Bayesian Streaming Sparse Tucker Decomposition

Shikai Fang, Robert M. Kirby, Shandian Zhe

School of Computing, University of Utah
Salt Lake City, UT 84112
shikai.fang@utah.edu, kirby@cs.utah.edu, zhe@cs.utah.edu

Abstract

Tucker decomposition is a classical tensor factorization model. Compared with the most widely used CP decomposition, Tucker model is much more flexible and interpretable in that it accounts for every possible (multiplicative) interaction between the factors in different modes. However, this also brings in the risk of overfitting and computational challenges, especially in the case of fast streaming data. To address these issues, we develop BASS-Tucker, a BAyesian Streaming Sparse Tucker decomposition method. We place a spike-and-slab prior over the core tensor elements to automatically select meaningful factor interactions so as to prevent overfitting and to further enhance the interpretability. To enable efficient streaming factorization, we use conditional moment matching and delta method to develop one-shot incremental update of the latent factors and core tensor upon receiving each streaming batch. Thereby, we avoid processing the data points one by one as in the standard assumed density filtering, which needs to update the core tensor for each point and is quite inefficient. We explicitly introduce and update a sparse prior approximation in the running posterior to fulfill effective sparse estimation in the streaming inference. We show the advantage of BASS-Tucker in several real-world applications.

1 INTRODUCTION

Multiway or multi-relation data are ubiquitous in real-world applications. Those data are naturally represented by tensors. For example, we can use a three-mode tensor, *(customer, product, page)*, to describe online shopping records. In order to analyze these tensors, tensor factorization provides a fundamental framework, which introduces a set of latent factors (or properties) to represent the nodes/entities in each mode. These latent factors are estimated by minimizing the reconstruction error of the observed tensor entries. Given the latent factors, we can discover the hidden patterns among the tensor nodes, such as communities and outliers. We can also use these latent factors as effective features for downstream tasks, such as commodity recommendation and click-through-rate prediction.

The most commonly used tensor decomposition approach is (probably) the CANDECOMP/PARAFAC (CP) decomposition [Harshman, 1970], which demands all the tensor nodes (across different modes) have the same number of latent factors R. CP assumes each tensor element is calculated by a summation of R products, where the r-th product is computed from the r-th latent factor of each involved node ($1 \leq r \leq R$). Despite the convenience, CP model can be quite restrictive in that it only considers R possible interactions between the latent factors. By contrast, Tucker decomposition [Tucker, 1966] is much more expressive. It allows the nodes in each mode k to have a different factor number R_k, and uses a linear combination of all possible $R_1 \times \cdots \times R_K$ interactions across the factors to model each entry value. The weights can be organized as an $R_1 \times \cdots \times R_K$ core tensor $\mathcal{W}$, and jointly estimated with the latent factors. CP decomposition can therefore be viewed as a special instance of Tucker decomposition with $R_1 = \ldots = R_K$ and $\mathcal{W}$ being diagonal. Hence, Tucker decomposition enables more flexibility and interpretability than CP decomposition.

However, in the mean time, Tucker decomposition also brings challenges in both modeling and computation. First, assuming that every possible interaction of the latent factors is taking effect might make the model overly complex, and enhance the risk of overfitting. Second, the model estimation is much more expensive (considering a relatively large core tensor $\mathcal{W}$), especially in the case of fast streaming data, which are ubiquitous in real-world applications [Du et al., 2018]. It is very costly to estimate the latent factors and core tensor from scratch every time when a batch of new entries

Accepted for the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021).

are received. Some applications that stress on privacy (*e.g.,* SnapChat) even forbid us from re-accessing previous data. Therefore, an efficient incremental update is in urgent need.

To address these issues, we develop BASS-Tucker, a Bayesian streaming sparse Tucker decomposition approach. We propose a Bayesian formulation of the Tucker decomposition, and assign a spike-and-slab prior over each element of the core tensor $\mathcal{W}$ so as to identify the effective factor interactions in the generation (reconstruction) of the entry values. Accordingly, we can reduce the model complexity and further enhance the interpretability. Second, to efficiently deal with streaming data, we extend the assumed density filtering (ADF) framework [Boyen and Koller, 1998] to jointly process each batch of incoming tensor entries. We use the conditional moment matching and delta method [Bickel and Doksum, 2015] to address the intractable moment calculation, and develop one-shot incremental posterior update for the latent factors and core tensor upon receiving each streaming batch. In so doing, we avoid sequentially processing every data point as in standard ADF, which requires us to repeatedly construct new approximations, match the moments, and is much more expensive. Next, to enable effective sparse estimation, we explicitly introduce a spike-and-slab approximation in the running posterior. By updating the approximation every a few batches, the sparse regularization of the spike-and-slab prior is constantly transmitted and reinforced in the running posterior, with which we can effectively select meaningful factor interactions during the streaming decomposition. Our incremental updates are closed-form, reliable and efficient.

We examined BASS-Tucker in four real-world applications. We compared with the state-of-the-art multilinear streaming decomposition algorithm, POST [Du et al., 2018], and static CP/Tucker decomposition algorithms that need to repeatedly access the data. In both running and final predictive performance, BASS-Tucker consistently outperforms POST, often by a large margin. The final prediction accuracy of BASS-Tucker is even significantly better than the state-of-the-art static decomposition algorithms. BASS-Tucker also enjoys a linear scalability in the data size. Finally, BASS-Tucker identifies sparse structures in the core tensor, which reflects interesting patterns within the factor interactions in reconstructing the tensor entries.

2 BACKGROUND

Tensor Decomposition. Denote a K-mode tensor by $\mathcal{Y} \in \mathbb{R}^{d_1 \times \ldots \times d_K}$, where mode k contains d_k nodes (or entities). Each tensor entry is indexed by a K-element tuple, $\mathbf{i} = (i_1, \ldots, i_K)$, where each i_k ($1 \le k \le K$) is the index of the involved node in mode k. The entry value is denoted by $y_{\mathbf{i}}$. To conduct tensor decomposition, we introduce a set of latent factors to represent the nodes in each mode. These factors may correspond to intrinsic properties or features of the nodes. Suppose in each mode k, we have R_k latent factors for every node. We can arrange these latent factors into K factor matrices, $\mathcal{U} = \{\mathbf{U}^1, \ldots, \mathbf{U}^K\}$, where each $\mathbf{U}^k$ is of size $d_k \times R_k$ and each row corresponds to one node. Our goal is to estimate $\mathcal{U}$ to recover the observed entry values in $\mathcal{Y}$. The most commonly used tensor decomposition model is CANDECOMP/PARAFAC (CP) decomposition [Harshman, 1970] which assumes $R_1 = \ldots = R_K = R$, and $\mathcal{Y} \approx \sum_{r=1}^{R} \lambda_r \cdot \left(\mathbf{U}^1(:, r) \otimes \ldots \otimes \mathbf{U}^K(:, r)\right)$, where $\otimes$ is the Kronecker product, and $\mathbf{U}^k(:, r)$ is the r-th column of $\mathbf{U}^k$. The element-wise form is

$$y_{\mathbf{i}} \approx \sum_{r=1}^{R} \lambda_r \prod_{k=1}^{K} u_{i_k, r}^k, \qquad (1)$$

where $u_{i_k,r}^k = [\mathbf{U}^k]_{i_k,r}$. To estimate the latent factors $\mathcal{U}$, we can minimize a loss function, which is typically the mean squared error of using $\mathcal{U}$ to reconstruct the observed entries in $\mathcal{Y}$. Despite the simplicity and convenience, CP only considers the interaction between every r-th factor across the K modes, namely, $\{\prod_{k=1}^{K} u_{i_k,r}^k\}_r$, and ignores all the other interactions in reconstructing the entries.

A much more expressive model is Tucker decomposition model [Tucker, 1966], which takes into account all possible factor interactions. Specifically, we introduce an $R_1 \times \ldots \times R_K$ core tensor $\mathcal{W}$ and assume that

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

where $\times_k$ the mode-k tensor matrix product [Kolda, 2006], $\mathcal{W} \times_k \mathbf{U}^k$ results in an $R_1 \times \ldots \times R_{k-1} \times d_k \times R_{k+1} \times \ldots \times R_K$ tensor and each element is calculated by

$$\left[\mathcal{W} \times_k \mathbf{U}^k\right]_{i_1,\ldots,i_K} = \sum_{j=1}^{R_k} w_{(i_1,\ldots,i_{k-1},j,i_{k+1},\ldots,i_K)} \cdot u_{i_k,j}^k.$$

The element-wise form is given by

$$y_{\mathbf{i}} \approx \sum_{r_1=1}^{R_1} \cdots \sum_{r_K=1}^{R_K} \left[w_{(r_1,\ldots,r_K)} \cdot \prod_{k=1}^{K} u_{i_k,r_k}^k \right]. \qquad (2)$$

Now, we can see that all $R_1 \times \cdots \times R_K$ factor interactions across the K modes are considered in the generation of the entry values, weighted by the elements of the core tensor $\mathcal{W}$. Therefore, Tucker model is much more expressive in capturing the interactions in data than the CP model, and can give more interpretable results.

Streaming Model Inference. Streaming variational Bayes (SVB) [Broderick et al., 2013] is a popular approach to conduct incremental posterior inference given data streams. It is based upon the incremental Bayes' rule,

$$p(\boldsymbol{\theta}|\mathcal{D}_{\text{old}} \cup \mathcal{D}_{\text{new}}) \propto p(\boldsymbol{\theta}|\mathcal{D}_{\text{old}})p(\mathcal{D}_{\text{new}}|\boldsymbol{\theta}) \qquad (3)$$

where $\boldsymbol{\theta}$ are the latent random variables in the probabilistic model we use, $\mathcal{D}_{\text{old}}$ all the data points that so far have been accessed, and $\mathcal{D}_{\text{new}}$ the newly received data points. SVB uses the variational inference framework to conduct a cyclic update of the current posterior $p(\boldsymbol{\theta}|\mathcal{D}_{\text{old}})$. Specifically, SVB introduces a tractable variational posterior $q_{\text{cur}}(\boldsymbol{\theta})$ that is usually from the exponential family to approximate the current posterior $p(\boldsymbol{\theta}|\mathcal{D}_{\text{old}})$. When $\mathcal{D}_{\text{new}}$ arrives, SVB multiplies $q_{\text{cur}}(\boldsymbol{\theta})$ with the likelihood of $\mathcal{D}_{\text{new}}$ to obtain a blending distribution,

$$\tilde{p}(\boldsymbol{\theta}) = q_{\text{cur}}(\boldsymbol{\theta})p(\mathcal{D}_{\text{new}}|\boldsymbol{\theta}), \tag{4}$$

which is considered as approximately proportional to the joint distribution $p(\boldsymbol{\theta}, \mathcal{D}_{\text{old}} \cup \mathcal{D}_{\text{new}})$. The latter, however, is infeasible to compute in the streaming scenario. Therefore, SVB uses $\tilde{p}(\boldsymbol{\theta})$ to construct a variational evidence lower bound (ELBO) [Wainwright et al., 2008], $\mathcal{L}(q(\boldsymbol{\theta})) = \mathbb{E}_q\left[\log\left(\frac{\tilde{p}(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right)\right]$, and maximizes the ELBO to update the posterior, $q_{\text{cur}} \leftarrow \operatorname{argmax}_q \mathcal{L}(q)$. This is equivalent to minimizing Kullback-Leibler (KL) divergence between q and $\tilde{p}$. We continue this procedure to conduct incremental updates for the new incoming data. At the beginning, we set $q_{\text{cur}} = p(\boldsymbol{\theta})$, the original prior in the model. For efficiency and convenience, we usually use a mean-field variational posterior, $q(\boldsymbol{\theta}) = \prod_j q(\theta_j)$, and alternately update each $q(\boldsymbol{\theta}_j)$, which is often closed-form. The state-of-the-art streaming CP decomposition algorithm, POST [Du et al., 2018], applies SVB in a Bayesian CP model to incrementally update the posterior of the factors upon receiving new tensor entries.

3 BAYESIAN SPARSE TUCKER DECOMPOSITION

Tucker decomposition is much more flexible than the most commonly used CP decomposition in that it combines all possible (multiplicative) factor interactions across the modes (see (2)) to reconstruct the tensor entries. Therefore, Tucker decomposition can capture as many effective and interpretable patterns as possible. However, such flexibility can also cause significant challenges in modeling and computation. First, assuming every factor interaction is taking effect in generating the entry values can make the model overly complex, especially given that in most applications, the data are extremely sparse, *i.e.,* the number of observed entries is far less than the tensor size. This modeling assumption can increase the risk of overfitting. Second, we have to estimate an extra parametric core-tensor $\mathcal{W}$, of size $R_1 \times \ldots \times R_K$, which is relatively large. The joint estimation of $\mathcal{W}$ and $\mathcal{U}$ can be much more costly, especially when we handle streaming data, where each newly observed entry might incur a whole update of $\mathcal{W}$. To address these challenges, we propose a Bayesian sparse Tucker decomposition model and develop an efficient one-shot streaming posterior inference algorithm (see Sec. 4).

Specifically, we formulate the Tucker decomposition in a Bayesian framework. We first sample the latent factors in each mode from the standard Gaussian prior distribution,

$$p(\mathcal{U}) = \prod_{k=1}^{K}\prod_{j=1}^{d_k}\mathcal{N}(\mathbf{u}_j^k|\mathbf{0}, \mathbf{I}). \tag{5}$$

Next, to sample the core tensor $\mathcal{W}$, we place a spike-and-slab prior over each element of $\mathcal{W}$,

$$p(\mathcal{S}|\rho_0) = \prod_{\mathbf{j}=(1,\ldots,1)}^{(R_1,\ldots,R_K)} \text{Bern}(s_{\mathbf{j}}|\rho_0),$$
$$p(\mathcal{W}|\mathcal{S}) = \prod_{\mathbf{j}} s_{\mathbf{j}}\mathcal{N}(w_{\mathbf{j}}|0, \sigma_0^2) + (1 - s_{\mathbf{j}})\delta(w_{\mathbf{j}}), \tag{6}$$

where $\mathcal{S}$ is a binary tensor of the same size with $\mathcal{W}$, Bern$(\cdot)$ is the Bernoulli distribution, and $\delta(\cdot)$ is the Dirac-delta function.

Give the latent factors $\mathcal{U}$ and core tensor $\mathcal{W}$, we sample each observed entry value $y_{\mathbf{i}}$ from

$$\begin{aligned} &p(y_{\mathbf{i}}|\mathcal{U}, \mathcal{W}, \tau) \\ &= \mathcal{N}\left(y_{\mathbf{i}}|\mathcal{W} \times_1 \left(\mathbf{u}_{i_1}^1\right)^\top \times_2 \ldots \times_K \left(\mathbf{u}_{i_K}^K\right)^\top, \tau^{-1}\right), \quad (7) \\ &= \mathcal{N}\left(y_{\mathbf{i}}|\sum_{j_1=1}^{R_1}\cdots\sum_{j_K=1}^{R_K}\left[w_{(j_1,\ldots,j_K)} \cdot \prod_{k=1}^{K} u_{i_k,j_k}^k\right], \tau^{-1}\right), \end{aligned}$$

where τ is the inverse of the noise variance. Now, combining with (6), we can see that according to the selection indicators in $\mathcal{S}$, ineffective or useless core-tensor elements will concentrate around 0, and hence deactivate the corresponding factor interactions (*i.e.,* $\prod_{k=1}^K u_{i_k,j_k}^k$). Through the posterior inference of $\mathcal{S}$, we can automatically identify effective interactions, discarding the ineffective ones, to reduce the model complexity, alleviate overfitting, and also to enhance the interpretability. Throughout this paper, we focus on continuous entry values, and hence use the Gaussian distribution. It is straightforward to extend our model and inference algorithm with other likelihoods or link functions.

We then assign a Gamma prior over τ, $p(\tau) = \text{Gam}(\tau|a_0, b_0)$. Denote the observed tensor entries by $\mathcal{F}$. The joint probability of our model is given by

$$\begin{aligned} p(\mathcal{S}, \mathcal{W}, \mathcal{U}, \mathcal{Y}, \tau) &= \prod_k\prod_j \mathcal{N}(\mathbf{u}_j^k|\mathbf{0}, \mathbf{I})\text{Gam}(\tau|a_0, b_0) \\ &\cdot \prod_{\mathbf{j}} \text{Bern}(s_{\mathbf{j}}|\rho_0)\big(s_{\mathbf{j}}\mathcal{N}(w_{\mathbf{j}}|0, \sigma_0^2) + (1 - s_{\mathbf{j}})\delta(w_{\mathbf{j}})\big) \quad (8) \\ &\cdot \prod_{\mathbf{i}\in\mathcal{F}} \mathcal{N}(y_{\mathbf{i}}|\mathcal{W} \times_1 \left(\mathbf{u}_{i_1}^1\right)^\top \times_2 \ldots \times_K \left(\mathbf{u}_{i_K}^K\right)^\top, \tau^{-1}). \end{aligned}$$

4 STREAMING INFERENCE

We now present our streaming inference algorithm. We assume the observed entries are streamed in a series of small batches, $\{\mathcal{B}_1, \mathcal{B}_2, \ldots\}$. These batches can have a varying number of tensor entries. Upon the arrival of each batch $\mathcal{B}_n$, our goal is to incrementally update the posterior of the latent factors $\mathcal{U}$, the core tensor $\mathcal{W}$, the selection indicators $\mathcal{S}$, and the inverse of the noise variance τ, without revisiting the previous batches $\{\mathcal{B}_1, \ldots, \mathcal{B}_{n-1}\}$.

One might consider applying the SVB framework discussed in Sec. 2. However, a severe problem is that SVB cannot well integrate the spike-and-slab (SS) prior in (6) with the streaming data to effectively estimate the sparse posterior of the core tensor $\mathcal{W}$. The reason is that the spike-and-slab prior is a mixture prior and we cannot set q_{cur} to the SS prior at the beginning. Instead, we have to choose a much simpler form for q_{cur} (in the exponential family) to ensure a tractable update, *e.g.*, a mean-field form $q_{\text{cur}}(\mathcal{W}, \mathcal{S}, \mathcal{U}) = q_{\text{cur}}(\mathcal{W})q_{\text{cur}}(\mathcal{S})q_{\text{cur}}(\mathcal{U})$. The SS prior will only take effect in dealing with the very first batch of entries to obtain the first estimate of q_{cur}. After that, the prior will be replaced by q_{cur}, which will repeatedly integrate with the subsequent streaming batches to update itself (see (3)). Hence, in the subsequent updates, the SS prior is separated from the data likelihoods, and cannot perform any sparse regularization in the model update. The regularization is performed by q_{cur} itself, which is much weaker than the original SS prior. For example, if we use the aforementioned mean-field form for q_{cur}, the selection indicators $\mathcal{S}$ and core tensor elements $\mathcal{W}$ are assumed to be independent. It does not encourage any sparsity in $\mathcal{W}$. In addition, for each streaming batch, SVB needs to iteratively update each component in the mean-field posterior until convergence, and hence can be quite expensive, especially for the relatively large core-tensor $\mathcal{W}$.

To address these issues, we explicitly introduce an approximation term of the SS prior in the current posterior q_{cur}. After every a few streaming batches, we update the approximation term via moment matching. In this way, the sparse regularization of the SS will be constantly injected and reinforced in the current posterior q_{cur}, which further integrates with the new data, to improve the sparse posterior estimation of $\mathcal{W}$. Furthermore, we extend the assumed-density-filtering (ADF) [Boyen and Koller, 1998] framework to perform one-shot incremental posterior update for $\mathcal{U}$, $\mathcal{W}$ and τ in parallel, avoiding the expensive iterative, alternating updates.

4.1 REFINING SPIKE-AND-SLAB PRIOR APPROXIMATION IN THE RUNNING POSTERIOR

Specifically, we approximate the current (or running) posterior with

$$q_{\text{cur}}(\mathcal{W}, \mathcal{U}, \tau) \propto p(\mathcal{S})\xi(\mathcal{W}, \mathcal{S}) \cdot \prod_{k=1}^{K}\prod_{j=1}^{d_k} \mathcal{N}(\mathbf{u}_j^k | \boldsymbol{\mu}_j^k, \mathbf{V}_j^k) \cdot \mathcal{N}\left(\text{vec}(\mathcal{W})|\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \text{Gam}(\tau|a, b), \tag{9}$$

where $\xi(\mathcal{W}, \mathcal{S})$ is an approximation to the SS prior in (6),

$$\begin{aligned}\xi(\mathcal{W}, \mathcal{S}) &= \prod\nolimits_{\mathbf{j}} \xi_{\mathbf{j}}(w_{\mathbf{j}}, s_{\mathbf{j}}) \\ &= \prod\nolimits_{\mathbf{j}} \text{Bern}\left(s_{\mathbf{j}}|c(\rho_{\mathbf{j}})\right)\mathcal{N}(w_{\mathbf{j}}|m_{\mathbf{j}}, \eta_{\mathbf{j}}) \\ &\overset{\propto}{\sim} p(\mathcal{W}|\mathcal{S}),\end{aligned} \tag{10}$$

and c is the sigmoid function, $\overset{\propto}{\sim}$ means "approximately proportional to", and $\text{vec}(\cdot)$ denotes vectorization. Contrasting to the joint probability of our model in (8), we can see that the terms other than $p(\mathcal{S})\xi(\mathcal{S}, \mathcal{W})$ in (9) essentially integrate (or summarize) the prior of $\mathcal{U}$ and τ and the likelihood of all the data points that have been seen so far.

Similar to (4), given an incoming batch $\mathcal{B}_n$, we combine the current posterior with the new data likelihood to construct a blending distribution,

$$\tilde{p}(\mathcal{U}, \mathcal{W}, \tau) = q_{\text{cur}}(\mathcal{U}, \mathcal{W}, \tau) \prod_{\mathbf{i} \in \mathcal{B}_n} p(y_{\mathbf{i}}|\mathcal{U}, \mathcal{W}, \tau), \tag{11}$$

which is an approximation to the joint distribution of all the data (see (8)). We use the idea of moment matching and projection to update $\mathcal{N}\left(\text{vec}(\mathcal{W})|\mathbf{m}, \boldsymbol{\Sigma}\right)$, $\text{Gam}(\tau|a, b)$, and the associated $\mathcal{N}(\mathbf{u}_j^k|\boldsymbol{\mu}_j^k, \mathbf{V}_j^k)$. The details will be discussed in Sec. 4.2. Then after every a few batches, we update $\xi(\mathcal{W}, \mathcal{S})$ to find the best approximation of the SS prior under the current data context (reflected in the other terms in q_{cur}). In this way, the sparse regularization effect of the SS prior can be injected and reinforced in q_{cur}, and leveraged in the subsequent posterior updates.

The update of $\xi(\mathcal{W}, \mathcal{S})$ resembles standard expectation propagation [Minka, 2001]. We update each ξ_j in parallel. We first divide the marginal posterior by the prior approximation to obtain the calibrated (or context) distribution,

$$q^{\setminus}(w_{\mathbf{j}}, s_{\mathbf{j}}) \propto \frac{q_{\text{cur}}(w_{\mathbf{j}}, s_{\mathbf{j}})}{\xi_j(w_{\mathbf{j}}, s_{\mathbf{j}})} = \text{Bern}(s_{\mathbf{j}}|\rho_0)\mathcal{N}(w_{\mathbf{j}}|\mu_{\mathbf{j}}^{\setminus}, v_{\mathbf{j}}^{\setminus}).$$

We then combine the calibrated distribution and the exact prior to obtain a tilted distribution (which is similar to the blending distribution in the streaming case),

$$\tilde{p}(w_{\mathbf{j}}, s_{\mathbf{j}}) \propto q^{\setminus}(w_{\mathbf{j}}, s_{\mathbf{j}})\left(s_{\mathbf{j}}\mathcal{N}(w_{\mathbf{j}}|0, \sigma_0^2) + (1 - s_{\mathbf{j}})\delta(w_{\mathbf{j}})\right).$$

We project $\tilde{p}$ to the exponential family to obtain the updated posterior. This is done by moment matching. That is, we compute the moments of $\tilde{p}$, with which to calculate the natural parameters to construct the updated posterior in the exponential family. It is well known that moment matching is equivalent to minimizing the KL divergence from $\tilde{p}$ to the approximate posterior.

$$q^*(w_{\mathbf{j}}, s_{\mathbf{j}}) = \text{Bern}(s_{\mathbf{j}}|c(\rho_{\mathbf{j}}^*))\mathcal{N}(w_{\mathbf{j}}|\mu_{\mathbf{j}}^*, v_{\mathbf{j}}^*), \tag{12}$$

where $\rho_{\mathbf{j}}^* = \log\left(\mathcal{N}(\mu_{\mathbf{j}}^{\setminus}|0, \sigma_0^2 + v_{\mathbf{j}}^{\setminus})/\mathcal{N}(\mu_{\mathbf{j}}^{\setminus}|0, v_{\mathbf{j}}^{\setminus})\right)$, $\mu_{\mathbf{j}}^* = c(\widehat{\rho}_{\mathbf{j}})\widehat{\mu}_{\mathbf{j}}$, $v_{\mathbf{j}}^* = c(\widehat{\rho}_{\mathbf{j}})\left(\widehat{v}_{\mathbf{j}} + (1 - c(\widehat{\rho}_{\mathbf{j}}))\widehat{\mu}_{\mathbf{j}}^2\right)$, where $\widehat{\rho}_{\mathbf{j}} =$

$\rho_{\mathbf{j}}^* + c^{-1}(\rho_0)$, $\widehat{v}_{\mathbf{j}} = \left((v_{\mathbf{j}}^{\backslash})^{-1} + \sigma_0^{-2}\right)^{-1}$, and $\widehat{\mu}_{\mathbf{j}} = \widehat{v}_{\mathbf{j}} \frac{\mu_{\mathbf{j}}^{\backslash}}{v_{\mathbf{j}}^{\backslash}}$. Finally, we update the prior approximation by

$$\xi_{\mathbf{j}}(\mathbf{w}_{\mathbf{j}}, \mathbf{s}_{\mathbf{j}}) \leftarrow q^*(w_{\mathbf{j}}, s_{\mathbf{j}}) / q^{\backslash}(w_{\mathbf{j}}, s_{\mathbf{j}}). \tag{13}$$

4.2 ONE-SHOT INCREMENTAL UPDATE

Now, let us look at how to update the remaining terms in (9). As mentioned in Sec. 4.1, upon the arrival of a streaming batch $\mathcal{B}_n$, we will construct the blending distribution in (11), which combines the prior, the information of the previous data (reflected in q_{cur}), and the information of new data. We use the ADF framework that projects the blending distribution onto the exponential family to obtain the updated posterior. This essentially is to minimize $\text{KL}(\frac{1}{Z}\tilde{p}\|q)$ where Z is the normalization constant. The optimal q is obtained by moment matching. Standard ADF only subsumes one data point at each step to construct the blending distribution and perform projection. Hence, it needs to sequentially process each entry in $\mathcal{B}_n$ and repeatedly match moments. This one-by-one strategy is inefficient, because we need to match the moments of the core tensor $\mathcal{W}$ many times, which can be very costly. In addition, repeatedly constructing approximations by projection can affect the quality of posterior updates. Therefore, we jointly integrate the whole streaming batch $\mathcal{B}_n$ via (11), and perform moment-matching only once. Our one-shot posterior update not only is much more efficient, but also avoids multiple approximation steps and hence can improve the quality of posterior estimation.

However, a critical bottleneck is that the moment of the blending distribution $\tilde{p}$ in (11) is analytically intractable, due to the complex coupling of $\mathcal{U}$ and $\mathcal{W}$ in the likelihood (see (7)). To overcome this problem, we first perform conditional moment matching, and then seek to calculate the expected conditional moments. Specifically, let us consider the update of $\mathcal{N}(\text{vec}(\mathcal{W})|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ in (9). We denote by Θ all the latent random variables in our model, and define $\Theta_{\backslash\mathcal{W}} = \Theta \backslash \{\mathcal{W}\}$. Our key observation is

$$\mathbb{E}_{\tilde{p}}\left[\boldsymbol{\phi}(\mathcal{W})\right] = \mathbb{E}_{\tilde{p}(\Theta_{\backslash\mathcal{W}})} \mathbb{E}_{\tilde{p}(\mathcal{W}|\Theta_{\backslash\mathcal{W}})}\left[\boldsymbol{\phi}(\mathcal{W})|\Theta_{\backslash\mathcal{W}}\right], \tag{14}$$

where $\boldsymbol{\phi}(\mathcal{W})$ are the required moments of $\mathcal{W}$, including the first and second-order moments here. Therefore, we can calculate the inner expectation first, namely, the conditional moments. Since it is under the conditional blending distribution (*i.e.,* given all the remaining variables fixed), the computation can be much easier. Specifically, we derive that

$$\tilde{p}(\mathcal{W}|\Theta_{\backslash\mathcal{W}}) \propto \mathcal{N}(\text{vec}(\mathcal{W})|\mathbf{m}, \text{diag}(\boldsymbol{\eta}))\, \mathcal{N}(\text{vec}(\mathcal{W})|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \cdot \prod_{\mathbf{i} \in \mathcal{F}} \mathcal{N}\left(y_{\mathbf{i}} | \left((\mathbf{u}_{i_K}^K)^\top \otimes \ldots \otimes (\mathbf{u}_{i_1}^1)^\top\right) \text{vec}(\mathcal{W}), \tau^{-1}\right),$$

where $\mathbf{m}$ is the concatenation of all $\{m_{\mathbf{j}}\}$ and $\boldsymbol{\eta}$ is the concatenation of all $\{\eta_{\mathbf{j}}\}$ (see (10)). Note that the mean in the Gaussian of $y_{\mathbf{i}}$ is obtained from the tensor algebra [Kolda, 2006]. Together this gives another Gaussian distribution, and we can immediately derive the closed-form conditional moments,

$$\mathbb{E}\left[\text{vec}(\mathcal{W})|\Theta_{\backslash\mathcal{W}}\right] = \boldsymbol{\Omega}^{-1}\left(\sum_{\mathbf{i}} \tau y_{\mathbf{i}} \mathbf{b}_{\mathbf{i}} + \text{diag}\left(\frac{\mathbf{m}}{\boldsymbol{\eta}}\right) + \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right),$$
$$\mathbb{E}\left[\text{vec}(\mathcal{W})\text{vec}(\mathcal{W})^\top|\Theta_{\backslash\mathcal{W}}\right] = \boldsymbol{\Omega}^{-1} + \mathbb{E}\left[\text{vec}(\mathcal{W})|\Theta_{\backslash\mathcal{W}}\right] \mathbb{E}\left[\text{vec}(\mathcal{W})|\Theta_{\backslash\mathcal{W}}\right]^\top, \tag{15}$$

where $\boldsymbol{\Omega} = \tau \sum_{\mathbf{i} \in \mathcal{F}} \mathbf{b}_{\mathbf{i}} \mathbf{b}_{\mathbf{i}}^\top + \text{diag}(\boldsymbol{\eta}^{-1}) + \boldsymbol{\Sigma}^{-1}$ and $\mathbf{b}_{\mathbf{i}} = \mathbf{u}_{i_K}^K \otimes \ldots \otimes \mathbf{u}_{i_1}^1$.

To obtain the moment, we need to take the expectation of the conditional moment under the marginal blending distribution, $\tilde{p}(\Theta_{\backslash\mathcal{W}})$, namely the outer expectation in (14). However, this is infeasible, because $\tilde{p}(\Theta_{\backslash\mathcal{W}})$ is analytically intractable. To overcome this problem, we observe that due to the factorization structure of q_{cur} in (9), we also maintain the moment matching between $q_{\text{cur}}(\Theta_{\backslash\mathcal{W}})$ and $\tilde{p}(\Theta_{\backslash\mathcal{W}})$, hence we can assume they are close in high-density regions. In addition, since $\tilde{p}$ is an integration of q_{cur} and a small batch of new data points, when many streaming batches have been processed, adding one more batch is unlikely to significantly change the current posterior, especially in the high-density regions. Therefore, we can use $q_{\text{cur}}(\Theta_{\backslash\mathcal{W}})$ to approximate $\tilde{p}(\Theta_{\backslash\mathcal{W}})$, and calculate the expected conditional moments. For notional convenience, we denote all the conditional moments of $\mathcal{W}$ by $\mathbf{h}(\Theta_{\backslash\mathcal{W}})$. We now resort to calculate

$$\mathbb{E}_{\tilde{p}}\left[\boldsymbol{\phi}(\mathcal{W})\right] \approx \mathbb{E}_{q_{\text{cur}}(\Theta_{\backslash\mathcal{W}})}\left[\mathbf{h}(\Theta_{\backslash\mathcal{W}})\right].$$

This is still intractable, because the conditional moments $\mathbf{h}$ is a nonlinear function of the remaining variables $\Theta_{\backslash\mathcal{W}}$ (see (15)). However, with the nice form of q_{cur}, we can use the delta method [Oehlert, 1992, Bickel and Doksum, 2015] to calculate the expectation. We present $\mathbf{h}$ with the first-order Taylor approximation at the mean,

$$\mathbf{h}(\Theta_{\backslash\mathcal{W}}) \approx \mathbf{h}\left(\mathbb{E}_{q_{\text{cur}}}[\Theta_{\backslash\mathcal{W}}]\right) + \mathbf{J}\left(\mathbb{E}_{q_{\text{cur}}}[\Theta_{\backslash\mathcal{W}}]\right)\left(\text{vec}\left(\Theta_{\backslash\mathcal{W}}\right) - \text{vec}\left(\mathbb{E}_{q_{\text{cur}}}[\Theta_{\backslash\mathcal{W}}]\right)\right), \tag{16}$$

where $\mathbf{J}(\cdot)$ is the Jacobian matrix. We now take the expectation over the Taylor approximation of $\mathbf{h}$ and obtain

$$\mathbb{E}_{q_{\text{cur}}(\Theta_{\backslash\mathcal{W}})}\left[\mathbf{h}(\Theta_{\backslash\mathcal{W}})\right] \approx \mathbf{h}\left(\mathbb{E}_{q_{\text{cur}}}[\Theta_{\backslash\mathcal{W}}]\right). \tag{17}$$

Therefore, we can simply substitute the current expectation or moments for $\Theta_{\backslash\mathcal{W}}$ in the conditional moments of $\mathcal{W}$. The computation is convenient and efficient. The theoretical analysis and guarantees about the delta method is discussed in Oehlert [1992], Wolter [2007].

We then use the moments calculated from (17) to construct the updated marginal posterior $q^*(\mathcal{W}) = \mathcal{N}(\text{vec}(\mathcal{W})|\boldsymbol{\beta}^*, \mathbf{G}^*)$. Contrasting to (9), we update

$$\mathcal{N}(\text{vec}(\mathcal{W})|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \propto \frac{\mathcal{N}(\text{vec}(\mathcal{W})|\boldsymbol{\beta}^*, \mathbf{G}^*)}{\mathcal{N}(\text{vec}(\mathcal{W})|\mathbf{m}, \text{diag}(\boldsymbol{\eta}))}. \tag{18}$$

Algorithm 1 BASS-Tucker

```
1: Initialize the spike-and-slab prior approximation and multiply
   it with all the other priors to initialize $q_{\text{cur}}(\cdot)$.
2: while a new batch of tensor entries $\mathcal{B}_n$ arrives do
3:     In parallel update $\mathcal{N}(\text{vec}(\mathcal{W})|\mathbf{m}, \mathbf{\Sigma})$, $\text{Gam}(\tau|a, b)$, and related $\mathcal{N}(\mathbf{u}_j^k|\boldsymbol{\mu}_j^k, \mathbf{V}_j^k)$ in (9) via conditional moment matching and delta method.
4:     if T streaming batches have been processed then
5:         Update the spike-and-slab prior approximation following (12) and (13).
6:     end if
7: end while
8: return the current posterior $q_{\text{cur}}(\cdot)$.
```

In parallel, we use the same approach to update $\text{Gam}(\tau|a, b)$ and all the Gaussian terms $\mathcal{N}(\mathbf{u}_j^k|\boldsymbol{\mu}_j^k, \mathbf{V}_j^k)$ in (9) associated with the current batch $\mathcal{B}_n$, namely, those corresponding to the entries of $\mathcal{B}_n$. Note that if a latent factor has no relevant tensor entries observed in $\mathcal{B}_n$, we do not update its posterior, because we do not have data to improve it.

In this way, we only perform one moment matching in parallel to update the approximate terms of q_{cur}, without the need for sequentially processing each entry in the streaming batch or cyclically update each term in many iterations. Our one-shot update is analytical, efficient and reliable. The streaming inference is summarized in Algorithm 1.

4.3 ALGORITHM COMPLEXITY

The time complexity of our incremental inference in each streaming batch is $\mathcal{O}(C^3 + BC + B\sum_{k=1}^K R_k^3)$, where $C = \prod_{k=1}^K R_k$ is the size of the core tensor and B is the batch size. Note that if we use the standard ADF to sequentially process each data point, the time complexity will increase to $\mathcal{O}(BC^3)$. The space complexity is $\mathcal{O}(C + C^2 + \sum_{k=1}^K (d_k + d_k^2))$, which is to store the posterior mean and covariance of the latent factors and core tensor.

5 RELATED WORK

As a natural extension of matrix decomposition, tensor decomposition is a classical and important topic in machine learning. Canonical approaches include CP [Harshman, 1970] and Tucker [Tucker, 1966] decomposition. Although numerous other methods have also been developed, such as [Shashua and Hazan, 2005, Chu and Ghahramani, 2009, Bader and Kolda, 2008, Sutskever et al., 2009, Acar et al., 2011, Hoff, 2011, Kang et al., 2012, Yang and Dunson, 2013, Rai et al., 2014, Choi and Vishwanathan, 2014, Hu et al., 2015, Rai et al., 2015, Schein et al., 2015, 2016, Oh et al., 2018, Du et al., 2018], the majority of these methods still inherit the CP or Tucker form, not only for their elegance and convenience, but also due to the model transparency and interpretability. Recently, a few efforts have been made to develop nonlinear decomposition models based on Gaussian processes [Rasmussen and Williams, 2006] or neural networks, such as [Xu et al., 2012, Zhe et al., 2016b, 2015, 2016a, Liu et al., 2019]. Despite their power, these methods rely on black-box models and hence sacrifice the interpretability.

Expectation propagation [Minka, 2001] is a deterministic approximate inference approach that unifies assumed-density-filtering (ADF) [Boyen and Koller, 1998] and (loopy) belief propagation [Murphy et al., 1999]. EP approximates the prior and likelihood of each data point by a term from the exponential family. ADF can be considered as an online version of EP — it maintains a holistic posterior, and applies EP to update the model with one incoming data point at a time. Hence, ADF does not need to keep approximation terms for the past data points. EP is problematic when the moment computation is intractable. Recently, Wang and Zhe [2019] proposed conditional EP to address this problem by conditional moment matching, Taylor approximation and numerical quadrature with fully factorized posteriors. Our work extends ADF to perform one-shot posterior update upon receiving a batch of streaming entries. Hence, we do not need to repeatedly conduct moment matching, which is quite expensive and might also impair the inference quality. To fulfill an efficient, tractable moment calculation, we use similar ideas as in [Wang and Zhe, 2019]. In addition, another important difference from ADF is that we explicitly maintain the approximation term of the spike and slab prior in the running posterior, rather than absorb everything in a singleton term in the exponential family. In so doing, we are able to constantly update the prior approximation, transmit and reinforce the regularization effect of the original prior to ensure a high-quality sparse posterior estimate under streaming data.

6 EXPERIMENT

6.1 PREDICTIVE PERFORMANCE

Datasets. We evaluated BASS-Tucker in four real-world applications. (1) *Alog* [Zhe et al., 2016b], a real-valued three-mode tensor of size $200 \times 100 \times 200$, representing three-way resource management operations *(user, action, resource)*. It includes 0.66% observed entries. (2) *MovieLen1M* (`www.grouplens.org/datasets/movielens/`), a two-mode tensor of size $6,040 \times 3,706$, comprising continuous *(user, movie)* ratings. We have $1,000,209$ observed entries. (3) *ACC* [Du et al., 2018], a real-valued tensor extracted from a large file-access log *(user, action, file)*, of size $3,000 \times 150 \times 30,000$, including 0.9% nonzero entries. In addition, we also tested a binary tensor (4) *Anime* (`www.kaggle.com/CooperUnion/anime-recommendations-database`), a two-mode tensor about *(user, anime)* preferences, of size $25,838 \times 4,066$, including $1,300,160$ observed elements.

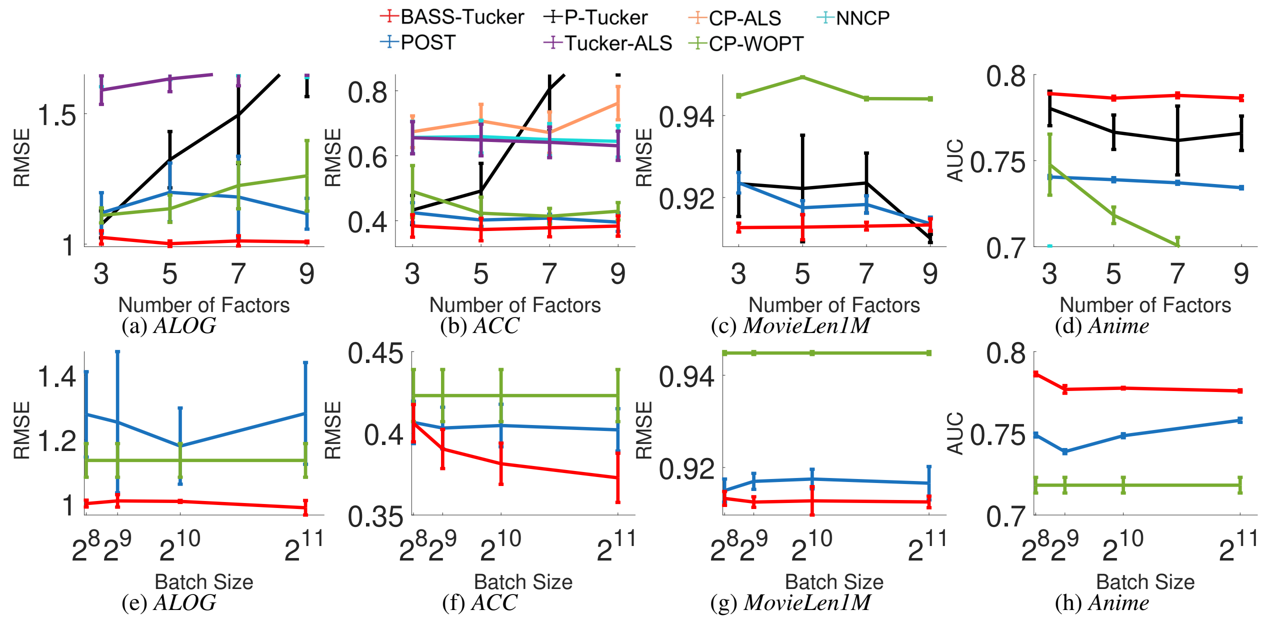

Figure 1: Predictive performance with different numbers of factors (top row) and streaming batch sizes (bottom row). In the top row, the streaming bath size is fixed to 512; in the bottom row, the factor number is fixed to 5. The results are averaged over 5 runs. Note that the performance of several baselines are missing or incomplete, because they are far worse than all the other methods and hence not included.

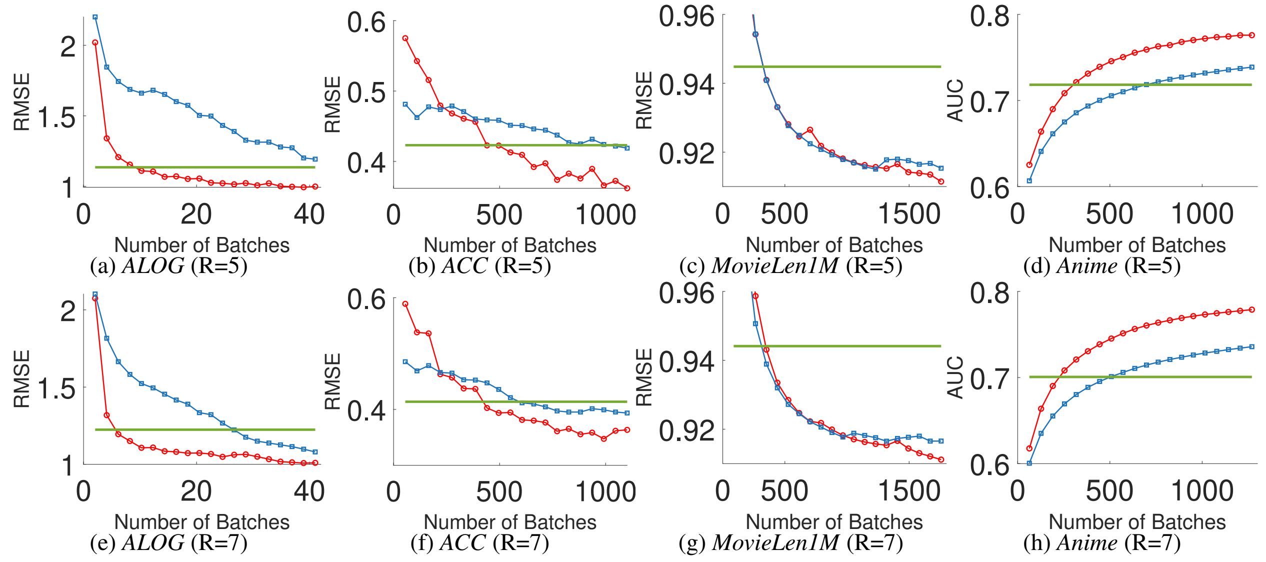

Figure 2: Running prediction accuracy along with the number of processed streaming batches. In the top row, the factor number is fixed to 5; in the bottom row, the factor number is fixed to 7. The batch size is fixed to 512.

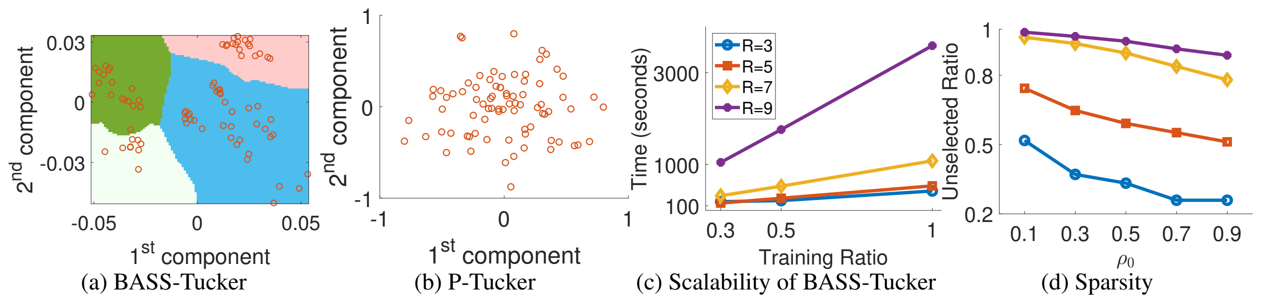

Figure 3: The structures of the estimated core-tensor by BASS-Tucker and P-Tucker (a, b), the scalability of BASS-Tucker (c) and the sparsity achieved by BASS-Tucker.

Competing methods. We compared with the state-of-the-art (SOA) multilinear streaming tensor decomposition algorithm (1) POST [Du et al., 2018], which is based on SVB [Broderick et al., 2013]. In addition, we compared with SOA static multilinear decomposition methods, including (2) P-Tucker [Oh et al., 2018], a highly efficient Tucker decomposition algorithm that conducts row-wise updates in parallel, (3) CP-WOPT [Acar et al., 2011], CP decomposition by conjugate gradient descent, (4) CP-ALS Bader et al. [2015], CP decomposition via alternating least square updates, (5) NNCP [Lee and Seung, 1999], non-negative CP decomposition with multiplicative updates, and (6) Tucker-ALS [De Lathauwer et al., 2000], Tucker decomposition with alternating least square updates.

Settings and Results. We implemented BASS-Tucker with MATLAB. To estimate a sparse core tensor $\mathcal{W}$, we set $\rho_0 = 0.5$ and $\sigma_0^2 = 1$ (see (6)). We used the original implementation of all the competing approaches and their default settings (*e.g.,* the maximum number of iterations for static decomposition). We first evaluated the final predictive performance, namely when all the streamed entries have been processed. To this end, we randomly split the observed entries of *Alog* into 75% for training and 25% for test and the other datasets 90% for training and 10% for test. For BASS-Tucker and POST, we randomly partitioned the training entries into a stream of small batches. The other methods conducted iterative static decomposition and need to repeatedly access the whole data. We repeated the experiment for 5 times, and calculated the average root-mean-squared-error (RMSE) for the real-valued datasets and area under ROC curve (AUC) for the binary dataset. The average RMSE/AUC and standard deviation are reported in Fig. 1. In Fig. 1 a-d, we fixed the streaming batch size to 512, and show how the final predictive performance of each method varied with different factor numbers, $\{3, 5, 7, 9\}$. In Fig. e-h, we fixed the factor number to 5, and examined the performance under different batch sizes, $\{2^8, 2^9, 2^{10}, 2^{11}\}$. The more the factors/larger the streaming batch size, the more expensive for BASS-Tucker to factorize each batch. Hence these settings examined the trade-off between the accuracy and computational complexity. In Fig. 2 of the supplementary material, we show the running time of BASS-Tucker under different sizes of streaming batches. As we can see, in all the cases BASS-Tucker outperforms SOA streaming approach POST and SOA static decomposition methods except that in Fig. 1 c, BASS-Tucker is slightly worse than P-Tucker when the number factor is 9. In most cases, BASS-Tucker shows an significant improvement ($p<0.05$). Note that P-Tucker estimates a dense core-tensor and needs to access the data many time. The results demonstrate the advantage of BASS-Tucker (including sparse core tensor estimation) in prediction accuracy.

6.2 PREDICTION ON THE FLY

Next, we evaluated the running predictive performance of BASS-Tucker. We fixed the batch size to 512, and randomly streamed the training entries to BASS-Tucker and POST. We tested the prediction accuracy after each streaming batch was processed, with the factor number $R = 5$ and $R = 7$. The running RMSE/AUC is shown in Fig. 2. As we can see, at the beginning, POST is close to or event better than BASS-Tucker in prediction accuracy. This is reasonable, because our Tucker model is much more complex than the CP model. However, at later stages, BASS-Tucker consistently outperforms POST, mostly by a large margin. Even in Fig. 2 c and g, while POST and BASS-Tucker overlapped quite a long time, BASS-Tucker beat POST when the data stream is about to be finished.

In addition, we examined the scalability of BASS-Tucker. On *ACC*, we fixed the batch size to 512, and streamed 30%, 50% and 100% of observed entries to BASS-Tucker and tested the streaming decomposition time. We varied the factor number R from $\{5, 7, 9, 11\}$. As we can see in Fig. 3c, the running time of BASS-Tucker grows linearly in the number of streamed entries, and the factor number R determines the slope. Therefore, BASS-Tucker enjoys a linear scalability to the data size.

To evaluate if BASS-Tucker can indeed estimate a sparse core-tensor. We varied ρ_0 from (0.1, 0.9) and ran BASS-Tucker on *Alog* dataset. An element of core-tensor $\mathcal{W}$ is viewed as selected if the posterior mean of its selection probability is no less than 0.5. We showed how the ratio of the unselected elements varies with ρ_0 in Fig. 3 d under different factor numbers. We can see that small ρ_0 pruned most of the elements while large ρ_0 preserved most, showing that BASS-Tucker can effectively achieve sparsity in the streaming setting.

6.3 SPARSE STRUCTURE IN CORE TENSOR

Finally, we examined if the estimated sparse core-tensor by BASS-Tucker can reflect some structure, as compared with the standard Tucker decomposition. To this end, we set the number of latent factors to 9, and ran BASS-Tucker and P-Tucker on *Alog* dataset. The core tensor is of size $9 \times 9 \times 9$. Then, in each mode, we fold the core tensor to an 81×9 matrix, where each row indicates how strongly each factor combination from the other modes interact with the 9 factors in the current mode. We then ran Principled Component Analysis (PCA) and projected the interaction matrix onto a plane. The positions of the points represent the first and second principle components, which summarize how every factor combination from the other modes interact with the factors in the current mode. We show the results for the first mode in Fig. 3 a and b. As we can see, from BASS-Tucker, the interaction between the factors in the first mode and in the other modes clearly exhibit clustering structures, implying different interaction patterns. To show the structures, we ran the k-means algorithm and filled the cluster regions with different colors. By contrast, the core-tensor estimated by P-Tucker do not reflect apparent structures, and the interaction strengths are distributed like a symmetric Gaussian. In the supplementary material, we show that the principled components of BASS-Tucker for the second mode also exhibit interesting structures while P-Tucker does not. While all the datasets have been completely anonymized and we are unable to look into the meaning of the patterns discovered by BASS-Tucker, these results have demonstrated that BASS-Tucker, as compared with standard Tucker decomposition, can potentially discover more interesting patterns and knowledge, and enhance the interpretability.

7 CONCLUSION

We have presented BASS-Tucker, a streaming Bayesian sparse Tucker decomposition algorithm. Our method can efficiently handle streaming data, provide one-shot posterior update, and estimate a sparse core tensor online to alleviate overfitting and to enhance knowledge discovery.

ACKNOWLEDGMENTS

This work has been supported by MURI AFOSR grant FA9550-20-1-0358 and NSF IIS-1910983.

References

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

Brett W Bader and Tamara G Kolda. Efficient matlab computations with sparse and factored tensors. SIAM Journal on Scientific Computing, 30(1):205–231, 2008.

Brett W. Bader, Tamara G. Kolda, et al. Matlab tensor toolbox version 2.6. Available online, February 2015. URL www.sandia.gov/~tgkolda/TensorToolbox/.

Peter J Bickel and Kjell A Doksum. Mathematical statistics: basic ideas and selected topics, volume I, volume 117. CRC Press, 2015.

Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence, pages 33–42, 1998.

Tamara Broderick, Nicholas Boyd, Andre Wibisono, Ashia C Wilson, and Michael I Jordan. Streaming variational bayes. In Advances in neural information processing systems, pages 1727–1735, 2013.

Joon Hee Choi and S Vishwanathan. Dfacto: Distributed factorization of tensors. In Advances in Neural Information Processing Systems, pages 1296–1304, 2014.

Wei Chu and Zoubin Ghahramani. Probabilistic models for incomplete multi-dimensional arrays. AISTATS, 2009.

Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors. SIAM journal on Matrix Analysis and Applications, 21(4):1324–1342, 2000.

Yishuai Du, Yimin Zheng, Kuang-chih Lee, and Shandian Zhe. Probabilistic streaming tensor decomposition. In 2018 IEEE International Conference on Data Mining (ICDM), pages 99–108. IEEE, 2018.

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

P.D. Hoff. Hierarchical multilinear models for multiway data. Computational Statistics & Data Analysis, 55:530–543, 2011. ISSN 0167-9473. URL http://www.stat.washington.edu/hoff/Code/hoff_2011_csda.

Changwei Hu, Piyush Rai, and Lawrence Carin. Zero-truncated poisson tensor factorization for massive binary tensors. In UAI, 2015.

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

Tamara Gibson Kolda. Multilinear operators for higher-order decompositions, volume 2. United States. Department of Energy, 2006.

Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. Nature, 401 (6755):788–791, 1999.

Hanpeng Liu, Yaguang Li, Michael Tsang, and Yan Liu. Costco: A neural tensor completion model for sparse tensors. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 324–334, 2019.

Thomas P Minka. Expectation propagation for approximate bayesian inference. In Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence, pages 362–369, 2001.

Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. In Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence, pages 467–475. Morgan Kaufmann Publishers Inc., 1999.

Gary W Oehlert. A note on the delta method. The American Statistician, 46(1):27–29, 1992.

Sejoon Oh, Namyong Park, Sael Lee, and Uksong Kang. Scalable tucker factorization for sparse tensors-algorithms and discoveries. In 2018 IEEE 34th International Conference on Data Engineering (ICDE), pages 1120–1131. IEEE, 2018.

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

Piyush Rai, Changwei Hu, Matthew Harding, and Lawrence Carin. Scalable probabilistic tensor factorization for binary and count data. In IJCAI, 2015.

Carl Edward Rasmussen and Christopher K. I. Williams. Gaussian Processes for Machine Learning. MIT Press, 2006.

Aaron Schein, John Paisley, David M Blei, and Hanna Wallach. Bayesian poisson tensor factorization for inferring multilateral relations from sparse dyadic event counts. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1045–1054. ACM, 2015.

Aaron Schein, Mingyuan Zhou, David M. Blei, and Hanna Wallach. Bayesian poisson tucker decomposition for learning the structure of international relations. In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML'16, pages 2810–2819. JMLR.org, 2016. URL http://dl.acm.org/citation.cfm?id=3045390.3045686.

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

Ilya Sutskever, Joshua B Tenenbaum, and Ruslan R Salakhutdinov. Modelling relational data using bayesian clustered tensor factorization. In Advances in neural information processing systems, pages 1821–1828, 2009.

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

Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. Foundations and Trends® in Machine Learning, 1(1–2): 1–305, 2008.

Zheng Wang and Shandian Zhe. Conditional expectation propagation. In UAI, page 6, 2019.

Kirk Wolter. Introduction to variance estimation. Springer Science & Business Media, 2007.

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

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

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

Shandian Zhe, Yuan Qi, Youngja Park, Zenglin Xu, Ian Molloy, and Suresh Chari. Dintucker: Scaling up gaussian process models on large multidimensional arrays. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 30, 2016a.

Shandian Zhe, Kai Zhang, Pengyuan Wang, Kuang-chih Lee, Zenglin Xu, Yuan Qi, and Zoubin Ghahramani. Distributed flexible nonlinear tensor factorization. In Advances in Neural Information Processing Systems, pages 928–936, 2016b.