i}_n \in Q_t$, we first generate parameterized samples for the latent factors $\{\mathbf{u}_{i_{nk}}^k\}$ associated with $\mathbf{i}_n$ from their approximate posteriors, $\tilde{\mathbf{u}}_{i_{nk}}^k = \boldsymbol{\alpha}_{i_{nk}}^k + \mathrm{diag}(\sqrt{\mathbf{v}_{i_{nk}}^k}) \cdot \boldsymbol{\eta}$ where $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$; then we concatenate them to obtain the parameterized sample for $\mathbf{x}_{\mathbf{i}_n}$: $\tilde{\mathbf{x}}_{\mathbf{i}_n} = [(\tilde{\mathbf{u}}_{i_{n1}}^1)^\top, \ldots, (\tilde{\mathbf{u}}_{i_{nK}}^K)^\top]^\top$. Next, we generate a parameterized sample for the pseudo outputs $\mathbf{b}$, $\tilde{\mathbf{b}} = \boldsymbol{\mu} + \boldsymbol{\Sigma} \cdot \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Finally, we generate the parameterized sample for f_n, $\tilde{f}_n = \mu_n + \sigma_n \cdot \epsilon$ where $\epsilon \sim \mathcal{N}(0, 1)$. Note that $\tilde{\mathbf{x}}_{\mathbf{i}_n}$ and $\tilde{\mathbf{b}}$ are used to compute μ_n and σ_n (see (16)). Now we substitute $\tilde{f}_n$ for f_n in each expectation term of $\tilde{\mathcal{L}}_t$ and obtain an unbiased stochastic estimate of $\tilde{\mathcal{L}}_t$. Then we calculate the gradient of the estimate, which will be an unbiased stochastic gradient of $\tilde{\mathcal{L}}_t$, and in turn an unbiased stochastic gradient of $\mathcal{L}$. Next, we can use any stochastic optimization algorithm to maximize $\mathcal{L}$. The computation of the stochastic gradient is restricted in the mini-batch only, and hence is very cheap and efficient. Finally, the overall model estimation procedure is summarized in Algorithm 1.

Algorithm 1 POND ($\mathcal{S}$, B, M, T, γ_0)

```
Initialize variational posterior $q(\mathbf{b}) = \mathcal{N}(\mathbf{b}|\mathbf{0}, \mathbf{I})$ and each $q(\mathbf{u}_t^k) = \mathcal{N}(\mathbf{u}_t^k|\mathbf{0}, \mathbf{I})$ in (12).
Initialize $M$ pseudo inputs $\mathbf{Z}$ by sampling from the standard normal distribution.
for epoch = $1..T$ do
    Randomly shuffle the training tensor entries $\{y_{\mathbf{i}_n}\}(\mathbf{i}_n \in \mathcal{S})$ and partition the entries into mini-batches of size $B$.
    for each mini-batch $Q_t$ do
        Calculate the unbiased stochastic gradient $\nabla\widehat{\mathcal{L}}_t$ for (18).
        Update all the parameters: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \gamma_t \nabla\widehat{\mathcal{L}}_t$.
        Adjust the learning rate $\gamma_t$.
    end for
end for
return The posteriors of the latent factors $\{q(\mathbf{u}_t^k) = \mathcal{N}\big(\mathbf{u}_t^k|\boldsymbol{\alpha}_t^k, \mathrm{diag}(\mathbf{v}_t^k)\big)\}$ and the pseudo outputs $\mathcal{N}(\mathbf{b}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $\mathbf{Z}$, the inverse noise variance $\tau$ and the kernel parameters.
```

C. Algorithm Complexity

The time complexity of our model estimation algorithm is $\mathcal{O}(NM^3/B + MN)$, which is for computing the kernel matrix on the pseudo inputs $\mathbf{Z}$ and the cross kernel between the latent factors in each entry in the mini-batches and the pseudo inputs. Since the number of pseudo inputs M is fixed (typically 100 or 200) and close to B, both $M, B \ll N$, the computational cost is proportional to the data size N. The space complexity is $\mathcal{O}(\sum_{k=1}^K d_k R + M^2)$, including the storage of the approximate posterior of the latent factors and the pseudo outputs, and the kernel matrix on the pseudo inputs.

V. Related Work

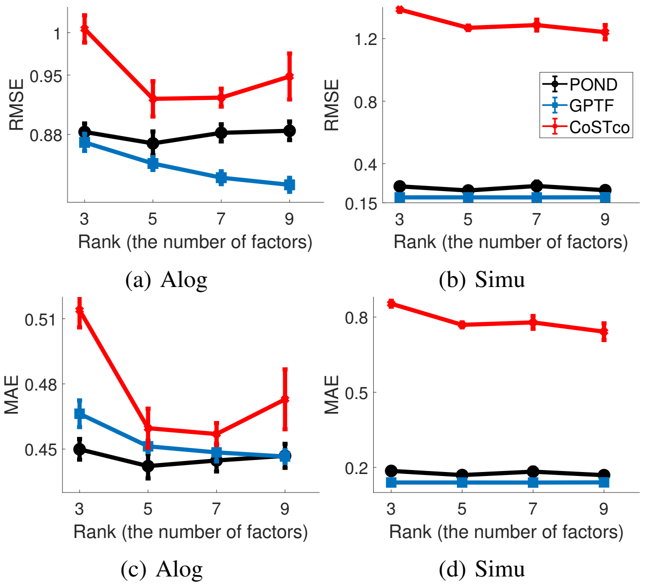
Classical tensor decomposition methods include Tucker [1] and CP [2] decompositions, based on which many works have been done, such as [3]–[5], [16]–[19]. Despite the widespread success of these methods, their underlying multilinear factorization structure can limit their capability to capture more complex, nonlinear interactions in real-world applications. To overcome this problem, a few nonparametric tensor decomposition methods were developed recently [6]–[9], which used GPs to capture possible nonlinear relationships in data. Along this line of research, the state-of-the-art is DFNT [9], which also constructs an input vector for each entry by concatenating the associated latent factors, and learns the unknown relationship with a GP. However, DFNT uses a shallow kernel and has a limited capability of estimating highly complex relationships. For training, DFNT integrates out the optimal variational posterior to obtain a tight variational ELBO, for which a distributed batch optimization algorithm is developed. Despite its tightness, the ELBO might be more difficult to optimize. In our experiments, the proposed stochastic learning achieves much better predication accuracy, even with a shallow kernel. Finally, DFNT only gives a point estimation of the latent factors and cannot quantify the uncertainty. The recent works have also developed a streaming version [20], [21] of the these multilinear and nonlinear decomposition models.

There have been a few attempts to apply (deep) neural networks for tensor decomposition [10], [22], [23]. However, as shown in the most recent work [10], using densely connected networks, *e.g.,* multi-layer perceptron (MLP), tends to overfit sparse tensor data that is often the case in practice. To alleviate this problem, a new neural decomposition method, CoSTco, is proposed in [10]. CoSTco first applies two convolutional layers to integrate the local information from the latent factors in each entry, and then adds dense layers to produce the entry values. Compared with MLP-based and other nonlinear decomposition methods, CoSTco often shows better performance in missing value prediction. However, as a non-probabilistic approach, CoSTco cannot quantify the uncertainty of the latent factor estimations and predictions for missing entries. When the observed data are a few or the interactions are relatively simple, CoSTco is still at risk of overfitting (see Section VI-A).

VI. Experiment

For evaluation, we conducted experiments to answer the following questions. **Q1:** How does POND perform when the observed tensor entries are a few or the hidden relationships are actually simple, and will it tend to be overfitting? **Q2:** Howe does POND perform in completing real-world large and complex tensors, with a variety of sparse levels? **Q3:** What does the uncertainty information provided by POND reflect, and how does it potentially benefit real applications, such as online advertising?

To answer the first question, we used two datasets, one is real and the other synthetic. To answer the second question, we tested POND in four real-world sparse tensors. To answer the third question, we ran POND on a real-world mobile ads click-through-rate dataset, and investigated the posterior mean and variance of the latent factors and predictions of the click probabilities.



(a) Alog (b) Simu

(c) Alog (d) Simu

Fig. 1: Prediction error on two small datasets. The results were averaged from 5 runs.

A. Evaluation on Small or Simple Data

First, we tested POND on the following two datasets: (1) *Alog* [9], a three-mode tensor of size 200 × 100 × 200, representing three-way file-access operations *(user, action, resource)*. It includes 0.66% observed entries. (2) *Simu*, a $100 \times 100 \times 100$ synthetic tensor. For each node, we drew three latent factors from the standard normal distribution, and then simulated the entry values through the CP form (see (3)), where we set $\boldsymbol{\lambda} = [0.8, 0.7, 0.5]$.

We compared with (1) GPTF, our Gaussian process tensor decomposition model (see Section III-A) only using the RBF kernel, and (2) CoSTco [10], the state-of-the-art nonlinear tensor decomposition model, which is purely based on neural networks, including 2 convolutional layers and 2 dense layers. We implemented GPTF and POND with TensorFlow [24] and used ADAM [25] for optimization. We set the number of pseudo inputs M to 100. For CoSTco, we used the original implementation (https://github.com/USC-Melady/KDD19-CoSTCo). For all the methods, we set the maximum number of epochs to 300. We chose the learning rate from $\{10^{-4}, 2\times 10^{-4}, 5\times 10^{-4}, 10^{-3}, 2\times 10^{-3}, 5\times 10^{-3}, 10^{-2}\}$.

To examine if these approaches tend to overfit when the number of observed entries is very small, we randomly sampled 21K observed entries from *Alog* for training. Furthermore, to examine if these approaches can fail to estimate the simple multilinear relationships in data due to (complex) nonlinear modeling, we randomly sampled 200K entries from *Simu* for training. We then computed the root-mean-square-error (RMSE) and mean-absolute-error (MAE) of each method in tensor completion. We varied the number of latent factors, *i.e.,* rank, from {3, 5, 7, 9}. We repeated each test for five times, and reported the average RMSE, average MAE and their standard deviations in Figure 1. As we can see, on both dataset, POND is significantly better than CoSTco (p-value <0.05, shown by non-overlapping standard error

TABLE I: Statistics of real-world sparse tensors.

Dataset	Size	#Observed entries
MovingMNIST	$20 \times 100 \times 64 \times 64$	819200
ExtremeClimate	$360 \times 16 \times 768 \times 1152$	41198
SG	$2321 \times 5596 \times 1600$	105764
Gowalla	$18737 \times 1000 \times 32510$	821931

bars [26]). Although both methods use neural networks to enhance the expressive power, POND is much less likely to overfit the data, which might be due to the self-adaption of its nonparametric modeling to the data complexity. CoSTco performed particularly worse in *Simu*, implying that it always attempted to use a very complicated function to fit the data. Note that the CoSTco implementation has already used early-stopping to alleviate overfitting. Furthermore, in most cases GPTF shows the smallest RMSE/MAE. This is reasonable, because in addition to nonparametric modeling, GPTF uses the simple shallow RBF kernel, including only one kernel parameter, and hence is more robust in small/simple data. The results of GPTF further confirms the advantage of our nonparametric decomposition.

B. Evaluation on Real-World Sparse Tensors

Next, we tested POND on four real-world sparse tensors, whose statistics are summarized Table I. (1) MovieMNSIT [27], a four-mode *(video, time, row, column)* tensor that represents 20 grey-scale videos. The observed entries take 10% of total elements. (2) ExtremeClimate [28], a four-mode tensor extracted from the history records of extreme weather cases, *(time, lattitute, longitude, variable)*. The entire tensor has only $\%0.0008$ observed entries. (3) SG [29], a three-mode tensor describing *(user, location, point of interest)* checkins, collected by Foursquare in Singapore. There are $\%0.0005$ observed entries. (4) Gowalla [30], another checkin tensor collected by Foursquare all over the world, having $\%0.0001$ observed entries. Note that the CoSTco paper [10] has used all these data sources to verify the performance of CoSTco. We exactly followed the CoSTco paper to extract and preprocess the data. Hence the details are referred to the paper.

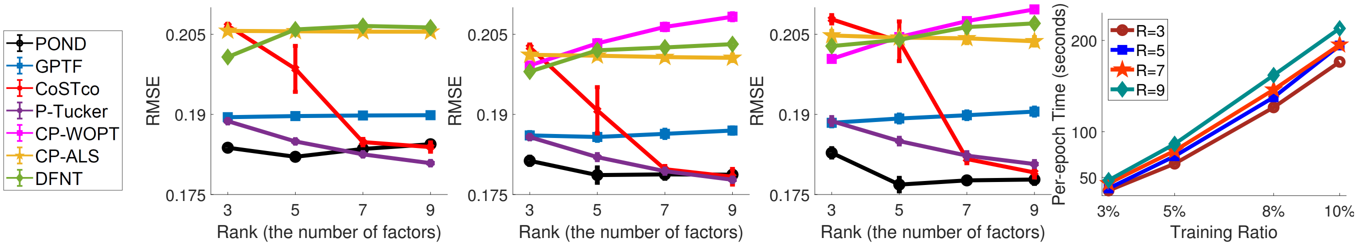
Competing methods. We compared POND with the following baselines: (1) CoSTco; (2) GPTF; (3) DFNT [9], a distributed nonlinear tensor factorization based on GPs; (4) CP-WOPT [17], a scalable CP decomposition algorithm based on conjugate gradient descent; (5) CP-ALS [31], an efficient CP decomposition algorithm using alternative least square update; (6) P-Tucker [32], a scalable Tucker decomposition algorithm that performs parallel row-wise updates.

We first tested all the methods on MovieMNIST. To investigate their performance under different sparsity levels, we randomly chose {3%, 5%, 10%} observed entries for training and tested the performance in predicting the remaining entries. We varied the rank from {3, 5, 7, 9}. For each training ratio, we repeated the experiment for five times, and then reported the average RMSE and their standard deviations in Figure 2a-c. Note that we did not show CP-WOPT's result in Figure 2a, because it is far worse than all the other methods. As we can see, POND nearly always performs the best. In particular, POND outperforms GPTF by a large margin, showing the advantage of our neural kernel in capturing very complex relationships. In all the cases, POND either significantly outperforms CoSTco (when the rank is small) or obtains close accuracy to CoSTco (when the rank is large). The performance of POND does not change as much along with the rank as CoSTco. This might be because our nonparametric decomposition is more flexible to adapt to the data, and less dependent on the chose of hyper-parameters. It is worth noting that DFNT is far worse than GPTF, although they both use GPs and shallow kernels. The difference might arise from the training procedure. DFNT uses a tight yet hard ELBO as the training objective, and its batch optimization might early converge to inferior local maximums. By contrast, GPTF and POND, use stochastic optimization to maximize a relatively easy ELBO, and can prevent being quickly trapped in bad local maximums. We also examined the scalability of POND with regard to the size of training data. As shown in Fig. 2d, under all the rank settings, the average per-epoch running time grows linearly with the training ratio. Therefore, POND enjoys a linear scalability, consistent with our algorithm complexity (see Section IV-C).

We then evaluated all the methods on the remaining datasets. For Climate, we varied the rank from {3, 5, 10, 20}, while for the checkin datasets Gowalla and SG, we varied the rank form {5, 10, 20, 60, 80}. Note that in the latter case, we tested all the methods with quite large ranks. On each dataset, we randomly split the observed entries into 5 folds, and used 4 folds for training and the remaining fold for testing. We repeated the test for 5 times, and reported the average RMSE, MAE and their standard deviations in Table II and III. As we can see, in almost all the cases, POND significantly outperforms all the competing approaches in both RMSE and MAE (p-value<0.05). The results further confirm the advantages of our method in prediction accuracy. We also compared the speeds of all the methods. We ran all the algorithms on a desktop machine with Intel i9-9900K CPU and 32GB memory. The speed of POND is comparable to CoSTco, DFNT and faster than GPTF. For example, on Gowalla dataset and rank = 20, their running time (seconds) are {POND: 2884, DFNT: 2521, CoSTco: 2719, GPTF: 4998}. By contrast, the multilinear methods are much faster, {CP-WOPT: 6, CP-ALS: 18, P-Tucker: 514}. This is reasonable, because the nonlinear methods are much more complex and need to estimate many more parameters.

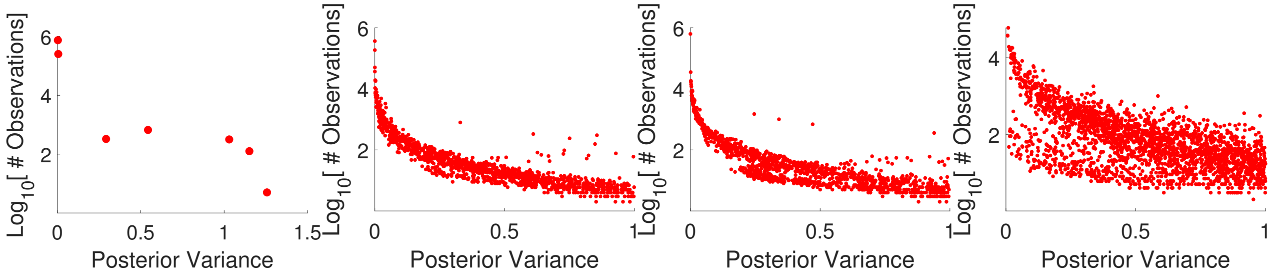
C. Uncertainty Quantification

Finally, we looked into the uncertainty information provided by POND, and discussed their usage in practical applications. To this end, we used a real-world mobile ads click-through-rate dataset (www.kaggle.com/c/avazu-ctr-prediction/data) in a Kaggle competition task. The dataset record 10 days of ads impressions and their clicks from an online advertising system. From this dataset, we extracted a four-mode binary tensor, *(banner pos, site, app, device)*, from the first one million records. The value of each entry indicates if the corresponding mobile advertisement was clicked or not. The tensor is of size $7 \times 2854 \times 4114 \times 6061$. Among the observed entries are 174K nonzero values, *i.e.,* clicks, which is 17.4%. Therefore, most of the ads displayed did not receive any click. We then used POND



(a) 3% (b) 5% (c) 10% (d) POND Scalability

Fig. 2: Predictive performance on MovingMNIST with different ratios of the observed entries for training (a-c) and the scalability of POND (d). The prediction accuracy was averaged from 5 runs.



(a) *banner position* (b) *site* (c) *app* (d) *device*

Fig. 3: The posterior variance of the learned latent factors *v.s.* the frequency of the corresponding objects in the observed entries.

TABLE II: Tensor completion accuracy on Climate

metric	method/rank	3	5	10	20
RMSE	CP-ALS	0.7904 ± 0.0022	0.7904 ± 0.0022	0.7904 ± 0.0022	0.7904 ± 0.0022
	CP-WOPT	2.3604 ± 0.1462	3.3917 ± 0.1670	6.0489 ± 0.2027	1.8680 ± 0.0179
	P-Tucker	0.1038 ± 0.0046	0.1496 ± 0.0147	0.1731 ± 0.0029	0.2632 ± 0.0049
	DFNT	0.1412 ± 0.0014	0.4534 ± 0.0042	0.7900 ± 0.0021	0.7900 ± 0.0021
	CoSTco	0.0842 ± 0.0009	0.0849 ± 0.0009	0.0839 ± 0.0009	$\mathbf{0.0833 \pm 0.0011}$
	GPTF	0.0916 ± 0.0016	0.0969 ± 0.0015	0.969 ± 0.0014	0.0938 ± 0.0016
	POND	$\mathbf{0.0829 \pm 0.0012}$	$\mathbf{0.0827 \pm 0.0012}$	$\mathbf{0.0837 \pm 0.0013}$	0.0847 ± 0.0012
MAE	CP-ALS	0.7369 ± 0.0026	0.7369 ± 0.0026	0.7369 ± 0.0025	0.7369 ± 0.0025
	CP-WOPT	1.0552 ± 0.0136	1.3527 ± 0.0117	2.4118 ± 0.0225	1.3271 ± 0.0091
	P-Tucker	0.0601 ± 0.0014	0.0831 ± 0.0066	0.1116 ± 0.0023	0.1961 ± 0.0035
	DFNT	0.0974 ± 0.0019	0.3865 ± 0.0048	0.7369 ± 0.0023	0.7369 ± 0.0023
	CoSTco	0.0508 ± 0.0006	0.0514 ± 0.0006	0.0505 ± 0.0006	0.0498 ± 0.0006
	GPTF	0.0581 ± 0.0012	0.0621 ± 0.0010	0.0621 ± 0.0014	0.0597 ± 0.0011
	POND	$\mathbf{0.0491 \pm 0.0007}$	$\mathbf{0.0492 \pm 0.0006}$	$\mathbf{0.0495 \pm 0.0007}$	$\mathbf{0.0497 \pm 0.0007}$

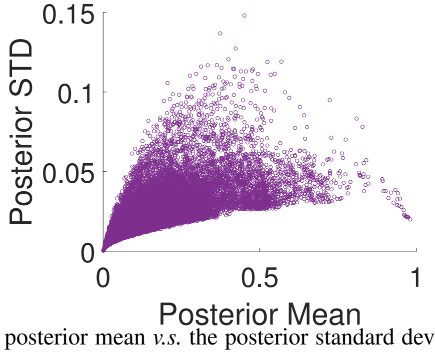


Fig. 4: The posterior mean *v.s.* the posterior standard deviation (STD) of the click probability prediction.

to estimate the latent factors and their posterior distribution. To this end, we modified the likelihood for each observed entry $\mathbf{i}_n$ to the Binomial likelihood used in Probit regression [33](see (7) and (8)), $p\big(\mathbf{y}_{\mathbf{i}_n}|f(\mathbf{x}_{\mathbf{i}_n})\big) = \Phi\big((2y_{\mathbf{i}_n}-1)f(\mathbf{x}_{\mathbf{i}_n})\big)$ where $\Phi(\cdot)$ is the cumulative density function (CDF) of the standard normal distribution. We followed the same approach as presented in Section IV to fulfill stochastic variational learning. For convenient analysis of the posterior variance, we set the number of latent factors to 1. In Figure 3, we show how the posterior variances of the latent factors are distributed according to the count of the corresponding objects in the observed tensor entries. As we can see, in each mode, the more frequently an object (*e.g.,* a particular banner position or site) appears in the ads impressions, the smaller the variance and so the more certain its factor estimation. This is reasonable because active nodes appear in many training samples, and their learning results should be confident. Note that the posterior variance of these latent factors mostly reside in [0, 1], because the prior variance of the latent factors is set to 1 (see (8)). When data are observed, the variance (*i.e.,* uncertainty) tends to decrease.

Next, we examined the uncertainty of the click probability (*i.e.,* $c_{\mathbf{i}} = p(y_{\mathbf{i}} = 1) = \Phi\big(f(\mathbf{x}_{\mathbf{i}})\big)$ for a test entry $\mathbf{i}$). To this end,

(a) MAE

data	method/rank	5	10	20	60	80
Gowalla	CP-ALS	0.1454 ± 0.0001	0.1449 ± 0.0002	0.1434 ± 0.0002	0.1417 ± 0.0001	0.1413 ± 0.0001
	CP-WOPT	0.2658 ± 0.0052	0.2319 ± 0.0038	0.1762 ± 0.0011	0.1485 ± 0.0004	0.1484 ± 0.0003
	P-Tucker	0.1331 ± 0.0003	0.1642 ± 0.0006	0.1711 ± 0.0003	0.1452 ± 0.0003	0.1373 ± 0.0003
	DFNT	0.0931 ± 0.0003	0.1462 ± 0.0008	0.1478 ± 0.0001	0.1479 ± 0.0001	0.1479 ± 0.0001
	CoSTco	0.0837 ± 0.0009	0.0822 ± 0.0007	0.0841 ± 0.0002	0.0822 ± 0.0004	0.0811 ± 0.0003
	GPTF	0.0999 ± 0.0005	0.1131 ± 0.0001	0.1040 ± 0.0004	0.0718 ± 0.0002	0.0722 ± 0.0002
	POND	**0.0757 ± 0.0001**	**0.0753 ± 0.0001**	**0.06893 ± 0.0002**	**0.0674 ± 0.0006**	**0.0671 ± 0.0007**
SG	CP-ALS	0.1586 ± 0.0003	0.1575 ± 0.0004	0.1567 ± 0.0005	0.1581 ± 0.0005	0.1588 ± 0.0005
	CP-WOPT	0.5057 ± 0.0253	0.7328 ± 0.0303	0.6109 ± 0.0073	0.1299 ± 0.0005	0.1270 ± 0.0007
	P-Tucker	0.1503 ± 0.0014	0.1866 ± 0.0015	0.2116 ± 0.0002	0.1795 ± 0.0022	0.1922 ± 0.0004
	DFNT	0.1054 ± 0.0004	0.1606 ± 0.0003	0.1607 ± 0.0003	0.1607 ± 0.0003	0.1607 ± 0.0003
	CoSTco	**0.0898 ± 0.0014**	0.0963 ± 0.0012	0.0990 ± 0.0007	0.0948 ± 0.0012	0.0974 ± 0.0015
	GPTF	0.1207 ± 0.0004	0.1261 ± 0.0008	0.1236 ± 0.0006	0.1017 ± 0.0004	0.1607 ± 0.0003
	POND	0.0916 ± 0.0005	**0.0899 ± 0.0014**	**0.0848 ± 0.0011**	**0.0917 ± 0.0002**	**0.0908 ± 0.0007**

(b) RMSE

data	method/rank	5	10	20	60	80
Gowalla	CP-ALS	0.1975 ± 0.0003	0.1973 ± 0.0004	0.1956 ± 0.0003	0.1938 ± 0.0003	0.1935 ± 0.0002
	CP-WOPT	1.7328 ± 0.1778	0.7209 ± 0.0785	0.3085 ± 0.0074	0.2019 ± 0.0003	0.2015 ± 0.0004
	P-Tucker	0.3575 ± 0.0011	0.3264 ± 0.0021	0.2904 ± 0.0012	0.2416 ± 0.0004	0.2294 ± 0.0013
	DFNT	0.1614 ± 0.0004	0.1988 ± 0.0007	0.2000 ± 0.0003	0.2001 ± 0.0002	0.2001 ± 0.0002
	CoSTco	0.1808 ± 0.0013	0.17590 ± 0.0017	0.1792 ± 0.0002	0.1691 ± 0.0004	0.1625 ± 0.0005
	GPTF	0.1958 ± 0.0007	0.1958 ± 0.0002	0.1736 ± 0.0004	**0.1303 ± 0.0002**	**0.1305 ± 0.0003**
	POND	**0.1346 ± 0.0003**	**0.1347 ± 0.0003**	**0.1320 ± 0.0006**	0.1314 ± 0.0003	0.1317 ± 0.0003
SG	CP-ALS	0.2228 ± 0.0004	0.2209 ± 0.0006	0.2201 ± 0.0008	0.2244 ± 0.0008	0.2261 ± 0.0008
	CP-WOPT	4.5058 ± 0.6257	3.5519 ± 0.3335	1.4935 ± 0.0361	0.2081 ± 0.0007	0.2024 ± 0.0008
	P-Tucker	0.3558 ± 0.0132	0.4952 ± 0.1247	0.3514 ± 0.0006	0.3311 ± 0.0137	0.3296 ± 0.0014
	DFNT	0.1858 ± 0.0004	0.2256 ± 0.0004	0.2256 ± 0.0004	0.2256 ± 0.0004	0.2256 ± 0.0004
	CoSTco	0.1835 ± 0.0027	0.1977 ± 0.0012	0.2017 ± 0.0009	0.1947 ± 0.0016	0.2005 ± 0.0012
	GPTF	0.2129 ± 0.0008	0.2106 ± 0.0009	0.2100 ± 0.0012	0.1864 ± 0.0015	0.2257 ± 0.0004
	POND	**0.1585 ± 0.0003**	**0.1565 ± 0.0009**	**0.1537 ± 0.0003**	**0.1584 ± 0.0003**	**0.1574 ± 0.0007**

TABLE III: Tensor completion performance on Gowalla and SG

after using one million impressions to build the training tensor, we chose $100K$ impressions for a test set. We made sure the indices in each test entry are included in the training tensor. For each impression $\mathbf{i}$, we computed the predictive (posterior) mean and standard deviation (STD) of the click probability with Monte-Carlo approximation. The details are given in the appendix. We show the distribution of those posterior mean and STD pairs in Figure 4. It is interesting to see that when the mean estimation of the click probability is closer and closer to 0, the posterior STD becomes smaller and smaller, implying increased confidence. By contrast, when the click probability is predicted to be large, *i.e.,* close to 1, the STD becomes much larger, indicating much less confidence. This is reasonable, because most of the ads impressions do not have clicks. The clicked examples are quite scarce. Therefore, it is much more confident to predict that an ad will not be clicked than that the ad will be clicked. We also tested the prediction accuracy of POND, CoSTco and CP-ALS in terms of area under ROC curves (AUC), and obtained {POND: 0.740, CoSTco:0.738, CP-ALS: 0.596}. Hence, POND is a slightly better than CoSTco; both POND and CoSTco greatly outperform CP-ALS. The result confirms the advantage of nonlinear decomposition.

The uncertainty information about the prediction of the click probability can be very useful for ads ranking and display. For extreme predictions (close to 0 or 1) with high confidence levels, we can directly use them to determine showing or not showing the ads. However, when considerable uncertainty is present, we might combine the deterministic ranking results with randomly selected ads to increase the diversity in ads display and maintain freshness in the user experience. This is a trade-off between the exploration and exploitation [34], [35]. We want to utilize the prediction known to be good, but meanwhile we want to attract users with more diversity.

VII. Conclusion

We have presented POND, a probabilistic nonparametric tensor decomposition model equipped with neural kernels. Plenty of experiments have shown the advantage of POND in self-adapting to the data complexity and capturing highly nonlinear relationships. In the future, we will continue to apply POND in important problems, such as recommendation and online advertising, and in-depth investigate the usage of the latent factors and uncertainty information.

Acknowledgement

This work has been supported by the NSF IIS-1910983.

References

[1] L. Tucker, “Some mathematical notes on three-mode factor analysis,” *Psychometrika*, vol. 31, pp. 279–311, 1966.

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

[3] W. Chu and Z. Ghahramani, “Probabilistic models for incomplete multi-dimensional arrays,” *AISTATS*, 2009.

[4] U. Kang *et al.*, “Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries,” in *KDD*, 2012, pp. 316–324.

[5] J. H. Choi and S. Vishwanathan, “Dfacto: Distributed factorization of tensors,” in *Advances in Neural Information Processing Systems*, 2014, pp. 1296–1304.

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

[7] S. Zhe, Y. Qi, Y. Park, I. Molloy, and S. Chari, “Dintucker: Scaling up gaussian process models on multidimensional arrays with billions of elements,” *arXiv preprint arXiv:1311.2663*, 2013.

[8] S. Zhe, Z. Xu, X. Chu, Y. Qi, and Y. Park, “Scalable nonparametric multiway data analysis,” in *AISTATS*, 2015, pp. 1125–1134.

[9] S. Zhe, *et al.*, “Distributed flexible nonlinear tensor factorization,” in *Advances in Neural Information Processing Systems*, 2016, pp. 928–936.

[10] H. Liu, Y. Li, M. Tsang, and Y. Liu, “Costco: A neural tensor completion model for sparse tensors,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 324–334.

[11] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[12] J. Hensman, N. Fusi, and N. D. Lawrence, “Gaussian processes for big data,” in *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2013, pp. 282–290.

[13] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.

[14] T. G. Kolda, *Multilinear operators for higher-order decompositions*. United States. Department of Energy, 2006, vol. 2.

[15] M. J. Wainwright, M. I. Jordan *et al.*, “Graphical models, exponential families, and variational inference,” *Foundations and Trends® in Machine Learning*, vol. 1, no. 1–2, pp. 1–305, 2008.

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

[17] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Morup, “Scalable tensor factorizations for incomplete data,” *Chemometrics and Intelligent Laboratory Systems*, vol. 106, no. 1, pp. 41–56, 2011.

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

[19] W. Sun, J. Lu, H. Liu, and G. Cheng, “Provable sparse tensor decomposition,” *arXiv preprint arXiv:1502.01425*, 2015.

[20] Y. Du, Y. Zheng, K. Lee, and S. Zhe, “Probabilistic streaming tensor decomposition,” in *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 99–108.

[21] Z. Pan, Z. Wang, and S. Zhe, “Streaming nonlinear Bayesian tensor decomposition,” in *Conference on Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 490–499.

[22] B. Liu, L. He, Y. Li, S. Zhe, and Z. Xu, “Neuralcp: Bayesian multiway data analysis with neural tensor decomposition,” *Cognitive Computation*, vol. 10, no. 6, pp. 1051–1061, 2018.

[23] X. Wu, B. Shi, Y. Dong, C. Huang, and N. Chawla, “Neural tensor factorization,” *arXiv preprint arXiv:1802.04416*, 2018.

[24] M. Abadi, P. Barham *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.

[25] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[26] T. P. Minka, “Judging significance from error bars,” MIT, Tech. Rep., 2002.

[27] N. Srivastava, E. Mansimov, and R. Salakhudinov, “Unsupervised learning of video representations using lstms,” in *International conference on machine learning*, 2015, pp. 843–852.

[28] E. Racah, C. Beckham, T. Maharaj, S. E. Kahou, M. Prabhat, and C. Pal, “Extremeweather: A large-scale climate dataset for semi-supervised detection, localization, and understanding of extreme weather events,” in *Advances in Neural Information Processing Systems*, 2017, pp. 3402–3413.

[29] X. Li *et al.*, “Rank-geofm: A ranking based geographical factorization method for point of interest recommendation,” in *SIGIR*, 2015, pp. 433–442.

[30] Y. Liu, T.-A. N. Pham, G. Cong, and Q. Yuan, “An experimental evaluation of point-of-interest recommendation in location-based social networks,” 2017.

[31] B. W. Bader, T. G. Kolda *et al.*, “Matlab tensor toolbox version 2.6,” Available online, February 2015. [Online]. Available: http://www.sandia.gov/~tgkolda/TensorToolbox/

[32] S. Oh, N. Park, S. Lee, and U. Kang, “Scalable tucker factorization for sparse tensors-algorithms and discoveries,” in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 2018, pp. 1120–1131.

[33] T. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich, “Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft’s bing search engine.” Omnipress, 2010.

[34] S.-M. Li, M. Mahdian, and R. P. McAfee, “Value of learning in sponsored search auctions,” in *International Workshop on Internet and Network Economics*. Springer, 2010, pp. 294–305.

[35] W. Li *et al.*, “Exploitation and exploration in a performance based contextual advertising system,” in *KDD*, 2010, pp. 27–36.

Appendix

A. Posterior Mean and Variance of the Click Probability

In online advertising, we are interested in the uncertainty of the prediction of the click probabilities. Given the test impression $\mathbf{i}$ (corresponding to a test entry), we denote its click probability by $c_{\mathbf{i}}$. According to our model, $c_{\mathbf{i}} = p(y_{\mathbf{i}}) = \Phi\big(f(\mathbf{x}_{\mathbf{i}})\big)$, where $\mathbf{x}_{\mathbf{i}} = [(\mathbf{u}^1_{i_1})^\top, \ldots, (\mathbf{u}^K_{i_K})^\top]^\top$. We aim to compute the posterior mean and standard deviation (STD) of $c_{\mathbf{i}}$. To this end, we first compute $q\big(f(\mathbf{x}_{\mathbf{i}})\big)$, the (approximate) posterior distribution of the function value $f(\mathbf{x}_{\mathbf{i}})$. According to (12) and (16), we have

$$q\big(f(\mathbf{x}_{\mathbf{i}})\big) = \mathcal{N}\big(f(\mathbf{x}_{\mathbf{i}})|a_{\mathbf{i}}, g_{\mathbf{i}}\big) \tag{20}$$

where $a_{\mathbf{i}} = \mathbf{k}_n^\top \mathbf{K}_{ZZ}^{-1} \boldsymbol{\mu}$ and $g_{\mathbf{i}} = \mathbf{k}_n^\top \mathbf{K}_{ZZ}^{-1} \boldsymbol{\Sigma} \mathbf{K}_{ZZ}^{-1} \mathbf{k}_n + k(\mathbf{x}_{\mathbf{i}}, \mathbf{x}_{\mathbf{i}}) - \mathbf{k}_n^\top \mathbf{K}_{ZZ}^{-1} \mathbf{k}_n$. Then, conditioned on the latent factors in $\mathbf{x}_{\mathbf{i}}$, we have

$$\mathbb{E}[c_{\mathbf{i}}|\mathbf{x}_{\mathbf{i}}] = \int \Phi(f_{\mathbf{i}}) \mathcal{N}\big(f_{\mathbf{i}}|a_{\mathbf{i}}, g_{\mathbf{i}}\big) \mathrm{d} f_{\mathbf{i}},$$
$$\mathbb{E}[c_{\mathbf{i}}^2|\mathbf{x}_{\mathbf{i}}] = \int \Phi(f_{\mathbf{i}})^2 \mathcal{N}\big(f_{\mathbf{i}}|a_{\mathbf{i}}, g_{\mathbf{i}}\big) \mathrm{d} f_{\mathbf{i}}.$$

It is easy to derive that $\mathbb{E}[c_{\mathbf{i}}|\mathbf{x}_{\mathbf{i}}] = \Phi(\frac{a_{\mathbf{i}}}{\sqrt{1+g_{\mathbf{i}}}})$. However, $\mathbb{E}[c_{\mathbf{i}}^2|\mathbf{x}_{\mathbf{i}}]$ does not have a closed form. To address this problem, we can use Gaussian-Hermite quadrature to compute an accurate approximation. Now, to integrate out $\mathbf{x}_{\mathbf{i}}$ with the posterior distribution of the latent factors, we use a Monte-Carlo approximation. We use the learned approximate posteriors $\{q(\mathbf{u}^1_{i_1}), \ldots, q(\mathbf{u}^K_{i_K})\}$ to generate T i.i.d. samples of $\mathbf{x}_{\mathbf{i}}$: $\{\tilde{\mathbf{x}}_{\mathbf{i}}^t\}_{t=1}^T$. This is straightforward, because each posterior is a diagonal Gaussian distribution. Then we have

$$\mathbb{E}[c_{\mathbf{i}}] \approx \frac{1}{T} \mathbb{E}[c_{\mathbf{i}}|\tilde{\mathbf{x}}_{\mathbf{i}}^t], \tag{21}$$
$$\mathbb{E}[c_{\mathbf{i}}^2] \approx \frac{1}{T} \mathbb{E}[c_{\mathbf{i}}^2|\tilde{\mathbf{x}}_{\mathbf{i}}^t]. \tag{22}$$

The posterior mean of the click probability is given in (21). The posterior variance is calculated by

$$\mathrm{Var}(c_{\mathbf{i}}) \approx \frac{1}{T} \mathbb{E}[c_{\mathbf{i}}^2|\tilde{\mathbf{x}}_{\mathbf{i}}^t] - \big(\frac{1}{T} \mathbb{E}[c_{\mathbf{i}}|\tilde{\mathbf{x}}_{\mathbf{i}}^t]\big)^2. \tag{23}$$

We can compute the posterior STD accordingly. Note that Gauss-Hermite quadrature is very accurate, we can view its result as the true value of $\mathbb{E}[c_{\mathbf{i}}^2|\tilde{\mathbf{x}}_{\mathbf{i}}^t]$. Then the variance in (23), although computed from a Monte-Carlo approximation, is always non-negative.