Probabilistic Streaming Tensor Decomposition

Yishuai Du[†], Yimin Zheng[†], Kuang-chih Lee[♯], Shandian Zhe[†]
University of Utah[†], Alibaba Group[♯]
{u0884588,u0887427}@utah.edu[†], zhe@cs.utah.edu[†], leekc307@gmail.com[♯]

***Abstract*—Tensor decomposition is a fundamental tool for multiway data analysis. While most decomposition algorithms operate a collection of static data and perform batch processes, many applications produce data in a streaming manner — every time a subset of entries are generated, and previously seen entries cannot be revisited. In such scenarios, traditional decomposition approaches will be inappropriate, because they cannot provide timely updates when new data come in, and they need to access the whole dataset many times for batch optimization.**

To address this issue, we propose POST, a PrObabilistic Streaming Tensor decomposition algorithm, which enables real-time updates and predictions upon receiving new tensor entries, and supports dynamic growth of all the modes. Compared with the state-of-the-art streaming decomposition approach MAST [1], POST is more flexible in that it can handle arbitrary orders of streaming entries, and hence is more widely applicable. In addition, as a Bayesian inference algorithm, POST can quantify the uncertainty of the latent embeddings via their posterior distributions, and the confidence levels of the missing entry value predictions. On several real-world datasets, POST exhibits better or comparable predictive performance than MAST and other static decomposition algorithms.

***Index Terms*—tensor data, streaming decomposition, posterior inference**

I. Introduction

Multiway data, represented by tensors, or multidimensional arrays, are common in real-world applications, such as online advertising and recommendations. For example, we can extract a four-mode tensor (*user, advertisement, page-section, site*) from online advertisement click logs. An important tool for multiway data analysis is tensor decomposition, where we estimate embedding vectors for the objects in each tensor mode (e.g., specific users and advertisements), and use these embeddings to recover the observed entry values and to predict missing values (i.e., tensor completion).

Typical tensor decomposition approaches are based on CANDECOMP/PARAFAC (CP) decomposition, and use alternating least squares (ALS) [2] or gradient-based optimization [3] to update the embeddings. Despite the wide success, these approaches need to operate on static data and repeatedly access the whole dataset to perform alternative updates or to calculate the gradient. However, practical applications often produce data dynamically and incrementally. That is, each time only several or a set of tensor entries are generated, implying dynamic growth of the tensor in each mode. Moreover, in transient applications such as in Snapchat or Instagram, data are not allowed to be stored after being accessed. Hence, we cannot retrieve the previously seen entries, and need to immediately update the embeddings upon receiving new data increments. Obviously, in this scenario, traditional batch decomposition approaches will become inappropriate.

Recently, several seminal works were proposed to adapt to the dynamic growth of one tensor mode [4], [5], e.g., the time mode. Furthermore, a more powerful streaming decomposition approach, MAST [1], was developed to handle the streaming data, and can adapt to the growth of all the modes. However, these algorithms restrict the order of the streaming entries. That is, the newly coming entries must stay in the incremental portion of the tensor, and cannot belong to its previous, smaller version. Specifically, denote the tensor at step t and $t+1$ by $\mathcal{T}_t$ and $\mathcal{T}_{t+1}$, respectively; any entry received at $t+1$ must belong to $\mathcal{T}_{t+1} - \mathcal{T}_t$, rather than $\mathcal{T}_t$. However, in practice, both $\mathcal{T}_t$ and $\mathcal{T}_{t+1}$ are usually partially observed, and could have many missing entry values. It is often the case that some missing entry values in $\mathcal{T}_t$ are observed in later steps, say, $t+1$. Therefore, the restriction might hinder many applications, where the tensor entries stream in arbitrary orders.

To address this issue, we propose POST, a probabilistic streaming tensor decomposition algorithm, which is flexible to handle streaming entries in arbitrary orders. POST updates the latent embeddings upon receiving new tensor entries (without retrieving previous ones), and naturally supports dynamic increases of the objects in each tensor mode. Besides, as a probabilistic approach, POST simultaneously adjusts the uncertainty quantification of the embeddings, and provides the confidence level for missing value prediction.

Specifically, we first design a simple Bayesian CP decomposition model, based on which, we develop a streaming variational inference algorithm that recursively updates the posterior distributions of the latent embeddings. The algorithm starts with the prior distribution of the embeddings, and calculates the variational posterior distribution of the embeddings upon receiving observed entries. The posterior is then severed as the prior distribution, and integrated with the next set of entries to obtain the updated variational posterior. The variational updates are analytical and efficient. The procedure continues until all the entries are received. At any time, the current posterior summarizes the decomposition results over the data that have been seen so far, but does not maintain a fixed decomposition structure (as in MAST). Hence, afterwards our algorithm can still accept and process entries that belong to the previous tensor. Furthermore, the posterior distribution contains valuable uncertainty information, and can benefit data analysis, ranking, debugging, decision making, etc. With the posterior, we can calculate the confidence levels of the prediction for missing entry values, which correspond to many important tasks, such as click-through-rate prediction and commodity recommendation. The confidence estimations are useful for subsequent tasks, such as ads displaying [6], and recommendation results diversification [7].

For evaluation, we examined POST on several real-world multiway datasets. We first followed the requirement of MAST to ensure the entries received each time always incurred the expansion of the whole tensor. Under different rank settings for the latent embeddings, POST nearly always outperforms MAST in missing value prediction, and is often comparable to several static decomposition approaches. Then we evaluated POST with two large sparse tensors, where the observed entries streamed in completely random orders. POST achieves comparable or better predictive performance than the scalable static decomposition algorithm, CP-WOPT [8]. Finally, we analyzed the uncertainty information provided by POST in the online advertising application. We discussed the potential usage of these information, such as diagnosis of CTR prediction results and improving the ads display and user experience.

II. Preliminaries

We first introduce the notations and background knowledge. Throughout the paper, scalars are denoted 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 $\mathcal{Y} \in \mathbb{R}^{d_1 \times \ldots \times d_K}$, where d_k is the length of k-th mode, corresponding to d_k objects (e.g., users or items). The entry value at location $\mathbf{i} = (i_1, \ldots, i_K)$ is denoted by $y_{\mathbf{i}}$. The tensor $\mathcal{Y}$ can be flatten into a vector, denoted by $\text{vec}(\mathcal{Y})$, where the entry $\mathbf{i} = (i_1, \ldots, i_K)$ in $\mathcal{Y}$ is mapped to the entry at position $j = i_K + \sum_{k=1}^{K-1}(i_k - 1)\prod_{t=k+1}^{K} d_t$ in $\text{vec}(\mathcal{Y})$.

To perform tensor decomposition, we first introduce a set of latent embedding vectors to represent the tensor objects. Specifically, each object j in mode k is represented by an R dimensional vector $\mathbf{u}_j^k$. A $d_k \times R$ embedding matrix can then be constructed by stacking all the embedding vectors in mode k, $\mathbf{U}^k = [\mathbf{u}_1^k, \ldots, \mathbf{u}_{d_k}^k]^\top$. Given the embedding matrices of all the modes, $\mathcal{U} = \{\mathbf{U}^1, \ldots, \mathbf{U}^K\}$, tensor decomposition aims to use these embeddings to reconstruct the observed entries.

The CANDECOMP/PARAFAC (CP) decomposition [9] is the most popular tensor decomposition approach. Given the tensor $\mathcal{Y}$, CP assumes that $\mathcal{Y} \approx [\![\mathbf{U}^1, \ldots, \mathbf{U}^K]\!]$, where $[\![\cdot]\!]$ is the Kruskal operator [10]. This is equivalent to

$$\text{vec}(\mathcal{Y}) \approx \sum_{r=1}^{R} \mathbf{U}^1(:, r) \otimes \ldots \otimes \mathbf{U}^K(:, r)$$

where $\otimes$ is the Kronecker product, and $\mathbf{U}^k(:, r)$ is the r-th column of $\mathbf{U}^k$ $(1 \le k \le K)$. From the vectorized form, we can easily obtain that for each observed entry $\mathbf{i}$,

$$y_{\mathbf{i}} \approx \sum_{r=1}^{R}\prod_{k=1}^{K} u_{i_k,r}^k = \mathbf{1}^\top(\mathbf{u}_{i_1}^1 \circ \ldots \circ \mathbf{u}_{i_K}^K) \quad (1)$$

where $\mathbf{1}$ is the vector full of ones, $\circ$ is the Hadamard product that performs element-wise multiplication.

To identify the optimal embeddings, CP decomposition minimizes a mean square loss,

$$L(\mathcal{U}) = \frac{1}{2}\|\text{vec}(\mathcal{Y}) - \sum_{r=1}^{R} \mathbf{U}^1(:, r) \otimes \ldots \otimes \mathbf{U}^K(:, r)\|^2.$$

We can use alternating least squares [2], namely, alternatively updating each embedding matrix given all the other fixed, or gradient-based approaches [3], e.g., nonlinear conjugate gradient descent or L-BFGS, to minimize the loss function.

III. Probabilistic Tensor Decomposition

The standard CP decomposition finds the point estimation of the embeddings only and is unable to evaluate the uncertainty of these estimations. However, the uncertainty may vary significantly among the tensor objects. For example, if an object frequently interacts with objects in other modes, our estimation of its embeddings should be much more certain than ones having scarce interactions. Furthermore, the prediction w.r.t. these active objects should have high confidence levels, and those inactive objects low levels. The confidence levels can be used to analyze the data, debug/improve the predictive model, and optimize subsequent decisions, such as determining the web ads showing [6] and commodity ranking for recommendation [7]. In addition, the least mean square objective of the standard CP decomposition implicitly assumes the entry values are continuous, and ignore the possible heterogeneity in practical applications. For examples, many tensors are binary, say, for online advertising and social links.

To overcome these problems, we instead introduce a Bayesian generative model that is a probabilistic version of the CP decomposition. The Bayesian model enables the uncertainty quantification in a principled, posterior inference framework. The data heterogeneity can be naturally accounted for via different likelihoods. Specifically, we first sample the embedding vectors from a Gaussian prior,

$$p(\mathcal{U}) = \prod_{k=1}^{K}\prod_{s=1}^{d_k} \mathcal{N}(\mathbf{u}_s^k | \mathbf{m}_s^k, v\mathbf{I}) \quad (2)$$

where $\mathbf{m}_s^k$ is the mean, and v is a scalar, which controls the flatness of the Gaussian prior. The bigger v, the more uninformative the prior. Given the embeddings, the value of each observed entry $\mathbf{i}$ is sampled from a noisy data distribution $p(y_{\mathbf{i}}|\mathcal{U})$. Here, we consider two value types, continuous and binary. For continuous entry values, we use the Gaussian likelihood,

$$p(y_{\mathbf{i}}|\mathcal{U}) = \mathcal{N}(y_{\mathbf{i}} | \mathbf{1}^\top(\mathbf{u}_{i_1}^1 \circ \ldots \circ \mathbf{u}_{i_K}^K), \tau^{-1}) \quad (3)$$

where τ is the inverse variance. We further assign a conjugate, Gamma prior over τ,

$$p(\tau|a_0, b_0) = \text{Gam}(\tau|a_0, b_0) = \frac{b_0^{a_0}}{\Gamma(a_0)}\tau^{a_0-1}e^{-b_0\tau}. \quad (4)$$

For binary entries, we use the Probit likelihood,

$$p(y_{\mathbf{i}}|\mathcal{U}) = \phi\big((2y_{\mathbf{i}} - 1)\mathbf{1}^\top(\mathbf{u}_{i_1}^1 \circ \ldots \circ \mathbf{u}_{i_K}^K)\big) \quad (5)$$

where $\phi(\cdot)$ is the cumulative density function (CDF) of the standard normal distribution, $\phi(x) = \int_{-\infty}^{x} \mathcal{N}(x|0,1)\mathrm{d}x$. Denote the set of observed entries by S. The joint probability of our Bayesian decomposition model for continuous data, according to (2)(3)(4), is given by

$$p(\{y_{\mathbf{i}}\}_{\mathbf{i}\in S}, \mathcal{U}, \tau) = \mathrm{Gam}(\tau|a_0, b_0) \prod_{k=1}^{K} \prod_{s=1}^{d_k} \mathcal{N}(\mathbf{u}_s^k | \mathbf{m}_s^k, v\mathbf{I}) \cdot \prod_{\mathbf{i}\in S} \mathcal{N}(y_{\mathbf{i}} | \mathbf{1}^\top (\mathbf{u}_{i_1}^1 \circ \ldots \circ \mathbf{u}_{i_K}^K), \tau^{-1}), \quad (6)$$

and for binary data, according to (2)(5), is given by

$$p(\{y_{\mathbf{i}}\}_{\mathbf{i}\in S}, \mathcal{U}) = \prod_{k=1}^{K} \prod_{s=1}^{d_k} \mathcal{N}(\mathbf{u}_s^k | \mathbf{m}_s^k, v\mathbf{I}) \cdot \prod_{\mathbf{i}\in S} \phi\big((2y_{\mathbf{i}} - 1)\mathbf{1}^\top (\mathbf{u}_{i_1}^1 \circ \ldots \circ \mathbf{u}_{i_K}^K)\big). \quad (7)$$

IV. Streaming Posterior Inference

Now, we present POST, our posterior inference algorithm for streaming data, based on the streaming variational inference framework [11] . The observed tensor entries are provided in a series of batches, $\{S_1, S_2, \ldots\}$. Note that different batches do not necessarily have the same number of entries. We aim to update the posterior distribution of the latent embeddings upon receiving each batch of entries S_t, without using the previously seen batches $\{S_1, \ldots, S_{t-1}\}$.

A. Continuous Tensor

Let us first consider the continuous entry values. Denote by $\mathcal{D}_t$ all the observed data before we see the batch S_t, namely, $\mathcal{D}_t = \{y_{\mathbf{i}}\}_{\mathbf{i}\in S_1 \cup \ldots \cup S_{t-1}}$. From the joint probability in (6), we can easily observe that

$$p(\mathcal{D}_t \cup \{y_{\mathbf{i}}\}_{\mathbf{i}\in S_t}, \mathcal{U}, \tau) = p(\mathcal{D}_t, \mathcal{U}, \tau) p(\{y_{\mathbf{i}}\}_{\mathbf{i}\in S_t} | \mathcal{U}, \tau). \quad (8)$$

Dividing both sides of (8) by the marginal probability $p(\mathcal{D}_t \cup \{y_{\mathbf{i}}\}_{\mathbf{i}\in S_t})$, we can obtain

$$p(\mathcal{U}, \tau | \mathcal{D}_t \cup \{y_{\mathbf{i}}\}_{\mathbf{i}\in S_t}) \propto p(\mathcal{U}, \tau | \mathcal{D}_t) p(\{y_{\mathbf{i}}\}_{\mathbf{i}\in S_t} | \mathcal{U}, \tau). \quad (9)$$

As we can see, the updated posterior is proportional to the current posterior multiplying with the likelihood of the received data batch. Hence, we can use a recursive process for the streaming inference. At the beginning, the posterior of $\mathcal{U}$ and τ is assigned to their prior (2) and (4). Each time when we receive a new batch of entries S_t, we combine the current posterior, which is now served as the prior, with the likelihood of the observed entry values in S_t, to obtain the updated posterior.

However, the computation of the posterior update is intractable, because the likelihood (3) intertwines the embeddings vectors. Hence, we resort to variational approximations for tractable inference [12]. Specifically, we introduce a factorized variational posterior distribution,

$$q(\mathcal{U}, \tau) = q(\tau) \prod_{k=1}^{K} \prod_{s=1}^{d_k} q(\mathbf{u}_s^k), \quad (10)$$

to approximate $p(\mathcal{U}, \tau | \mathcal{D}_t)$ in (9). Each time, we use $q(\mathcal{U}, \tau)$ to replace $p(\mathcal{U}, \tau | \mathcal{D}_t)$, and combine with the data likelihood $p(\{y_{\mathbf{i}}\}_{\mathbf{i}\in S_t} | \mathcal{U}, \tau)$ to calculate the best approximation for $p(\mathcal{U}, \tau | \mathcal{D}_{\mathbf{i}} \cup \{y_{\mathbf{i}}\}_{\mathbf{i}\in S_t})$, namely, $q^*(\mathcal{U}, \tau)$. Then $q^*(\mathcal{U}, \tau)$ is assigned back to $q(\mathcal{U}, \tau)$, and combined with the likelihood of the next batch S_{t+1} to update the approximate posterior again. We repeat this process until we finish processing all the tensor entries. To obtain $q^*(\mathcal{U}, \tau)$, we use the variational inference framework [12] — we minimize the Kullback Leibler (KL) divergence between $q^*(\mathcal{U}, \tau)$ and $\frac{1}{C} p(\{y_{\mathbf{i}}\}_{\mathbf{i}\in S_t} | \mathcal{U}, \tau) q(\mathcal{U}, \tau)$ where C is the normalization const. This is equivalent to maximizing the following variational model evidence lower bound [12],

$$\mathcal{L} = \int q^*(\mathcal{U}, \tau) \log \frac{p(\{y_{\mathbf{i}}\}_{\mathbf{i}\in S_t} | \mathcal{U}, \tau) q(\mathcal{U}, \tau)}{q^*(\mathcal{U}, \tau)} \mathrm{d}\mathcal{U}\mathrm{d}\tau. \quad (11)$$

Due to the factorized form (10), we can optimize $q^*(\mathcal{U}, \tau)$ with efficient close-form updates. Specifically, before we see any data, we initialize $q(\mathcal{U}, \tau)$ with the prior distribution (see (2) (4)). As we will see, after each update, $q(\mathcal{U}, \tau)$ will keep the factorized form identical to the prior, namely, a Gamma distribution multiplying with (multivariate) Gaussian distributions for the latent embedding vectors of all the tensor objects. Assume currently, $q(\mathcal{U}, \tau)$ is parameterized by

$$q(\mathcal{U}, \tau) = \mathrm{Gamma}(\tau | a, b) \prod_{k=1}^{K} \prod_{s=1}^{d_k} \mathcal{N}(\mathbf{u}_s^k | \boldsymbol{\mu}_s^k, \boldsymbol{\Sigma}_s^k). \quad (12)$$

In maximizing the variational bound (11) w.r.t $q^*(\mathcal{U}, \tau) = q^*(\tau) \prod_{k=1}^{K} \prod_{s=1}^{d_k} q^*(\mathbf{u}_s^k)$, we only need to update the posterior terms that associate with the batch S_t, namely, $q^*(\tau)$ and $\{\{q^*(\mathbf{u}_{i_k}^k)\}_{k=1}^K\}_{\mathbf{i}\in S_t}$. For any other term $q^*(\mathbf{u}_{j_k}^k)(\mathbf{j} \notin S_t, 1 \leq k \leq K)$, because it does not connect to the data likelihood, the optimal result will simply be the corresponding term in $q(\mathcal{U}, \tau)$, i.e., $\mathcal{N}(\mathbf{u}_{j_k}^k | \boldsymbol{\mu}_{j_k}^k, \boldsymbol{\Sigma}_{j_k}^k)$. Therefore, we only need to focus on a small subset of embeddings related to the current data batch, and hence can save much computation.

We can alternatively update $q^*(\tau)$ and each $q^*(\mathbf{u}_{i_k}^k)(\mathbf{i} \in S_t)$ — each time, we optimize one while fixing the others. By setting the functional derivative of $\mathcal{L}$ in (11) w.r.t to each posterior to 0, we can derive the following close-form updates,

$$q^*(\mathbf{u}_{i_k}^k) = \mathcal{N}(\mathbf{u}_{i_k}^k | \boldsymbol{\mu}_{i_k}^{k\,*}, \boldsymbol{\Sigma}_{i_k}^{k\,*}), \quad (13)$$

$$q^*(\tau) = \mathrm{Gam}(\tau | a^*, b^*). \quad (14)$$

As we can see, the updated posteriors maintain the same form as the priors. The sufficient statistics are given by

$$\boldsymbol{\Sigma}_{i_k}^{k\,*} = \big({\boldsymbol{\Sigma}_{i_k}^{k}}^{-1} + \langle\tau\rangle \textstyle\sum_{\mathbf{j}\in S_t, j_k = i_k} \langle \mathbf{t}_{\mathbf{j},\neg k} \mathbf{t}_{\mathbf{j},\neg k}^\top \rangle\big)^{-1}, \quad (15)$$

$$\boldsymbol{\mu}_{i_k}^{k\,*} = \boldsymbol{\Sigma}_{i_k}^{k\,*} ({\boldsymbol{\Sigma}_{i_k}^{k}}^{-1} \boldsymbol{\mu}_{i_k}^k + \langle\tau\rangle \textstyle\sum_{\mathbf{j}\in S_t, j_k = i_k} y_{\mathbf{j}} \langle \mathbf{t}_{\mathbf{j},\neg k} \rangle), \quad (16)$$

$$a^* = a + \frac{1}{2}|S_t|, \quad (17)$$

$$b^* = b + \frac{1}{2} \textstyle\sum_{\mathbf{i}\in S_t} \big[y_{\mathbf{i}}^2 - 2y_{\mathbf{i}} \langle \mathbf{1}^\top \mathbf{t}_{\mathbf{i}} \rangle + \langle (\mathbf{1}^\top \mathbf{t}_{\mathbf{i}})^2 \rangle\big], \quad (18)$$

where $\langle\cdot\rangle$ is the expectation w.r.t the other posteriors, $|\cdot|$ is the size of the batch, $\langle\tau\rangle = \frac{a^*}{b^*}$,

$$\mathbf{t}_\mathbf{i} = \mathbf{u}^1_{i_1} \circ \ldots \circ \mathbf{u}^K_{i_K},$$
$$\mathbf{t}_{\mathbf{j},\neg k} = \mathbf{u}^1_{j_1} \circ \ldots \circ \mathbf{u}^{k-1}_{j_{k-1}} \circ \mathbf{u}^{k+1}_{j_{k+1}} \circ \ldots \circ \mathbf{u}^K_{j_K},$$

and

$$\begin{aligned}
\langle \mathbf{t}_{\mathbf{j},\neg k}\rangle &= \boldsymbol{\mu}^{1\,*}_{j_1} \circ \ldots \circ \boldsymbol{\mu}^{k-1\,*}_{j_{k-1}} \circ \boldsymbol{\mu}^{k+1\,*}_{j_{k+1}} \circ \ldots \circ \boldsymbol{\mu}^{K\,*}_{j_K},\\
\langle \mathbf{t}_{\mathbf{j},\neg k}\mathbf{t}_{\mathbf{j},\neg k}^\top\rangle &= \langle \mathbf{u}^1_{j_1}\mathbf{u}^{1\,\top}_{j_1}\rangle \circ \ldots \circ \langle \mathbf{u}^{k-1}_{j_{k-1}}\mathbf{u}^{k-1\,\top}_{j_{k-1}}\rangle\\
&\quad \circ \langle \mathbf{u}^{k+1}_{j_{k+1}}\mathbf{u}^{k+1\,\top}_{j_{k+1}}\rangle \circ \ldots \circ \langle \mathbf{u}^K_{j_K}\mathbf{u}^{K\,\top}_{j_K}\rangle\\
\langle \mathbf{1}^\top \mathbf{t}_\mathbf{i}\rangle &= \mathbf{1}^\top(\boldsymbol{\mu}^1_{i_1} \circ \ldots \circ \boldsymbol{\mu}^K_{i_K}),\\
\langle (\mathbf{1}^\top \mathbf{t}_\mathbf{i})^2\rangle &= \mathrm{tr}\big(\mathbf{1}\cdot\mathbf{1}^\top(\langle \mathbf{u}^1_{i_1}\mathbf{u}^{1\,\top}_{i_1}\rangle \circ \ldots \circ \langle \mathbf{u}^K_{i_K}\mathbf{u}^{K\,\top}_{j_K}\rangle)\big)
\end{aligned}$$

where for any object s in mode l, $\langle \mathbf{u}^l_s \mathbf{u}^{l\,\top}_s\rangle = \boldsymbol{\Sigma}^{l\,*}_s + \boldsymbol{\mu}^{l\,*}_s \boldsymbol{\mu}^{l\,*\top}_s (1 \le l \le K)$. The convergence is guaranteed by the following lemma.

Lemma IV.1. *The alternative updates among $q^*(\tau)$ and each $q^*(\mathbf{u}^k_{i_k})(\mathbf{i} \in S_t)$ according to* (13) (14) *will always converge to a stationary point of the variational bound $\mathcal{L}$ in* (11).

Proof. It is easy to show that

$$\mathcal{L} = -\mathrm{KL}\big(q^*(\mathcal{U},\tau)\|\frac{1}{C}p(\{y_\mathbf{i}\}_{\mathbf{i}\in S_t}|\mathcal{U},\tau)q(\mathcal{U},\tau)\big) + \log(C).$$

Since the KL divergence is always nonnegative, we have $\mathcal{L} \le \log(C)$. Each update in (13) or (14) will increase $\mathcal{L}$, hence $\mathcal{L}$ will converge to a local maximum. Since $\mathcal{L}$ is differentiable, the local maximum must be a stationary point. □

After convergence, we obtain the current posterior $q(\mathcal{U},\tau) = q^*(\mathcal{U},\tau)$, based on which we can process the next batch. The tensor can be expanded at any time — when some new objects join in (say, new users or items), we simply initialize their embedding posteriors with the prior distribution and update them according to the subsequent observed entries.

B. Binary Tensor

We use the same framework to perform streaming posterior inference for binary data, based on the model (7). However, due to the Probit likelihood (5), we cannot derive close-form updates for $q(\mathcal{U})$, even in the variational inference framework. To overcome this problem, we augment the Probit likelihood with a latent continuous random variable $z_\mathbf{i}$ for each observed entry $\mathbf{i}$, and we have

$$p(y_\mathbf{i}, z_\mathbf{i}|\mathcal{U},\tau) = \mathcal{N}\big(z_\mathbf{i}|\mathbf{1}^\top(\mathbf{u}^1_{i_1} \circ \ldots \circ \mathbf{u}^K_{i_K}),1\big)p(y_\mathbf{i}|z_\mathbf{i}) \quad (19)$$

where $p(y_\mathbf{i}|z_\mathbf{i}) = \mathbb{1}(y_\mathbf{i}=1)\mathbb{1}(z_\mathbf{i} \ge 0) + \mathbb{1}(y_\mathbf{i}=0)\mathbb{1}(z_\mathbf{i}<0)$ and $\mathbb{1}(\cdot)$ is the indicator function. It is easy to show that by marginalizing out $z_\mathbf{i}$ in (19), we can recover the original Probit likelihood (5).

Now, we follow the same framework as in Section IV-A. We use a factorized posterior distribution $q(\mathcal{U}) = \prod_{k=1}^K \prod_{s=1}^{d_k} q(\mathbf{u}^k_s)$. Given the current batch S_t, we introduce the latent variables $\mathbf{z} = \{z_\mathbf{i}\}_{\mathbf{i}\in S_t}$ for each entry, and derive a variational lower bound,

$$\mathcal{L} = \int q^*(\mathcal{U})q(\mathbf{z})\log\frac{p(\{z_\mathbf{i},y_\mathbf{i}\}_{\mathbf{i}\in S_t}|\mathcal{U},\tau)q(\mathcal{U})}{q^*(\mathcal{U})q(\mathbf{z})}\mathrm{d}\mathcal{U}\mathrm{d}\mathbf{z}. \quad (20)$$

We further use a factorized form for the posterior of $\mathbf{z}$, $q(\mathbf{z}) = \prod_{\mathbf{i}\in S_t} q(z_\mathbf{i})$. To maximize $\mathcal{L}$ in (20), we then alternatively update all the associated $q(\mathbf{u}^k_\mathbf{i})$ and each $q(z_\mathbf{i})$ in the batch. Now, we are able to derive analytical results. The update of each $q(z_\mathbf{i})$ is a truncated Gaussian distribution,

$$q(z_\mathbf{i}) \propto \mathcal{N}\big(z_\mathbf{i}|\langle\mathbf{1}^\top\mathbf{t}_\mathbf{i}\rangle,1\big)\mathbb{1}\big((2y_\mathbf{i}-1)z_\mathbf{i} \ge 0\big). \quad (21)$$

The update of each $q(\mathbf{u}_\mathbf{i})(\mathbf{i}\in S_t)$ is a Gaussian distribution, which is the same as in the continuous case (see (13)), except that we need to remove $\langle\tau\rangle$ in (15)(16), and replace $y_\mathbf{j}$ by $\langle z_\mathbf{j}\rangle$ in (16). We can calculate each $\langle z_\mathbf{i}\rangle$ through

$$\langle z_\mathbf{i}\rangle = \langle\mathbf{1}^\top\mathbf{t}_\mathbf{i}\rangle + \frac{(2y_\mathbf{i}-1)\mathcal{N}(\langle\mathbf{1}^\top\mathbf{t}_\mathbf{i}\rangle|0,1)}{\phi\big((2y_\mathbf{i}-1)\langle\mathbf{1}^\top\mathbf{t}_\mathbf{i}\rangle\big)}. \quad (22)$$

Apparently, the alternative updates guarantee to converge, following the same proof in Lemma IV.1.

The overall inference procedure, POST, is summarized in Algorithm 1. At any time, the variational posteriors of the embeddings, $\{q(\mathbf{u}^k_s)\}_{1\le k\le K}$, encode the decomposition results based on all the observed entries received so far. These Gaussian distributions include not only the embeddings estimations, but also the uncertainty information (reflected in the covariance matrices). Moreover, our algorithm is free to process streaming entries in arbitrary orders, because a new entry value can be modeled by the data likelihood at any time, and integrated into the posteriors through the variational updates presented above. In contrast, at each step, MAST uses a fixed decomposition to replace the entire tensor up to now (including the missing entries), and hence can only accept new entries which are outside the current tensor, i.e., the new entries must expand the whole tensor, and cannot belong to the current one. This actually restricts the order of the streaming data, and might deviate from the practical applications.

C. Computational Cost

The time complexity to update each $q(\mathbf{u}^k_s)$ is $\mathcal{O}(R^3)$, $q(\tau)$ for continuous data $\mathcal{O}(R^2)$, and each $q(z_\mathbf{i})$ for binary data $\mathcal{O}(R)$. Hence, the overall time complexity of POST at batch S_t is $\mathcal{O}(|S_t|KR^3)$. The space complexity is $\mathcal{O}(\sum_{k=1}^K d_k(R+R^2)+|S_t|K)$, which is to store the embedding posteriors, the data batch, the posterior $q(\tau)$ for continuous data and $q(\mathbf{z})$ for binary data.

D. Prediction

Since POST estimates the posterior distributions of the latent embeddings, we can hence integrate these posteriors to improve prediction, and to quantify the uncertainty. Specifically, suppose we want to predict $y_\mathbf{i}$, the value of a missing entry $\mathbf{i}$. The posterior predictive distribution of $y_\mathbf{i}$, for continuous tensor data, is given by

$$p(y_\mathbf{i}|D_t) = \int \mathcal{N}(y_\mathbf{i}|\mathbf{1}^\top\mathbf{t}_\mathbf{i},\tau^{-1})q(\tau)\prod\nolimits_k q(\mathbf{u}^k_{i_k})\mathrm{d}\tau\prod\nolimits_k \mathrm{d}\mathbf{u}^k_{i_k}$$

where D_t are all the entries observed so far, and, for binary data,

$$p(y_{\mathbf{i}}|D_t) = \int \phi\big((2y_{\mathbf{i}} - 1)\mathbf{1}^\top \mathbf{t}_{\mathbf{i}}\big) \prod\nolimits_k q(\mathbf{u}_{i_k}^k) \prod\nolimits_k \mathrm{d}\mathbf{u}_{i_k}^k.$$

Note that the predictive distribution integrates all the possible values of the embeddings (via their variational posteriors $\{q(\mathbf{u}_{i_k}^k)\}$), and hence can provide more robust and smooth predictions [13]. We can calculate the mean, for prediction, and variance for confidence level estimation. For continuous data, we can easily derive that

$$\mathrm{E}(y_{\mathbf{i}}) = \langle \mathbf{1}^\top \mathbf{t}_{\mathbf{i}} \rangle, \quad \mathrm{Var}(y_{\mathbf{i}}) = \langle (\mathbf{1}^\top \mathbf{t}_{\mathbf{i}})^2 \rangle + \frac{b}{a-1} - \mathrm{E}^2(y_{\mathbf{i}}).$$

The binary case is nontrivial, because the integration is not analytical. To overcome this problem, we consider the marginal distribution of $\mathbf{1}^\top \mathbf{t}_{\mathbf{i}}$. For brevity, we define $g_{\mathbf{i}} = \mathbf{1}^\top \mathbf{t}_{\mathbf{i}}$. Then equivalently, we have $p(y_{\mathbf{i}}|D_t) = \int \phi\big((2y_{\mathbf{i}} - 1)g_{\mathbf{i}}\big)p(g_{\mathbf{i}})\mathrm{d}g_{\mathbf{i}}$. We approximate $p(g_{\mathbf{i}})$ with a Gaussian distribution $\mathcal{N}\big(g_{\mathbf{i}}|\mathrm{E}(g_{\mathbf{i}}), \mathrm{Var}(g_{\mathbf{i}})\big)$. This is called moment matching, because $\mathrm{E}(g_{\mathbf{i}})$ and $\mathrm{Var}(g_{\mathbf{i}})$ correspond to the first and second moment of $g_{\mathbf{i}}$. Moment matching has been widely used and very successful in approximate Bayesian inference, such as expectation propagation [14]. Now the integration in $p(y_{\mathbf{i}}|D_t)$ is analytical, and we can obtain

$$\mathrm{E}(y_{\mathbf{i}}) = p(y_{\mathbf{i}} = 1|D_t) \approx \phi\Big(\frac{\mathrm{E}(g_{\mathbf{i}})}{\sqrt{1 + \mathrm{Var}(g_{\mathbf{i}})}}\Big), \tag{23}$$

$$\mathrm{Var}(y_{\mathbf{i}}) = \mathrm{E}(y_{\mathbf{i}}) - \mathrm{E}^2(y_{\mathbf{i}}). \tag{24}$$

In applications such as CTR prediction, besides the click action (i.e., $y_{\mathbf{i}}$), the uncertainty of the (click) probability, $\gamma_{\mathbf{i}} = \phi(g_{\mathbf{i}})$, is also highly interesting. Obviously, $\mathrm{E}(\gamma_{\mathbf{i}}) = \mathrm{E}(y_{\mathbf{i}})$. To calculate the variance, we need to compute $\langle \gamma_{\mathbf{i}}^2 \rangle = \int \phi(g_{\mathbf{i}})^2 \mathcal{N}\big(g_{\mathbf{i}}|\mathrm{E}(g_{\mathbf{i}}), \mathrm{Var}(g_{\mathbf{i}})\big)\mathrm{d}g_{\mathbf{i}}$, which is unfortunately intractable. However, we can use Gauss-Hermite quadrature to compute this integration very accurately. Then the variance of the probability is given by $\mathrm{Var}(\gamma_{\mathbf{i}}) = \langle \gamma_{\mathbf{i}}^2 \rangle - \mathrm{E}^2(\gamma_{\mathbf{i}})$.

V. Related Works

Many excellent works have been proposed for tensor decomposition, such as [8], [15]–[27]. Among these methods are quite a few Bayesian models, based on either CP [9] or Tucker decompositions [28]. For example, Xiong et. al. [15] developed a Bayesian temporal decomposition model for collaborative filtering. Zhao et. al. [25] placed an ARD prior over latent embeddings to determine the rank. Rai et. al. [20] used a multiplicative Gamma process (MGP) prior to automatically learn the rank of the embeddings. Hu et. al. [23] used Dirichlet prior to sample the latent embeddings and zero-truncated Poisson likelihoods for non-negative binary tensor decomposition. Schein et. al. [26] placed the Gamma prior over the embeddings and used the Tucker decomposition structure for international relation data analysis [27]. These methods perform batch or stochastic inference algorithms for pre-collected data, rather than for dynamic, streaming data.

Several incremental or dynamic tensor decomposition approaches have been developed to adapt to the dynamic growth of the tensor [1], [4], [5], [29], [30]. For example, Nion and Sidiropoulos [4] developed for three-mode tensors adaptive PARAFAC algorithms to update the decomposition with a new slice appended in the 'time' mode each step. Zhou et al. [5] proposed an accelerated online CP algorithm to track the decomposition for N-mode tensors. While the two methods focus on single-mode increasing data, Song et al. [1] proposed a streaming decomposition approach, MAST, which allows for the simultaneous expansion of all the modes. Each time, MAST uses a CP decomposition structure to replace the entire tensor so far. When there comes new tensor increments, along different modes or their combinations, MAST estimates a new CP decomposition, which jointly fits the observed entry values in the new tensor increments, and the existing (smaller) CP structure. Despite the elegance and the success, MAST still makes strong assumptions on the order of streaming data. Since the decomposition structure is fixed for the past tensor, MAST only receives new entries in the incremental portion. However, in practical applications, it is often possible that previous unseen entries are observed later. Hence, we proposed POST to handle streaming entries in arbitrary orders. Besides CP, several dynamic approaches were proposed based on Tucker decomposition and/or Higher-order SVD (HoSVD), such as [31]–[36]. Some of them, e.g., [34], supports the incremental updates along all the modes, like MAST, rather than a single mode [35], [36], but do not consider the missing entries.

Algorithm 1 The proposed streaming algorithm POST (R, v)

```
1: For continuous data, initialize $q(\tau)$ with an uninformative Gamma distribution, $q(\tau) = \mathrm{Gam}(\tau|10^{-3}, 10^{-3})$.
2: while A new set of tensor entries $S_t$ are received do
3:     For each brand-new object $s$ at each mode $k$, initialize $q(\mathbf{u}_s^k) = \mathcal{N}(\mathbf{u}_s^k|\mathbf{m}_s^k, v\mathbf{I})$ where $\mathbf{m}_s^k$ is an $R$ dimensional random vector.
4:     For binary data, introduce latent variables $\mathbf{z}$ for the entry values $\mathbf{y}$, and initialize $\langle \mathbf{z} \rangle$ with $2\mathbf{y} - 1$.
5:     repeat
6:         Update each $q^*(\mathbf{u}_{i_k}^k)$ associated with $S_t$, i.e., $\mathbf{i} \in S_t$ and $1 \le k \le K$, using (13)(15)(16). Note that for binary data, we need to remove $\langle \tau \rangle$ and replace $y_{\mathbf{j}}$ by $\langle z_{\mathbf{j}} \rangle$ in (15)(16).
7:         if Continuous data then
8:             Update $q^*(\tau)$ using (14)(17)(18).
9:         else
10:            Update $q^*(\mathbf{z})$ using (21).
11:        end if
12:    until Convergence or the maximum number of iterations have finished.
13:    Set each $q(\mathbf{u}_{i_k}^k) = q^*(\mathbf{u}_{i_k}^k)(\mathbf{i} \in S_t, 1 \le k \le K)$. For continuous data, set $q(\tau) = q^*(\tau)$.
14: end while
15: return The posteriors of the embedding vectors $\{q(\mathbf{u}_s^k) = \mathcal{N}(\mathbf{u}_s^k|\boldsymbol{\mu}_s^k, \boldsymbol{\Sigma}_s^k)\}(1 \le k \le K)$ for each object $s$ that have been seen.
```

Recently, a few nonlinear decomposition models were proposed to capture more complex relationships in tensor data [37]–[41], based on Bayesian nonparametrics and/or kernel methods. While powerful, these models usually have much higher computational costs. The streaming algorithms for these models remain an open and promising research direction.

VI. Experiment

To evaluate the proposed algorithm, POST, we conducted experiments to answer the following questions.

- **Q1**. How does POST compare with MAST, in the case that new entries always occur in tensor increments and increase the full tensor size?
- **Q2**. How does POST perform, when tensor entries stream in arbitrary orders, as compared with the traditional batch decomposition approaches?
- **Q3**. What does the uncertainty information produced by POST reveal and how does it potentially benefit practical applications, such as online advertising?

To answer the first question, we examined POST on two datasets, and followed [1] to generate dynamic tensor increments (Section VI-A). To answer the second question, we examined POST on two large datasets, under completely random orders of streaming entries (Section VI-B). To answer the third question, we looked into the means and variances of the embeddings and the predictions estimated by POST, on a real-world online advertisement clicks dataset (Section VI-C).

A. Evaluation on Dynamic Tensor Increments

We first examined POST on the following two datasets: (1) *Twitter Topic*[1], a three-mode $(user, expert, topic)$ tensor, of size $500 \times 500 \times 20$. The experts are referred to the producers of the high-quality content of 20 topics. Each entry value indicates whether to recommend some expert of a specific topic to a particular user. (2) *MovieLens*[2], a three-mode tensor of size $400 \times 400 \times 31$, describing the three-way $(user, movie, week)$ interactions. Each entry value represents whether some user rates a movie in a particular week. Hence, both *Twitter Topic* and *MovieLens* are binary tensors.

We compared with MAST and two static decomposition approaches, CP-ALS [8] that uses the traditional EM plus alternating least squares, and TNCP [42] that uses trace norm regularizations and an Alternating Direction Method of Multipliers (ADMM) algorithm for optimization. All the methods were implemented with MATLAB 2016, and we ran all the algorithms on a single Linux (Ubuntu) server with Intel(R) i7 CPUs and 24GB memory.

We followed a similar procedure to [1] to conduct the experiments. For each dataset, we started with a small subtensor, and produced new entries to increase its size at each step. In this way, MAST processes the dynamic tensor increments to performs streaming decomposition. Specifically, for *Twitter Topic*, we began with a $50 \times 50 \times 20$ subtensor, and increased 10 users and 10 experts at each step. For *MovieLens*, we started with a $40 \times 40 \times 31$ subtensor, and each time increased 10 users and 10 movies. To consider missing entries (which is common in practice), we randomly chose $\{50\%, 80\%, 90\%\}$ entries of the entire tensor as missing for each dataset, and used the remaining ones to generate the series of the tensor increments.

We started to run each decomposition method on the initial subtensor. The warm-start embeddings were given by the estimation of TNCP, as in [1]. At each step t, POST and MAST performed dynamic decomposition, upon receiving the observed entries in the tensor increment. CP-ALS and TNCP performed static decompositions on this new set of entries from scratch, where the embeddings for existing tensor objects were initialized with the previous estimation, and for new objects random values. Then the missing entries in the current tensor increment were used for testing, and the area under the ROC Curve (AUC) was calculated, denoted by AUC_t. We then calculated the running-average AUC as the overall performance measurement up to step t, RA-AUC $= \frac{1}{T}\sum_{t=1}^{T}\mathrm{AUC}_t$.

We varied the rank R, namely, the dimension of embedding vectors, from $\{3, 5, 8, 10\}$. The maximum number of iterations and the tolerance level were set to 500 and 10^{-5}, respectively, for all the methods. We followed [1] to set the optimal parameters for MAST. For our method, POST, we tuned v, the initial variance of the embeddings (see Algorithm 1), from $\{1, 3, 5, 10\}$ through extra validation procedures. For each particular missing rate and rank setting, we ran all the methods 5 times, where in each run the missing data entries were resampled to produce different observed entries in each tensor increment. The average of the RA-AUC was then calculated for all the methods.

The results at each step, for different choices of ranks and missing rates, are reported in Figure 1 for *Twitter Topic* and Figure 2 for *MovieLens*. As we can see, POST almost always outperforms MAST, except on *MovieLens* when 50% entries were missing and R was to set to 10 (Figure 2 l), their performance became close at the final stage. In addition, POST often exhibits better predictive performance than ALS and TNCP. On *Twitter Topic*, POST outperforms all the competing methods in most time. On *MovieLens*, POST tends to dominate the prediction accuracy when the missing rate is high (e.g., Figure 2a-c) or the rank is small (e.g., Figure 2a,e and i). The results demonstrate the advantage of POST using probabilistic/Bayesian frameworks for dynamic tensor decomposition and predictive posterior distributions for prediction (Section IV-D).

B. Evaluation on Streaming Tensor Entries in Arbitrary Orders

Next, we examined POST when tensor entries stream in arbitrary orders. In this scenario, MAST is no longer available, because the new tensor entries may still belong to the previous tensor, rather than appear in the tensor increment and increase the tensor size. We used two large sparse datasets from [38]: (1) *ACC*, a continuous tensor which record the three-way interactions $(user, action, resource)$. *ACC* was extracted from the log of a code repository management system. Each entry

[1]https://github.com/hanchengge/Helios-PyTen/tree/master/pyten/data
[2]https://movielens.org/

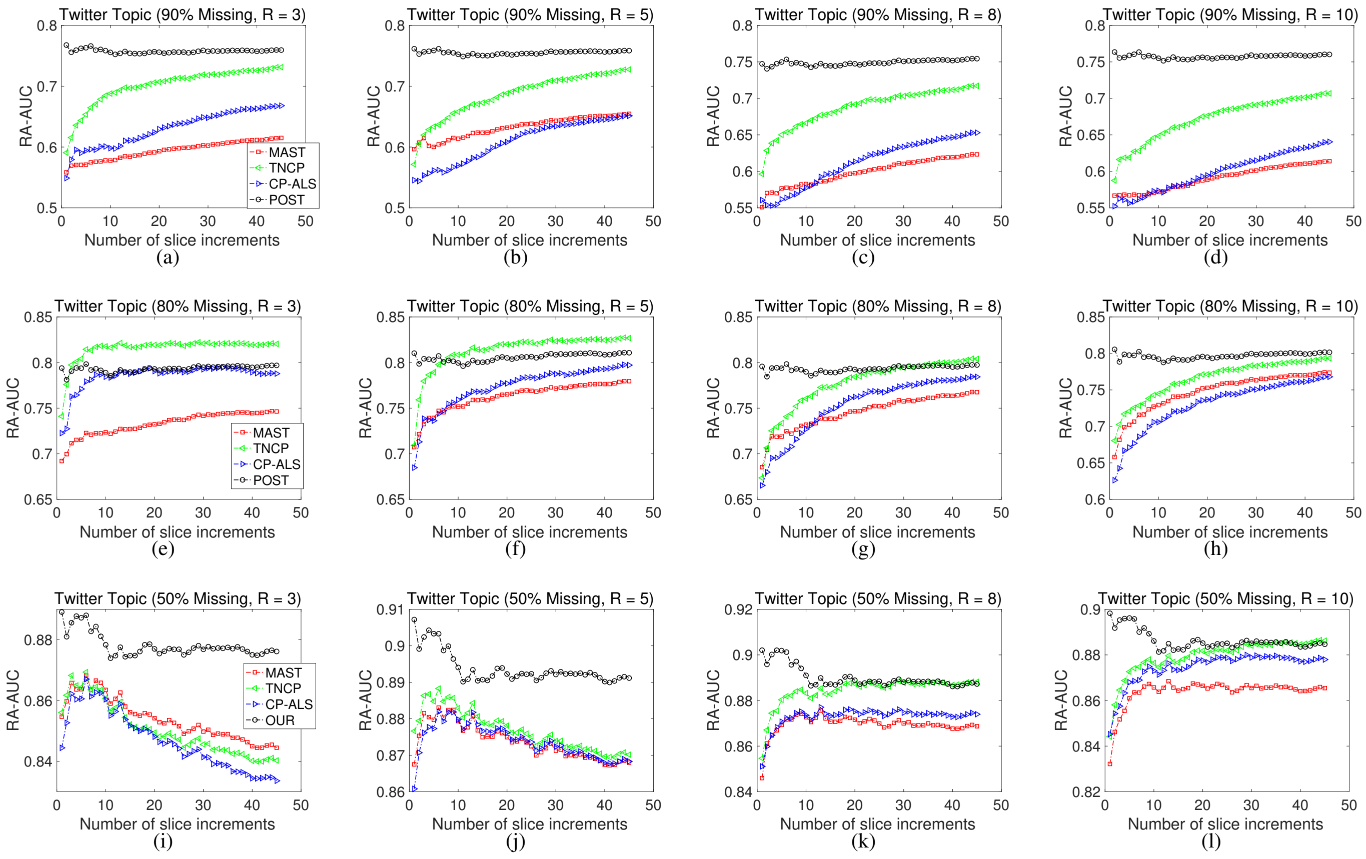

Fig. 1. The running average AUC on *Twitter Topic* dataset. Each time, 10 $user \times expert$ slices were appended. The results were averaged over 5 runs.

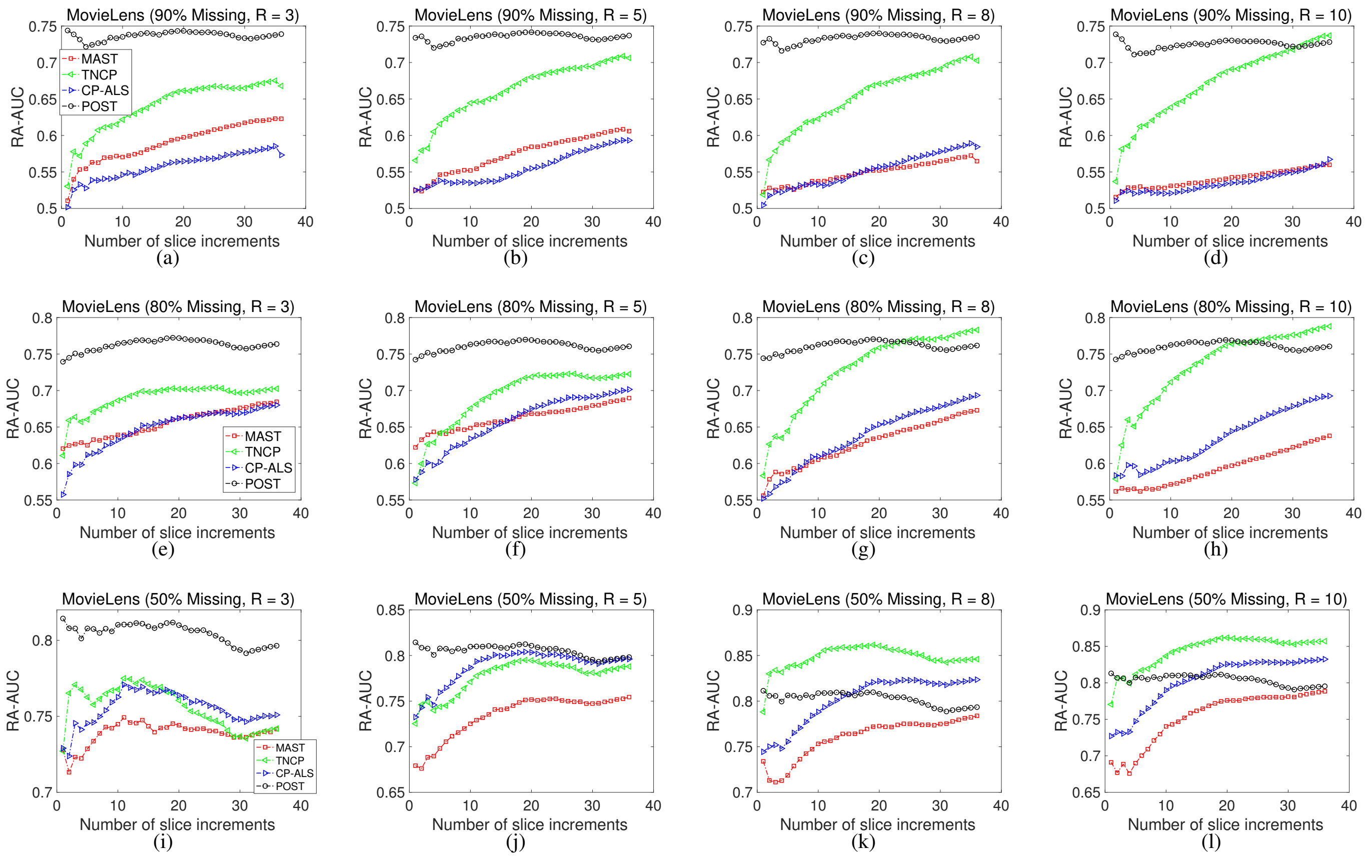

Fig. 2. The running average AUC on *MovieLen* data. Each time, 10 $user \times movie$ slices were appended. The results were averaged over 5 runs.

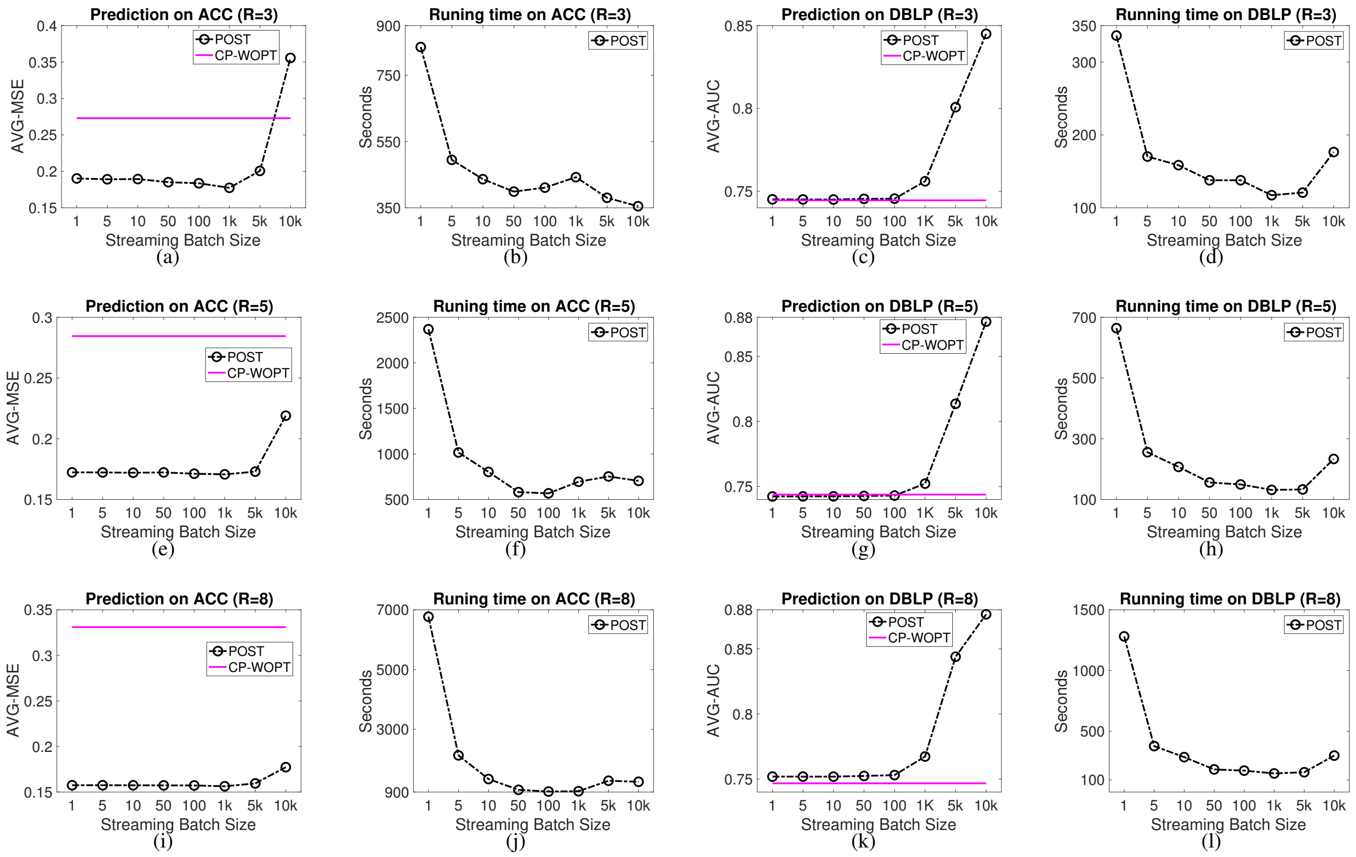

Fig. 3. The predictive performance and running time on *ACC* and *DBLP*. The results are averaged over 5 runs.

value is the logarithm of the frequency of a user accessing (e.g., downloading or uploading) some particular file (i.e., resources). The full size is $3K \times 150 \times 30K$, and $\%0.009$ entries are nonzero. (2) *DBLP*, a binary tensor describing the three-way $(author, conference, keyword)$ bibliography relationships, of size $3K \times 150 \times 30K$, contains $\%0.001$ nonzero entries.

We followed [38] to sample the training and test sets from *ACC* and *DBLP*. For each dataset, we sampled 80% nonzero entries and randomly sampled the same number of zero entries to obtain a balanced training set. Then 50 test sets were sampled from the remaining entries, each of which consisted of 200 nonzero and 1800 zero elements. In so doing, the test will not be dominated by the extremely large portion of zero entries.

To examine POST, we randomly shuffled the training entries, and partition the entries into many small batches. These batches were then fed to POST, one by one. After POST finished processing all the batches, we evaluated the predictive performance of the learned embeddings on the 50 test sets. We calculated the mean square error (MSE) and AUC for each test set of *ACC* and *DBLP*, respectively. We then averaged the results over the 50 test sets. We varied the size of the batches from $\{1, 5, 10, 50, 100, 1K, 5K, 10K\}$. We compared with CP-WOPT [3], a scalable static CP decomposition algorithm based on gradient-based optimization algorithms. We used the implementation from the MATLAB Tensor Toolbox developed by Bader et. al. [43], [44]. We used the default settings of CP-WOPT, and set $v = 1$ for POST (see Algorithm 1). We varied the rank R from $\{3, 5, 8\}$. For each setting of R, we conducted 5 runs, and in each run, we re-shuffled the training entries randomly, and hence produced the tensor entry stream in a totally different order. For a fair comparison, in each run, we also used the same initializations for the embeddings of CP-WOPT and the means of the embedding posteriors of POST (i.e., $\{\mathbf{m}_s^k\}$, see Algorithm 1), which were sampled element-wisely from a uniform distribution in $[0, 1]$.

The prediction accuracy and the running time of POST are reported in Figure 3. As we can see, POST always shows superior or comparable predictive performance to CP-WOPT, except when the batch size was set to $10K$, and $R = 3$ on *ACC* (Figure 3 a). Note that even when POST processes one tensor entry each time, it still obtains smaller MSE on *ACC* and comparable AUC on *DBLP*, under different rank settings. In most cases, the prediction accuracy of POST was improved when we used bigger streaming batches (except that on *ACC*, when the size increased from $1K$ to $5K$ and from $5K$ to $10K$, the performance dropped, and we conjecture that POST arrived at inferior local maximas in processing these batches). In terms of the running time, we can consistently observe a trade-off between the size and number of the streaming batches. When the batch size is smaller, it takes less time for POST to converge for each batch; however, POST will need to process more batches, and the total running time could be longer.

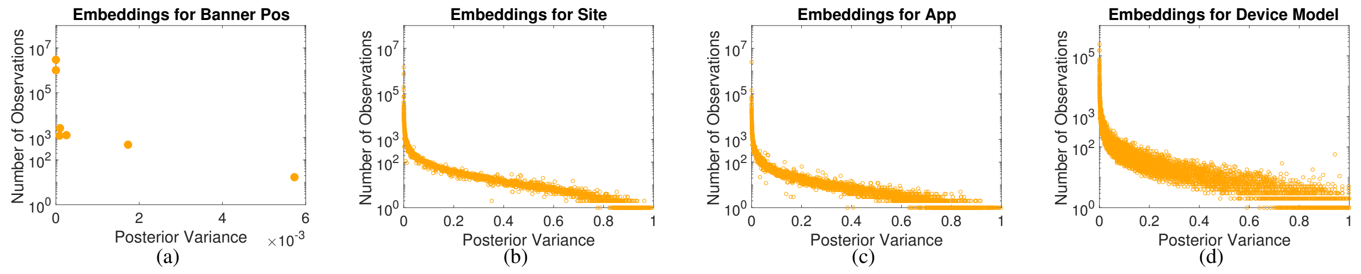

Fig. 4. The posterior variances of the embeddings ($R = 1$) *vs.* the number of observations for the corresponding objects in each mode.

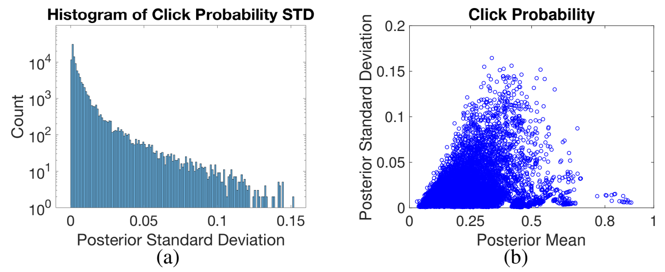

Fig. 5. The uncertainty of the click probabilities for $100K$ test entries.

C. Uncertainty Investigation

Finally, we looked into the uncertainty information provided by POST, and connected them to practical applications. To this end, we used the data from a Kaggle contest for click-through-rate (CTR) prediction in online advertising[3], sponsored by Avazu Inc. The data record the stream of ads impressions from the log of an online advertising system. We used four categorical features from the impressions, to build a four-mode binary tensor ($banner\_pos, site\_id, app\_id, device\_model$). From each impression, we extracted an observed tensor entry and its value (i.e., click or non-click). These entries were sent to POST one by one, so that POST can perform streaming decomposition. After the first $400M$ impressions were processed, we then extracted $100K$ test entries from the subsequent impressions. We skipped the impressions that associate with fresh objects which were never seen before. The tensor size finally increased to $7 \times 2854 \times 4114 \times 6061$. Out of the $400M$ impressions are $695K$ clicks (17%). We set $v = 1$. For better visualization, we set $R = 1$, so that the covariance of the embeddings will become a scalar. The AUC on the test entries is 0.74. We also ran CP-WOPT, which turned out to be dreadful, 0.49. We conjectured that it is because CP-WOPT converged too early and was trapped in poor local minimums.

We first investigated the (posterior) variances of the embeddings for the objects in each mode. As shown in Figure 4, the variances for the objects are strongly correlated to the number of observed entries that contain these objects (i.e., specific banners, sites, apps and device models). The more frequent an object was observed in past training entries, the smaller the posterior variance of its embedding. This is reasonable, because more observed data will reduce the uncertainty, and lead to an estimation with high confidence levels. When very few observations are found, the variance will approach to 1, the initial variance we set for all the embeddings. This is because very few updates were made for the corresponding embeddings during the streaming decomposition.

[3]www.kaggle.com/c/avazu-ctr-prediction/data

Second, we looked into the uncertainty of the click probability estimation, i.e., CTR. Figure 5 a shows the distribution of the posterior standard deviations (STDs) for the click probabilities of the $100K$ test entries. Most of the click probabilities have small STDs (86% of them are smaller than 0.01), implying high confidence levels. We might directly use these probabilities for online ads bidding and display. However, the STDs of the click probabilities for 1291 test entries are over 0.05, which are relatively big and not ignorable. For those entries, we might incorporate some randomly selected ads and combine with the deterministic ranking results to increase the diversity of ads displaying. This is actually a trade-off between the exploration and exploitation — we want to show more diverse types of ads to attract (new) users, but we do not want to miss those ads which are known to be good [45], [46]. Moreover, we can utilize these uncertainty information to improve the click-per-cost (CPC) performance of advertising platforms: if a campaign can spend all the budget, we can decrease the bid price for CTR predictions with high STDs to lower the risk of wasting the budget on non-clicks. On the other hand, we can increase the bid price for higher CTR predictions with smaller STDs so as to spend more budget but carefully control the CPC. More principled approaches can be further developed to integrate the uncertainty to manage the risk of bidding, such as [47].

We also contrasted the posterior means of the click probabilities with their STDs, as shown in Figure 5 b. First, we can see that most of the means of the click probabilities are less than 0.5. This is reasonable, because the ads impressions are known to be very biased — most of the ads impressions will not incur any click. In our data, only 17% impressions have clicks. Then we found that very high or very low click probabilities (e.g., the posterior means are greater than 0.8 or close to 0) have very small STDs (i.e., close to 0). That means if we predict a click will occur/not occur in a very large chance, our prediction is confident simultaneously. However, for the medium probabilities, say, between 0.25 and 0.7, their STDs vary much. This might be explained in two aspects. When the objects have little history of impressions, the click probabilities will tend to be closer to 0.5, and have big STDs (see (23), the denominator is large due to the large variance of $g_{\mathbf{i}} = \mathbf{1}^\top \mathbf{t}_{\mathbf{i}}$). However, when the objects have been observed many times, the prediction will be much more confident, i.e., with STDs close to 0. The smaller click probabilities mainly arise from more balanced click/non-click impressions associated with the objects.

VII. Conclusion

We have developed POST, a streaming probabilistic tensor decomposition algorithm. POST can process tensor entries that stream in arbitrary orders, and provide real-time estimation and uncertainty quantification. The predictive performance on real-world datasets are encouraging and the uncertainty information is useful and could potentially benefit practical tasks. In the future work, we will further investigate POST in-depth for various applications, including (but not being limited to) knowledge discovery, prediction, decision making, etc.

References

[1] Q. Song, X. Huang, H. Ge, J. Caverlee, and X. Hu, "Multi-aspect streaming tensor completion," in KDD, 2017.

[2] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," SIAM review, vol. 51, no. 3, pp. 455–500, 2009.

[3] 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, March 2011.

[4] D. Nion and N. D. Sidiropoulos, "Adaptive algorithms to track the parafac decomposition of a third-order tensor," IEEE Transactions on Signal Processing, vol. 57, no. 6, pp. 2299–2310, 2009.

[5] S. Zhou, N. X. Vinh, J. Bailey, Y. Jia, and I. Davidson, "Accelerating online CP decompositions for higher order tensors," in KDD, 2016.

[6] H. B. McMahan, G. Holt et al., "Ad click prediction: a view from the trenches," in KDD, 2013.

[7] Y. Shi, X. Zhao, J. Wang, M. Larson, and A. Hanjalic, "Adaptive diversification of recommendation results via latent factor portfolio," in SIGIR, 2012.

[8] 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.

[9] 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.

[10] T. G. Kolda, "Multilinear operators for higher-order decompositions." Sandia National Laboratories, Tech. Rep., 2006.

[11] T. Broderick, N. Boyd, A. Wibisono, A. C. Wilson, and M. I. Jordan, "Streaming variational Bayes," in NIPS, 2013.

[12] 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.

[13] J. A. Hoeting, D. Madigan, A. E. Raftery, and C. T. Volinsky, "Bayesian model averaging: a tutorial," Statistical science, pp. 382–401, 1999.

[14] T. P. Minka, "Expectation propagation for approximate Bayesian inference," in UAI, 2001.

[15] L. Xiong, X. Chen, T.-K. Huang, J. Schneider, and J. G. Carbonell, "Temporal collaborative filtering with bayesian probabilistic tensor factorization," in SDM, 2010.

[16] I. Sutskever, J. B. Tenenbaum, and R. R. Salakhutdinov, "Modelling relational data using bayesian clustered tensor factorization," in NIPS, 2009.

[17] P. Hoff, "Hierarchical multilinear models for multiway data," Computational Statistics & Data Analysis, 2011.

[18] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: Scaling tensor analysis up by 100 times—algorithms and discoveries," in KDD, 2012.

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

[20] P. Rai, Y. Wang, S. Guo, G. Chen, D. Dunson, and L. Carin, "Scalable Bayesian low-rank decomposition of incomplete multiway tensors," in ICML, 2014.

[21] J. H. Choi and S. Vishwanathan, "Dfacto: Distributed factorization of tensors," in NIPS, 2014.

[22] C. Hu, P. Rai, C. Chen, M. Harding, and L. Carin, "Scalable bayesian non-negative tensor factorization for massive count data," in ECML PKDD, 2015.

[23] C. Hu, P. Rai, and L. Carin, "Zero-truncated poisson tensor factorization for massive binary tensors," in UAI, 2015.

[24] P. Rai, C. Hu, M. Harding, and L. Carin, "Scalable probabilistic tensor factorization for binary and count data," in IJCAI, 2015.

[25] Q. Zhao, L. Zhang, and A. Cichocki, "Bayesian CP factorization of incomplete tensors with automatic rank determination," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 9, pp. 1751–1763, 2015.

[26] A. Schein, M. Zhou, D. M. Blei, and H. Wallach, "Bayesian Poisson Tucker decomposition for learning the structure of international relations," in ICML, 2016.

[27] A. Schein, J. Paisley, D. M. Blei, and H. Wallach, "Bayesian Poisson tensor factorization for inferring multilateral relations from sparse dyadic event counts," in KDD, 2015.

[28] L. Tucker, "Some mathematical notes on three-mode factor analysis," Psychometrika, vol. 31, pp. 279–311, 1966.

[29] H. Kasai, "Online low-rank tensor subspace tracking from incomplete data by cp decomposition using recursive least squares," in ICASSP, 2016.

[30] M. Mardani, G. Mateos, and G. B. Giannakis, "Subspace learning and imputation for streaming big data matrices and tensors," IEEE Transactions on Signal Processing, vol. 63, no. 10, pp. 2663–2677, 2015.

[31] J. Sun, D. Tao, and C. Faloutsos, "Beyond streams and graphs: dynamic tensor analysis," in KDD, 2006.

[32] J. Sun, D. Tao, S. Papadimitriou, P. S. Yu, and C. Faloutsos, "Incremental tensor analysis: Theory and applications," ACM Transactions on Knowledge Discovery from Data (TKDD), vol. 2, no. 3, p. 11, 2008.

[33] R. Yu, D. Cheng, and Y. Liu, "Accelerated online low rank tensor learning for multivariate spatiotemporal streams," in ICML, 2015.

[34] X. Ma, D. Schonfeld, and A. Khokhar, "Dynamic updating and downdating matrix svd and tensor hosvd for adaptive indexing and retrieval of motion trajectories," in ICASSP, 2009.

[35] W. Hu, X. Li, X. Zhang, X. Shi, S. Maybank, and Z. Zhang, "Incremental tensor subspace learning and its applications to foreground segmentation and tracking," International Journal of Computer Vision, vol. 91, no. 3, pp. 303–327, 2011.

[36] A. Sobral, C. G. Baker, T. Bouwmans, and E.-h. Zahzah, "Incremental and multi-feature tensor subspace learning applied for background modeling and subtraction," in International Conference Image Analysis and Recognition. Springer, 2014, pp. 94–103.

[37] Z. Xu, F. Yan, and Y. Qi, "Infinite Tucker decomposition: Nonparametric Bayesian models for multiway data analysis," in ICML, 2012.

[38] S. Zhe, K. Zhang, P. Wang, K.-c. Lee, Z. Xu, Y. Qi, and Z. Ghahramani, "Distributed flexible nonlinear tensor factorization," in NIPS, 2016.

[39] S. Zhe, Y. Qi, Y. Park, Z. Xu, I. Molloy, and S. Chari, "Dintucker: Scaling up Gaussian process models on large multidimensional arrays," in AAAI, 2016.

[40] S. Zhe, Z. Xu, X. Chu, Y. Qi, and Y. Park, "Scalable nonparametric multiway data analysis," in AISTATS, 2015.

[41] L. He, C.-T. Lu, G. Ma, S. Wang, L. Shen, S. Y. Philip, and A. B. Ragin, "Kernelized support tensor machines," in ICML, 2017.

[42] Y. Liu, F. Shang, L. Jiao, J. Cheng, and H. Cheng, "Trace norm regularized candecomp/parafac decomposition with missing data," IEEE Transactions on Cybernetics, vol. 45, no. 11, pp. 2437–2448, 2015.

[43] B. W. Bader, T. G. Kolda et al., "Matlab tensor toolbox version 2.6," Available online, February 2015.

[44] B. W. Bader and T. G. Kolda, "Efficient MATLAB computations with sparse and factored tensors," SIAM Journal on Scientific Computing, vol. 30, no. 1, pp. 205–231, December 2007.

[45] W. Li, X. Wang, R. Zhang, Y. Cui, J. Mao, and R. Jin, "Exploitation and exploration in a performance based contextual advertising system," in KDD, 2010.

[46] 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.

[47] H. Zhang, W. Zhang, Y. Rong, K. Ren, W. Li, and J. Wang, "Managing risk of bidding in display advertising," in WSDM, 2017, pp. 581–590.