BAYESIAN TENSOR DECOMPOSITION FOR DYNAMIC DATA

PROPOSAL

Shikai Fang
Kahlert School of Computing & Scientific Computing and Image Institute
The University of Utah
shikai.fang@utah.edu

1 Introduction

Tensor data is a type of multidimensional data that arises in many real-world applications. This data structure is a natural representation of information that has multiple modes or dimensions, such as time, space, and type. In recent years, the use of tensor data has become increasingly common in a wide range of fields, including recommendation systems, neuroscience, and climate modeling, among others. For example, in recommendation systems, a three-mode tensor: *user, item, website* can be extracted, whose entry values represent user-item interactions along the different websites. In climate modeling, 4-mode tensor: *longitude latitude, altitude, time* tensors are used to represent spatiotemporal data, such as satellite measurements of temperature change. While in neuroscience, they are used to model brain activity with multiple modalities. However, analyzing tensor data poses unique challenges due to its high dimensionality and complex structure. One promising approach to address these challenges is tensor decomposition, which aims to decompose a tensor into a set of low-rank representations, known as factors. This allows for more efficient processing, analysis, and interpretation of tensor data, and has led to the development of new methods for machine learning, data mining, and signal processing.

Classical tensor decomposition methods, such as CANDECOMP/PARAFAC (CP) (Harshman, 1970) and Tucker decomposition (Tucker, 1966), have been widely used in the analysis of tensor data. However, these methods have two main limitations when it comes to dynamic data. First, they are designed to run the factorization from scratch every time when receive a new set of entries, and are not well-suited to handle data that arrives in a dynamic way, known as streaming data. In this sense, the term "dynamic" refers to the manner in which the data arrives. Second, classical tensor decomposition methods do not take into account dynamic features of the data, such as timestamps. This means that they are not able to capture changes of temporal patterns in the data. In this sense, the term "dynamic" refers to the time-varying nature of the data. Additionally, these methods can result in dense models that are at high risk of overfitting when applied to complex tensor data, such as those encountered in deep neural network-based models.

To address these challenges, we propose a range of model designs and efficient algorithms using the Bayesian framework. Specifically, we propose a Bayesian tensor decomposition methods with the following abilities to model dynamic data:

- **Efficient Streaming Inference**: The proposed methods are designed to handle streaming data and conduct efficient online learning inference, which enables the incremental update of the model parameters, rather than starting from scratch as new data arrives. This approach is computationally efficient and can be applied to large-scale data sets in real time.

- **Temporal Pattern Modeling**: The proposed method is capable of capturing the temporal features present in tensor data and modeling the time-varying patterns of the data in a continuous and dynamic manner.

- **Flexible Priors and Interpretability**: The models can be assigned over different priors to fit the needs of various scenarios. Furthermore, the final learned tensor factors can be used to interpret the underlying structure of the data and provide insights into the data.

To solve those problems, we will present our contribution to Bayesian tensor learning for dynamic data in the following sections. Section 2 will provide background on the key concepts and methods used in our research, including Bayesian tensor decomposition, Assumed Density Filter (ADF) framework and Gaussian Process.

In Section 3, we will provide a detailed introduction of three of our previous works:

- **Bayesian Streaming Sparse Tucker(BASS)** (Fang et al., 2021a): BASS developed the one-shot incremental update with conditional moment-matching and Delta's method to handle streaming tensor data and applies sparse priors over the Tucker core.
- **Streaming Bayesian Deep Tensor Factorization(SBDT)** (Fang et al., 2021b): With Assumed Density Filter (ADF) and message passing, SBDT applies streaming inference and slab-and-spike priors to DNN-based tensor models, which have a greater capacity to capture nonlinear relationships in data without overfitting risks.
- **Bayesian Continuous-Time Tucker decomposition(BCTT)** (Fang et al., 2022): BCTT models the Tucker core as a time-varying function using the stochastic differential equation (SDE) representation of the Gaussian Process (GP) and is capable of modeling temporal patterns in a continuous and dynamic manner.

In Section 4, we will present our proposed research to combine the two meanings of "dynamic": streaming and temporal model together, to give a unified framework to model the temporal Bayesian tensor decomposition in a streaming manner.

2 Background

2.1 Bayesian Tensor Decomposition

Standard Tensor Decomposition Standard tensor decomposition methods operate under the assumption that a K-mode tensor $\mathcal{Y} \in \mathbb{R}^{d_1 \times \cdots \times d_K}$ contains d_k independent and individual objects, or "nodes," in each mode k. Each entry in the tensor is indexed by a K-size tuple $\mathbf{i} = (i_1, \ldots, i_K)$, with the corresponding value denoted as $y\mathbf{i}$. To decompose the tensor into a compact and low-rank form, K groups of latent factor matrices are used to represent the objects in each mode. These matrices are denoted as $\mathcal{U} = \{\mathbf{U}^1 \ldots \mathbf{U}^K\}$, where $\mathbf{U}^k = \left[\mathbf{u}_1^k \ldots \mathbf{u}_{d_k}^k\right]^\top \in \mathbb{R}^{d_k \times r_k}$. Each $\mathbf{u}_j^k \in \mathbb{R}^{r_k \times 1}$ represents the latent factor for the j-th object in the k-th mode, where r_k is the predefined rank of the k-th mode.

Based on the assumption that each entry value $y_\mathbf{i} = y_{(i_1,\ldots,i_K)}$ represents the interaction, denoted as f, of corresponding latent factors in each mode $\mathbf{u}_{i_1}^1, \ldots, \mathbf{u}_{i_K}^K$, the specific form of interaction function could vary. The classic CANDECOMP/PARAFAC (CP) (Harshman, 1970) assumes $r_1 = r_2 \cdots = r_K$ and designs f as a rank-wise product of all latent factors, followed by a sum-over procedure:

$$y_\mathbf{i} \approx f(\mathbf{u}_{i_1}^1, \ldots, \mathbf{u}_{i_K}^K) = \left(\mathbf{u}_{i_1}^1 \circ \ldots \circ \mathbf{u}_{i_K}^K\right)^\top \mathbf{1} \tag{1}$$

where $\circ$ is the element-wise product (also known as Hadamard product) and $\mathbf{1}$ is a constant all-one vector.

Tucker decomposition (Tucker, 1966) takes one more step towards a more expressive interaction. It parameterizes the f with a core tensor $\mathcal{W} \in \mathbb{R}^{r_1 \times \cdots \times r_K}$, which models the interaction as the sum over all possible cross-rank products weighted by $\mathcal{W}$ among the latent factors. Specifically, the element-wise formula is:

$$y_\mathbf{i} \approx f(\mathbf{u}_{i_1}^1, \ldots, \mathbf{u}_{i_K}^K) = \mathrm{vec}(\mathcal{W})^\top \left(\mathbf{u}_{i_1}^1 \otimes \ldots \otimes \mathbf{u}_{i_K}^K\right) \tag{2}$$

where$\otimes$ is the Kronecker product and $\mathrm{vec}(\cdot)$ is the vectorization operator. Tucker decomposition allows different ranks for each mode, making it more flexible than CP. It will degenerate to CP if we set all modes' ranks equal and $\mathcal{W}$ diagonal.

Bayesian framework In the Bayesian framework, the observed entry values of tensor $\mathcal{Y}$ is assumed to be generated from the decomposition model f with Gaussian noise, i.e.,$y_\mathbf{i} \approx \mathcal{N}(y_\mathbf{i} \mid f(\mathbf{u}_{i_1}^1, \ldots, \mathbf{u}_{i_K}^K), \tau^{-1})$, where τ is the precision of the noise. We further set the prior distribution of the latent factors is assumed to be Gaussian, i.e., $p(\mathbf{u}_j^k) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and the prior of the τ be Gamma distribution, i.e., $p(\tau) = \mathrm{Gamma}(\alpha, \beta)$. Then the cannon form of the joint probability of Bayesian tensor decomposition is:

$$p_f(y_\mathbf{i}, \mathcal{U}, \tau) = p(\mathcal{U})p(\tau)\mathcal{N}(y_\mathbf{i} \mid f(\mathbf{u}_{i_1}^1, \ldots, \mathbf{u}_{i_K}^K), \tau^{-1}) \tag{3}$$

2.2 Assumed Density Filtering: Bayesian Online Learning

Assumed-Density Filtering (ADF) (Boyen and Koller, 2013), also known as Bayesian online learning, is a Bayesian inference framework that allows for the efficient online posterior update for new data. It can be viewed as an online version of expectation propagation (EP) (Minka, 2001a), and is a special case of the more general class of streaming variational inference methods (Broderick et al., 2013). ADF is based on the incremental version of Bayes' rule:

$$p\left(\boldsymbol{\theta} \mid \mathcal{D}_{\text{old}} \cup \mathcal{D}_{\text{new}}\right) \propto p\left(\boldsymbol{\theta} \mid \mathcal{D}_{\text{old}}\right) p\left(\mathcal{D}_{\text{new}} \mid \boldsymbol{\theta}\right) \tag{4}$$

where $\mathcal{D}_{\text{old}}$ is the data observed so far, $\mathcal{D}_{\text{new}}$ is the new data, and $\boldsymbol{\theta}$ is the parameter of interest.

The key idea of ADF is to approximate the posterior distribution of the parameters $\boldsymbol{\theta}$ by a tractable distribution $q_{\text{old}}\left(\boldsymbol{\theta}\right)$ in the exponential family, i.e. Gaussian distribution, that is close to the true posterior $p(\boldsymbol{\theta} \mid \mathcal{D}_{\text{old}})$. Then, the new data $\mathcal{D}_{\text{new}}$ arrives, ADF integrates the current $q_{\text{old}}\left(\boldsymbol{\theta}\right)$ along with the data likelihood $p\left(\mathcal{D}_{\text{new}} \mid \boldsymbol{\theta}\right)$ and build an unnormalized distribution: $\tilde{p}(\boldsymbol{\theta}) = q_{\text{cur}}(\boldsymbol{\theta}) p\left(\mathcal{D}_{\text{new}} \mid \boldsymbol{\theta}\right)$. Then ADF projects the unnormalized distribution $\tilde{p}(\boldsymbol{\theta})$ to the exponential family to obtain the updated posterior $q_{\text{new}}\left(\boldsymbol{\theta}\right)$ through moment matching. Finally, ADF updates the current approximation $q_{\text{old}}\left(\boldsymbol{\theta}\right)$ to $q_{\text{new}}\left(\boldsymbol{\theta}\right)$, which is then used to approximate the true posterior $p(\boldsymbol{\theta} \mid \mathcal{D}_{\text{old}} \cup \mathcal{D}_{\text{new}})$. Thanks to the property of exponential family distribution, the projection of $\tilde{p}(\boldsymbol{\theta})$ to the $q_{\text{new}}\left(\boldsymbol{\theta}\right)$ can be done analytically:

$$\mu^* = \mu + v \frac{\partial \log Z}{\partial \mu} \tag{5}$$

$$v^* = v - v^2 \left[\left(\frac{\partial \log Z}{\partial \mu} \right)^2 - 2 \frac{\partial \log Z}{\partial v} \right] \tag{6}$$

where μ and v are the mean and variance of $q_{\text{old}}\left(\boldsymbol{\theta}\right)$, μ^* and v^* are the mean and variance of $q_{\text{new}}\left(\boldsymbol{\theta}\right)$, and Z is the normalization constant of $\tilde{p}(\boldsymbol{\theta})$.

2.3 Gaussian Process

Gaussian Process (GP) (Rasmussen and Williams, 2006) is a powerful non-parametric Bayesian model that can be used to model a wide range of functions. It is a generalization of the multivariate Gaussian distribution to infinite-dimensional spaces. The key idea of GP is to represent a function $f(\cdot)$ as a random variable:

$$f \sim \mathcal{GP}(m(\cdot), \kappa(\cdot, \cdot)) \tag{7}$$

where $m(\cdot)$ is the mean function and $\kappa(\cdot, \cdot)$ is the covariance function. The mean function $m(\cdot)$ is a deterministic function that is used to model the mean of the function, always set as $\mathbf{0}$ The covariance function $\kappa(\cdot, \cdot)$ is a positive definite function that is used to model the correlation between the function values at different points, also known as kernel function. One of the most popular and powerful kernel is Matérn kernel:

$$\kappa(x, x') = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}}{l} \Delta \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}}{l} \Delta \right) \tag{8}$$

where Δ is the distance measure between x and x', always set as Euclidean: $\Delta = |x - x'|$; ν is a half-integer positive number of kernel smoothness; $\{l, \sigma^2\}$ are the length-scale and magnitude parameter, respectively;Γ is the gamma function, K_ν is the modified Bessel function of the second kind. With finite N observations of $\{(x_i, y_i)\}_{i=1}^N$, GP prior (7) will turn out a multivariate Gaussian distribution: $p(f(x)) = \mathcal{N}(\mathbf{0}, \mathbf{K})$. $\mathbf{K} \in \mathbb{R}^{N \times N}$ is the kernel matrix, whose element $[K]_{i,j}$ is computed through the kernel $\kappa(x_i, x_j)$.

3 Research As Far

3.1 Bayesian Streaming Sparse Tucker Decomposition (BASS)

Tucker decomposition (Tucker, 1966) 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, 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 in several real-world applications.

3.1.1 Model

Tucker decomposition (Tucker, 1966) 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. 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 BASS model and develop an efficient one-shot streaming posterior inference algorithm..

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,

$$p(\mathcal{U}) = \prod_{k=1}^{K} \prod_{j=1}^{d_k} \mathcal{N}(\mathbf{u}_j^k | \mathbf{0}, \mathbf{I}). \tag{9}$$

Next, to sample the core tensor $\mathcal{W}$, we place a spike-and-slab prior (Ishwaran and Rao, 2005; Titsias and Lázaro-Gredilla, 2011) over each element of $\mathcal{W}$,

$$p(\mathcal{S}|\rho_0) = \prod_{\mathbf{j}=(1,\ldots,1)}^{(R_1,\ldots,R_K)} \text{Bern}(s_\mathbf{j}|\rho_0),\ p(\mathcal{W}|\mathcal{S}) = \prod_{\mathbf{j}} s_\mathbf{j}\mathcal{N}(w_\mathbf{j}|0,\sigma_0^2) + (1-s_\mathbf{j})\delta(w_\mathbf{j}), \tag{10}$$

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

$$\begin{aligned} p(y_\mathbf{i}|\mathcal{U},\mathcal{W},\tau) &= \mathcal{N}\left(y_\mathbf{i}|\mathcal{W} \times_1 \left(\mathbf{u}_{i_1}^1\right)^\top \times_2 \ldots \times_K \left(\mathbf{u}_{i_K}^K\right)^\top, \tau^{-1}\right), \\ &= \mathcal{N}\left(y_\mathbf{i}|\sum_{j_1=1}^{R_1} \ldots \sum_{j_K=1}^{R_K} \left[w_{(j_1,\ldots,j_K)} \cdot \prod_{k=1}^{K} u_{i_k,j_k}^k\right], \tau^{-1}\right), \end{aligned} \tag{11}$$

where $\times_k$ the mode-k tensor matrix product (Kolda, 2006), τ is the inverse of the noise variance. We claim (11) is equal (2) to Now, combining with (10), 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

$$p(\mathcal{S}, \mathcal{W}, \mathcal{U}, \mathcal{Y}, \tau) = \prod_k \prod_j \mathcal{N}(\mathbf{u}_j^k|\mathbf{0}, \mathbf{I}) \text{Gam}(\tau|a_0, b_0) \cdot \prod_{\mathbf{j}} \text{Bern}(s_{\mathbf{j}}|\rho_0) \big(s_{\mathbf{j}} \mathcal{N}(w_{\mathbf{j}}|0, \sigma_0^2) + (1 - s_{\mathbf{j}})\delta(w_{\mathbf{j}})\big)$$
$$\cdot \prod_{\mathbf{i} \in \mathcal{F}} \mathcal{N}(y_{\mathbf{i}}|\mathcal{W} \times_1 \left(\mathbf{u}_{i_1}^1\right)^\top \times_2 \ldots \times_K \left(\mathbf{u}_{i_K}^K\right)^\top, \tau^{-1}). \tag{12}$$

3.1.2 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 $\{\mathcal{B}_1, \ldots, \mathcal{B}_{n-1}\}$.

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, 2013) framework to perform one-shot incremental posterior update for $\mathcal{U}$, $\mathcal{W}$ and τ in parallel, avoiding the expensive iterative, alternating updates.

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

$$q_{\text{cur}}(\mathcal{W}, \mathcal{U}, \tau) \propto p(\mathcal{S})\xi(\mathcal{W}, \mathcal{S}) \cdot \prod_{k=1}^{K} \prod_{j=1}^{d_k} \mathcal{N}(\mathbf{u}_j^k|\boldsymbol{\mu}_j^k, \mathbf{V}_j^k) \cdot \mathcal{N}\left(\text{vec}(\mathcal{W})|\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \text{Gam}(\tau|a, b), \tag{13}$$

where $\xi(\mathcal{W}, \mathcal{S})$ is an approximation to the SS prior in (10),

$$\xi(\mathcal{W}, \mathcal{S}) = \prod\nolimits_{\mathbf{j}} \xi_{\mathbf{j}}(w_{\mathbf{j}}, s_{\mathbf{j}}) = \prod\nolimits_{\mathbf{j}} \text{Bern}\left(s_{\mathbf{j}}|c(\rho_{\mathbf{j}})\right) \mathcal{N}(w_{\mathbf{j}}|m_{\mathbf{j}}, \eta_{\mathbf{j}}) \lessapprox p(\mathcal{W}|\mathcal{S}) \tag{14}$$

and c is the sigmoid function, $\lessapprox$ means "approximately proportional to", and $\text{vec}(\cdot)$ denotes vectorization. Contrasting to the joint probability of our model in (12), we can see that the terms other than $p(\mathcal{S})\xi(\mathcal{S}, \mathcal{W})$ in (13) 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.

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

$$\tilde{p}(\mathcal{U}, \mathcal{W}, \tau) = q_{\text{cur}}(\mathcal{U}, \mathcal{W}, \tau) \prod_{\mathbf{i} \in \mathcal{B}_n} p(y_{\mathbf{i}}|\mathcal{U}, \mathcal{W}, \tau) \tag{15}$$

which is an approximation to the joint distribution of all the data (see (12)). 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 $\mathcal{N}(\mathbf{u}_j^k|\boldsymbol{\mu}_j^k, \mathbf{V}_j^k)$. 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, 2001a). We update each ξ_j in parallel. We first divide the marginal posterior by the prior approximation to obtain the calibrated (or context) distribution,

$$q^{\setminus}(w_{\mathbf{j}}, s_{\mathbf{j}}) \propto \frac{q_{\text{cur}}(w_{\mathbf{j}}, s_{\mathbf{j}})}{\xi_j(w_{\mathbf{j}}, s_{\mathbf{j}})} = \text{Bern}(s_{\mathbf{j}}|\rho_0) \mathcal{N}(w_{\mathbf{j}}|\mu_{\mathbf{j}}^{\setminus}, v_{\mathbf{j}}^{\setminus}). \tag{16}$$

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),

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

Algorithm 1 BASS

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 (13) 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 (17) and (18).
6: **end if**
7: **end while**
8: **return** the current posterior $q_{\text{cur}}(\cdot)$.

exponential family. It is well known that moment matching is equivalent to minimizing the KL divergence from $\tilde{p}$ to the approximate posterior.

$$q^*(w_{\mathbf{j}}, s_{\mathbf{j}}) = \text{Bern}(s_{\mathbf{j}}|c(\rho_{\mathbf{j}}^*))\mathcal{N}(w_{\mathbf{j}}|\mu_{\mathbf{j}}^*, v_{\mathbf{j}}^*), \tag{17}$$

where $\rho_{\mathbf{j}}^* = \log\left(\mathcal{N}(\mu_{\mathbf{j}}^{\setminus}|0, \sigma_0^2 + v_{\mathbf{j}}^{\setminus})/\mathcal{N}(\mu_{\mathbf{j}}^{\setminus}|0, v_{\mathbf{j}}^{\setminus})\right) + c^{-1}(\rho_0)$, $\mu_{\mathbf{j}}^* = c(\rho_{\mathbf{j}}^*)\widehat{\mu}_{\mathbf{j}}$, $v_{\mathbf{j}}^* = c(\rho_{\mathbf{j}}^*)\big(\widehat{v}_{\mathbf{j}} + (1 - c(\rho_{\mathbf{j}}^*))\widehat{\mu}_{\mathbf{j}}^2\big)$, where, $\widehat{v}_{\mathbf{j}} = \big(\big(v_{\mathbf{j}}^{\setminus}\big)^{-1} + \sigma_0^{-2}\big)^{-1}$, and $\widehat{\mu}_{\mathbf{j}} = \widehat{v}_{\mathbf{j}}\frac{\mu_{\mathbf{j}}^{\setminus}}{v_{\mathbf{j}}^{\setminus}}$. Finally, we update the prior approximation by

$$\xi_{\mathbf{j}}(\mathbf{w}_{\mathbf{j}}, \mathbf{s}_{\mathbf{j}}) \leftarrow q^*(w_{\mathbf{j}}, s_{\mathbf{j}})/q^{\setminus}(w_{\mathbf{j}}, s_{\mathbf{j}}). \tag{18}$$

To overcome the problem that complex coupling of $\mathcal{U}$ and $\mathcal{W}$ in the likelihood, we first perform conditional moment matching, and then seek to calculate the expected conditional moments. Specifically, let us consider the update of $\mathcal{N}(\text{vec}(\mathcal{W})|\boldsymbol{\mu}, \mathbf{\Sigma})$ in (13). We denote by Θ all the latent random variables in our model, and define $\Theta_{\setminus\mathcal{W}} = \Theta\setminus\{\mathcal{W}\}$. Our key observation is

$$\mathbb{E}_{\tilde{p}}\left[\boldsymbol{\phi}(\mathcal{W})\right] = \mathbb{E}_{\tilde{p}(\Theta_{\setminus\mathcal{W}})}\mathbb{E}_{\tilde{p}(\mathcal{W}|\Theta_{\setminus\mathcal{W}})}\left[\boldsymbol{\phi}(\mathcal{W})|\Theta_{\setminus\mathcal{W}}\right], \tag{19}$$

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

$$\tilde{p}(\mathcal{W}|\Theta_{\setminus\mathcal{W}}) \propto \mathcal{N}(\text{vec}(\mathcal{W})|\mathbf{m}, \text{diag}(\boldsymbol{\eta}))\,\mathcal{N}(\text{vec}(\mathcal{W})|\boldsymbol{\mu}, \mathbf{\Sigma}) \cdot \prod_{\mathbf{i}\in\mathcal{F}} \mathcal{N}\left(y_{\mathbf{i}}|\left(\left(\mathbf{u}_{i_K}^K\right)^\top \otimes \ldots \otimes \left(\mathbf{u}_{i_1}^1\right)^\top\right)\text{vec}(\mathcal{W}), \tau^{-1}\right) \tag{20}$$

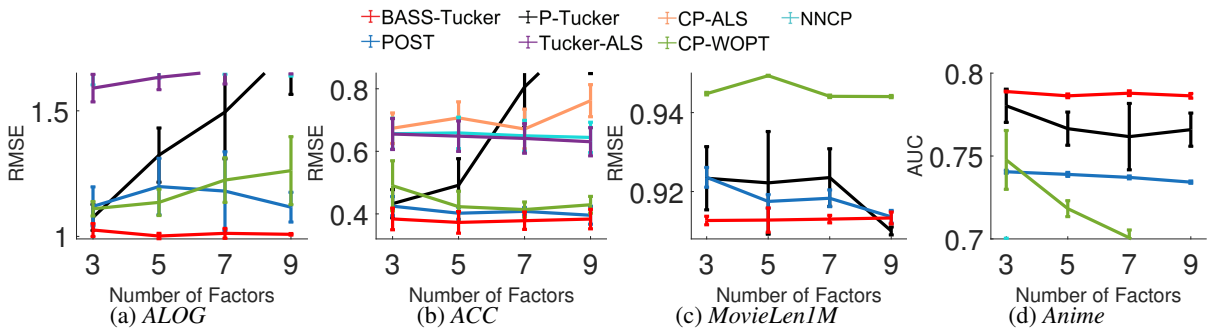
where $\mathbf{m}$ is the concatenation of all $\{m_{\mathbf{j}}\}$ and $\boldsymbol{\eta}$ is the concatenation of all $\{\eta_{\mathbf{j}}\}$ (see (14)). Note that the mean in the Gaussian of $y_{\mathbf{i}}$ is obtained from the tensor algebra (Kolda, 2006). Together this gives another Gaussian distribution, and we can immediately derive the closed-form conditional moments,

$$\mathbb{E}\left[\text{vec}(\mathcal{W})|\Theta_{\setminus\mathcal{W}}\right] = \mathbf{\Omega}^{-1}\left(\sum_{\mathbf{i}}\tau y_{\mathbf{i}}\mathbf{b}_i + \text{diag}\left(\frac{\mathbf{m}}{\boldsymbol{\eta}}\right) + \mathbf{\Sigma}^{-1}\boldsymbol{\mu}\right), \tag{21}$$

$$\mathbb{E}\left[\text{vec}(\mathcal{W})\text{vec}(\mathcal{W})^\top|\Theta_{\setminus\mathcal{W}}\right] = \mathbf{\Omega}^{-1} + \mathbb{E}\left[\text{vec}(\mathcal{W})|\Theta_{\setminus\mathcal{W}}\right]\mathbb{E}\left[\text{vec}(\mathcal{W})|\Theta_{\setminus\mathcal{W}}\right]^\top, \tag{22}$$

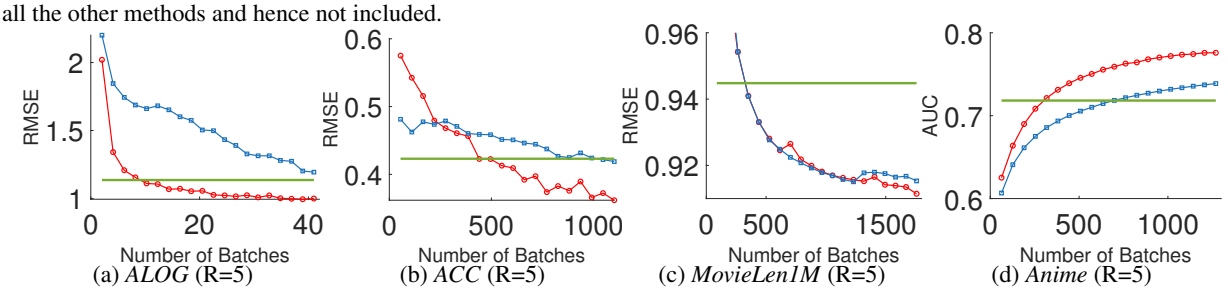
where $\mathbf{\Omega} = \tau\sum_{\mathbf{i}\in\mathcal{F}}\mathbf{b}_{\mathbf{i}}\mathbf{b}_{\mathbf{i}}^\top + \text{diag}(\boldsymbol{\eta}^{-1}) + \mathbf{\Sigma}^{-1}$ and $\mathbf{b}_{\mathbf{i}} = \mathbf{u}_{i_K}^K \otimes \ldots \otimes \mathbf{u}_{i_1}^1$. To obtain the moment, wecompute the expectation of the conditional moment under the marginal blending distribution, $\tilde{p}(\Theta_{\setminus\mathcal{W}})$, using delta method (Oehlert, 1992; Bickel and Doksum, 2015).

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.



(a) *ALOG* (b) *ACC* (c) *MovieLen1M* (d) *Anime*

Figure 1: Predictive performance with different numbers of factors of BASS. The streaming bath size is fixed to 512; 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.



(a) *ALOG* (R=5) (b) *ACC* (R=5) (c) *MovieLen1M* (R=5) (d) *Anime* (R=5)

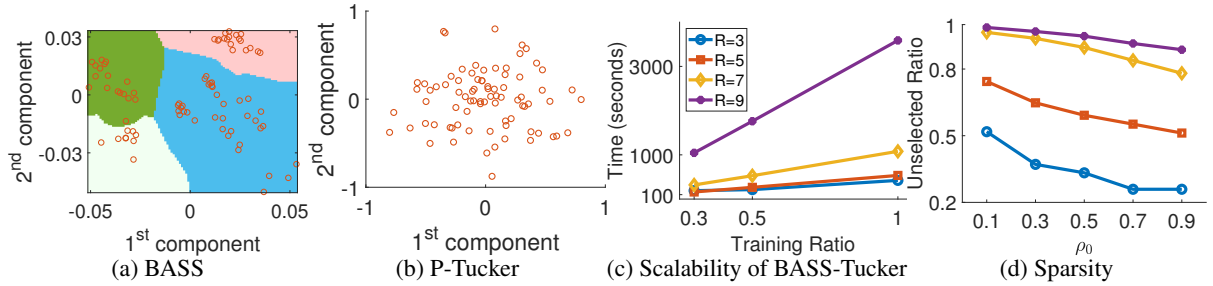
Figure 2: Running prediction accuracy along with the number of processed streaming batches of BASS. The factor number is fixed to 5; The batch size is fixed to 512.

3.1.3 Experiments

Datasets. We evaluated BASS 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.

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., 2011a), 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 (Bader et al., 2015), Tucker decomposition with alternating least square updates.

Settings and Results. We implemented BASS with MATLAB. To estimate a sparse core tensor $\mathcal{W}$, we set $\rho_0 = 0.5$ and $\sigma_0^2 = 1$ (see (10)). We used the original implementation of all the competing approaches and their default settings



(a) BASS (b) P-Tucker (c) Scalability of BASS-Tucker (d) Sparsity

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

(*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 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, $\{3, 5, 7, 9\}$.

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

In addition, we examined the scalability of BASS. On *ACC*, we fixed the batch size to 512, and streamed 30%, 50% and 100% of observed entries to BASS 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 grows linearly in the number of streamed entries, and the factor number R determines the slope. Therefore, BASS enjoys a linear scalability to the data size.

To evaluate if BASS can indeed estimate a sparse core-tensor. We varied ρ_0 from (0.1, 0.9) and ran BASS 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 can effectively achieve sparsity in the streaming setting.

Sparse Structure in Core Tensor Finally, we examined if the estimated sparse core-tensor by BASS 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 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, 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 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, these results have demonstrated that BASS, as compared with standard Tucker decomposition, can potentially discover more interesting patterns and knowledge, and enhance the interpretability.

3.2 Streaming Bayesian Deep Tensor Factorization(SBDT)

Despite the success of existing tensor factorization methods *e.g.,* (Chu and Ghahramani, 2009; Kang et al., 2012; Choi and Vishwanathan, 2014), most of them conduct a multilinear decomposition, and rarely exploit powerful modeling frameworks, like deep neural networks, to capture a variety of complicated interactions in data. More important, for highly expressive, deep factorization, we lack an effective approach to handle streaming data, which are ubiquitous in real-world applications. To address these issues, we propose SBDT, a Streaming Bayesian Deep Tensor factorization method. We first use Bayesian neural networks (NNs) to build a deep tensor factorization model. We assign a spike-and-slab prior over each NN weight to encourage sparsity and to prevent overfitting. We then use the multivariate delta method and moment matching to approximate the posterior of the NN output and calculate the running model evidence,

based on which we develop an efficient streaming posterior inference algorithm in the assumed-density-filtering and expectation propagation framework. Our algorithm provides responsive incremental updates for the posterior of the latent factors and NN weights upon receiving newly observed tensor entries, and meanwhile identify and inhibit redundant/useless weights. We show the advantages of our approach in four real-world applications.

3.2.1 Model

For each tensor entry $\mathbf{i}$, we construct an input $\mathbf{x_i}$ by concatenating all the latent factors associated with $\mathbf{i}$, namely, $\mathbf{x_i} = [\left(\mathbf{u}^1_{i_1}\right)^\top, \ldots, \left(\mathbf{u}^K_{i_K}\right)^\top]^\top$. We assume that there is an unknown mapping between the input factors $\mathbf{x_i}$ and the value of entry $\mathbf{i}$, $f : \mathbb{R}^{\sum_{k=1}^K r_k} \to \mathbb{R}$, which reflects the complex interactions/relationships between the tensor nodes in entry $\mathbf{i}$. Note that CP factorization uses a multilinear mapping. We use an M-layer neural network (NN) to model the mapping f, which are parameterized by M weight matrices $\mathcal{W} = \{\mathbf{W}_1, \ldots, \mathbf{W}_M\}$. Each $\mathbf{W}_m$ is $V_m \times (V_{m-1} + 1)$ where V_m and V_{m-1} are the widths of layer m and $m-1$, respectively. $V_0 = \sum_{k=1}^K r_k$ is the input dimension and $V_M = 1$. We denote the output in each hidden layer m by $\mathbf{h}_m$ $(1 \le m \le M-1)$ and define $\mathbf{h}_0 = \mathbf{x_i}$. We compute each $\mathbf{h}_m = \sigma(\mathbf{W}_m[\mathbf{h}_{m-1}; 1]/\sqrt{V_{m-1}+1})$where $\sigma(\cdot)$ is a nonlinear activation function, *e.g.,* ReLU and $\tanh$. Note that we append a constant feature 1 to introduce the bias terms in the linear transformation, namely the last column in each $\mathbf{W}_m$. For the last layer, we compute the output by $f_{\mathcal{W}}(\mathbf{x_i}) = \mathbf{W}_M[\mathbf{h}_{M-1}; 1]/\sqrt{V_{M-1}+1}$.

Despite their great flexibility, NNs take the risk of overfitting. The larger a network, *i.e.,* with more weight parameters, the easier the network overfits the data. In order to prevent overfitting, we assign a spike-and-slab prior (Ishwaran and Rao, 2005; Titsias and Lázaro-Gredilla, 2011) (that is ideal due to the selective shrinkage effect) over each NN weight to sparsify and condense the network. Specifically, for each weight $w_{mjt} = [\mathbf{W}_m]_{jt}$, we first sample a binary selection indicator s_{mjt} from $p(s_{mij}|\rho_0) = \mathrm{Bern}(s_{mjt}|\rho_0) = \rho_0^{s_{mjt}}(1-\rho_0)^{1-s_{mjt}}$. The weight is then sampled from

$$p(w_{mjt}|s_{mjt}) = s_{mjt}\mathcal{N}(w_{mjt}|0, \sigma_0^2) + (1 - s_{mjt})\delta(w_{mjt}), \tag{23}$$

where $\delta(\cdot)$ is the Dirac-delta function. Hence, the selection indicator s_{mjt} determines the type of prior over w_{mjt}: if s_{mjt} is 1, meaning the weight is useful and active, we assign a flat Gaussian prior with variance σ_0^2 (slab component); if otherwise s_{mjt} is 0, namely the weight is useless and should be deactivated, we assign a spike prior concentrating on 0 (spike component).

Finally, we place a standard normal prior over the factors $\mathcal{U}$. Given the set of observed tensor entries $\mathcal{D} = \{y_{\mathbf{i}_1}, \ldots, y_{\mathbf{i}_N}\}$, the joint probability of our model for continuous data is

$$\begin{aligned} p(\mathcal{U}, \mathcal{W}, \mathcal{S}, \tau) = &\prod_{m=1}^M \prod_{j=1}^{V_m} \prod_{t=1}^{V_{m-1}+1} \mathrm{Bern}(s_{mjt}|\rho_0) \cdot \left(s_{mjt}\mathcal{N}(w_{mjt}|0,\sigma_0^2) + (1-s_{mjt})\delta(w_{mjt})\right) \\ &\cdot \prod_{k=1}^K \prod_{j=1}^{d_k} \mathcal{N}(\mathbf{u}_j^k|\mathbf{0}, \mathbf{I}) \mathrm{Gamma}(\tau|a_0, b_0) \cdot \prod_{n=1}^N \mathcal{N}(y_{\mathbf{i}_n}|f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n}), \tau^{-1}) \end{aligned} \tag{24}$$

where $\mathcal{S} = \{s_{mjt}\}$, and for binary data is

$$\begin{aligned} p(\mathcal{U}, \mathcal{W}, \mathcal{S}) = &\prod_{m=1}^M \prod_{j=1}^{V_m} \prod_{t=1}^{V_{m-1}+1} \mathrm{Bern}(s_{mjt}|\rho_0) \cdot \left(s_{mjt}\mathcal{N}(w_{mjt}|0,\sigma_0^2) + (1-s_{mjt})\delta(w_{mjt})\right) \\ &\cdot \prod_{k=1}^K \prod_{j=1}^{d_k} \mathcal{N}(\mathbf{u}_j^k|\mathbf{0}, \mathbf{I}) \prod_{n=1}^N \Phi\left((2y_{\mathbf{i}_n} - 1) f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})\right) \end{aligned} \tag{25}$$

3.2.2 Online Moment Matching for Posterior Update

We now present our streaming model estimation algorithm. In general, the observed tensor entries are assumed to be streamed in a sequence of small batches, $\{\mathcal{B}_1, \mathcal{B}_2, \ldots\}$. Different batches do not have to include the same number of entries. Upon receiving each batch $\mathcal{B}_t$, we aim to update the posterior distribution of the factors $\mathcal{U}$, the inverse noise variance τ (for continuous data), the selection indicators $\mathcal{S}$ and the neural network weights $\mathcal{W}$, without re-accessing the previous batches $\{\mathcal{B}_j\}_{j<t}$. We exploit the assumed-density-filtering (ADF) framework (**?**), which can be viewed as an online version of expectation propagation (EP) (Minka, 2001a), a general approximate Bayesian inference algorithm. ADF is also based on the incremental version of Bayes' rule (see (4)). It uses a distribution in the exponential family (Wainwright and Jordan, 2008) to approximate the current posterior. When the new data arrive, instead of maximizing a variational ELBO, ADF projects the (unnormalized) blending distribution to the exponential

family to obtain the updated posterior. The projection is done by moment matching, which essentially is to minimize $\mathrm{KL}(\tilde{p}(\boldsymbol{\theta})/Z \| q(\boldsymbol{\theta}))$ where Z is the normalization constant. For illustration, suppose we choose $q(\boldsymbol{\theta})$ to be a fully factorized Gaussian distribution, $q(\boldsymbol{\theta}) = \prod_j q(\theta_j) = \prod_j \mathcal{N}(\theta_j|\mu_j, v_j)$. To update each $q(\theta_j)$, we compute the first and second moments of θ_j w.r.t $\tilde{p}(\boldsymbol{\theta})$, and match a Gaussian distribution with the same moments, namely, $\mu_j = \mathbb{E}_{\tilde{p}}(\theta_j)$ and $v_j = \mathrm{Var}_{\tilde{p}}(\theta_j) = \mathbb{E}_{\tilde{p}}(\theta_j^2) - \mathbb{E}_{\tilde{p}}(\theta_j)^2$.

For our model, we use a fully factorized distribution in the exponential family to approximate the current posterior. When a new batch of data $\mathcal{B}_t$ are received, we sequentially process each observed entry, and perform moment matching to update the posterior of the NN weights $\mathcal{W}$ and associated latent factors. Specifically, let us start with the binary data. We approximate the posterior with

$$q_{\text{cur}}(\mathcal{W}, \mathcal{U}, \mathcal{S}) = \prod_{m=1}^{M} \prod_{j=1}^{V_m} \prod_{t=1}^{V_{m-1}+1} \mathrm{Bern}(s_{mjt}|\rho_{mjt}) \mathcal{N}(w_{mjt}|\mu_{mjt}, v_{mjt}) \cdot \prod_{k=1}^{K} \prod_{j=1}^{d_k} \prod_{t=1}^{r_k} \mathcal{N}(u_{jt}^k|\psi_{kjt}, \nu_{kjt}). \tag{26}$$

Given each entry $\mathbf{i}_n$ in the new batch, we construct the blending distribution, $\tilde{p}(\mathcal{W}, \mathcal{U}, \mathcal{S}) \propto q_{\text{cur}}(\mathcal{W}, \mathcal{U}, \mathcal{S}) \Phi\big((2y_{\mathbf{i}_n} - 1) f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})\big)$. To obtain its moments, we consider the normalizer, *i.e.,* the model evidence under the blending distribution,

$$Z_n = \int q_{\text{cur}}(\mathcal{W}, \mathcal{U}, \mathcal{S}) \Phi\big((2y_{\mathbf{i}_n} - 1) f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})\big) \mathrm{d}\mathcal{W} \mathrm{d}\mathcal{U} \mathrm{d}\mathcal{S}. \tag{27}$$

Under the Gaussian form, according to (Minka, 2001b), we can compute the moments and update the posterior of each NN weight w_{mjt} and each factor associated with $\mathbf{i}_n$— $\{u_{i_{nk}}^k\}_k$ by (6). Note that since the likelihood does not include the binary selection indicators $\mathcal{S}$, their moments are the same as those under q_{cur} and we do not need to update their posterior.

However, a critical issue is that due to the nonlinear coupling of the $\mathcal{U}$ and $\mathcal{W}$ in computing the NN output $f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})$, the exact normalizer is analytically intractable. To overcome this issue, we consider approximating the current posterior of $f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})$ first. We use the multivariate delta method (Oehlert, 1992; Bickel and Doksum, 2015) that expands the NN output at the mean of $\mathcal{W}$ and $\mathcal{U}$ with a Taylor approximation,

$$f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n}) \approx f_{\mathbb{E}[\mathcal{W}]}(\mathbb{E}[\mathbf{x}_{\mathbf{i}_n}]) + \mathbf{g}_n^\top (\boldsymbol{\eta}_n - \mathbb{E}[\boldsymbol{\eta}_n]) \tag{28}$$

where the expectation is under $q_{\text{cur}}(\cdot)$, $\boldsymbol{\eta}_n = \mathrm{vec}(\mathcal{W} \cup \mathbf{x}_{\mathbf{i}_n})$, $\mathbf{g}_n = \nabla f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})|_{\boldsymbol{\eta}_n = \mathbb{E}[\boldsymbol{\eta}_n]}$. Note that $\mathbf{x}_{\mathbf{i}_n}$ is the concatenation of the latent factors associated with $\mathbf{i}_n$. The rationale of the approximation (28) is that the NN output is highly nonlinear and nonconvex in $\mathcal{U}$ and $\mathcal{W}$. Hence, the scale of the output change rate (*i.e.,* gradient) can be much larger than the scale of the posterior variances of $\mathcal{W}$ and $\mathcal{U}$, which are (much) smaller than prior variance 1 . Therefore, we can ignore the second-order term that involves the posterior variances. Note that despite the seemingly simpler structure, our approximation in (28) is still very complex — both the NN output and the Jacobian are highly nonlinear to $\mathcal{W}$, and hence the model expressiveness is not changed. We have also tried the second-order expansion, which, however, is unstable and does not improve the performance.

Based on (28), we can calculate the first and second moments of $f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})$,

$$\begin{aligned} \alpha_n &= \mathbb{E}_{q_{\text{cur}}}[f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})] \approx f_{\mathbb{E}[\mathcal{W}]}(\mathbb{E}[\mathbf{x}_{\mathbf{i}_n}]), \\ \beta_n &= \mathrm{Var}_{q_{\text{cur}}}(f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})) \approx \mathbf{g}_n^\top \mathrm{diag}(\boldsymbol{\gamma}_n) \mathbf{g}_n, \end{aligned} \tag{29}$$

where each $[\boldsymbol{\gamma}_n]_j = \mathrm{Var}_{q_{\text{cur}}}([\boldsymbol{\eta}_n]_j)$. Due to the fully factorized posterior form, we have $\mathrm{cov}(\boldsymbol{\eta}_n) = \mathrm{diag}(\boldsymbol{\eta}_n)$. Note that all the information in the Gaussian posterior (*i.e.,* mean and variance) of $\mathcal{W}$ and $\mathcal{U}$ have been integrated to approximate the moments of the NN output. Now we use moment matching to approximate the current (marginal) posterior of the NN output by $q_{\text{cur}}(f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})) = \mathcal{N}(f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})|\alpha_n, \beta_n)$. Then we compute the running model evidence (27) by

$$\begin{aligned} Z_n &= \mathbb{E}_{q_{\text{cur}}(\mathcal{W}, \mathcal{U}, \mathcal{S})}[\Phi\big((2y_{\mathbf{i}_n} - 1) f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})\big)] = \mathbb{E}_{q_{\text{cur}}(f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n}))}[\Phi\big((2y_{\mathbf{i}_n} - 1) f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})\big)] \\ &\approx \int \mathcal{N}(f_o|\alpha_n, \beta_n) \Phi\big((2y_{\mathbf{i}_n} - 1) f_o\big) \mathrm{d}f_o = \Phi\big(\frac{(2y_{\mathbf{i}_n} - 1)\alpha_n}{\sqrt{1 + \beta_n}}\big), \end{aligned} \tag{30}$$

where we redefine $f_o = f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})$ for simplicity. With the nice analytical form, we can immediately apply (6) to update the posterior for $\mathcal{W}$ and the associated factors in $\mathbf{i}_n$. In light of the NN structure, the gradient can be efficiently calculated via back-propagation, which can be automatically done by many deep learning libraries.

For continuous data, we introduce a Gamma posterior for the inverse noise variance, $q_{\text{cur}}(\tau) = \text{Gamma}(\tau|a, b)$, in addition to the fully factorized posterior for $\mathcal{W}, \mathcal{U}$ and $\mathcal{S}$ as in the binary case. After we use (28) and (29) to obtain the posterior of the NN output, we derive the running model evidence by

$$\begin{aligned} Z_n &= \mathbb{E}_{q_{\text{cur}}(\mathcal{W},\mathcal{U},\mathcal{S},\tau)}[\mathcal{N}\big(y_{\mathbf{i}_n}|f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n}), \tau^{-1}\big)] = \mathbb{E}_{q_{\text{cur}}(f_o)q_{\text{cur}}(\tau)}[\mathcal{N}(y_{\mathbf{i}_n}|f_o, \tau^{-1})] \\ &\approx \mathbb{E}_{q_{\text{cur}}(\tau)}\big[\int \mathcal{N}(f_o|\alpha_n, \beta_n)\mathcal{N}(y_{\mathbf{i}_n}|f_o, \tau^{-1})\mathrm{d}f_o\big] = \mathbb{E}_{q_{\text{cur}}(\tau)}[\mathcal{N}(y_{\mathbf{i}_n}|\alpha_n, \beta_n + \tau^{-1})]. \end{aligned} \tag{31}$$

Next, we use the Taylor expansion again, at the mean of τ, to approximate the Gaussian term inside the expectation,

$$\mathcal{N}(y_{\mathbf{i}_n}|\alpha_n, \beta_n + \tau^{-1}) \approx \mathcal{N}(y_{\mathbf{i}_n}|\alpha_n, \beta_n + \mathbb{E}_{q_{\text{cur}}}[\tau]^{-1}) + (\tau - \mathbb{E}_{q_{\text{cur}}}[\tau])\partial\mathcal{N}(y_{\mathbf{i}_n}|\alpha_n, \beta_n + \tau^{-1})/\partial\tau|_{\tau=\mathbb{E}_{q_{\text{cur}}}[\tau]}. \tag{32}$$

Taking expectation over the Taylor expansion gives

$$Z_n \approx \mathcal{N}(y_{\mathbf{i}_n}|\alpha_n, \beta_n + \mathbb{E}_{q_{\text{cur}}}(\tau)^{-1}) = \mathcal{N}(y_{\mathbf{i}_n}|\alpha_n, \beta_n + b/a). \tag{33}$$

We now can use (6) to update the posterior of $\mathcal{W}$ and the factors associated with the entry. While we can also use more accurate approximations, *e.g.,* the second-order Taylor expansion or quadrature, we found empirically our method achieves almost the same performance.

To update $q_{\text{cur}}(\tau)$, we consider the blending distribution only in terms of the NN output f_o and τ so we have $\tilde{p}(f_o, \tau) \propto q_{\text{cur}}(f_o)q_{\text{cur}}(\tau)\mathcal{N}(y_{\mathbf{i}_n}|f_o, \tau^{-1}) = \mathcal{N}(f_o|\alpha_n, \beta_n)\text{Gamma}(\tau|a, b)\mathcal{N}(y_{\mathbf{i}_n}|f_o, \tau^{-1})$. Then we follow (Wang and Zhe, 2019) to first derive the conditional moments and then approximate the expectation of the conditional moments to obtain the moments. The details are given in the supplementary material. The updated posterior is given by $q^*(\tau) = \text{Gamma}(\tau|a^*, b^*)$, where $a^* = a + \frac{1}{2}$ and $b^* = b + \frac{1}{2}((y_{\mathbf{i}_n} - \alpha_n)^2 + \beta_n)$.

As the spike-and-slab prior for each NN weight w_{mjt} (see (23)) is a mixture prior and does not belong to the exponential family. Hence, we introduce an approximation term,

$$p(w_{mjt}|s_{mjt}) \lesssim A(w_{mjt}, s_{mjt}) = \text{Bern}\big(s_{mjt}|c(\rho_{mjt})\big)\mathcal{N}(w_{mjt}|\mu^0_{mjt}, v^0_{mjt}), \tag{34}$$

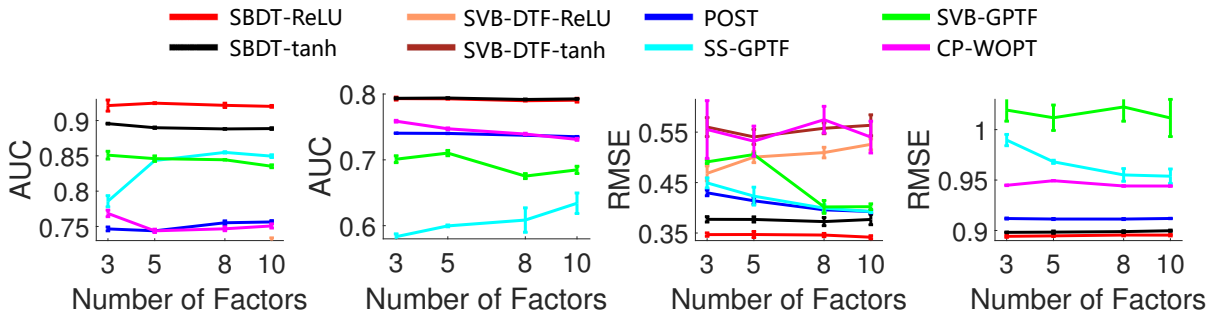
where $\lesssim$ means "approximately proportional to" and $c(x) = 1/(1 + \exp(-x))$. At the beginning, we initialize $v^0_{mjt} = \sigma_0^2$ and μ^0_{mjt} to be a random number generated from a standard Gaussian distribution truncated in $[-\sigma_0, \sigma_0]$; we initialize $\rho_{mjt} = 0$. Obviously, this is a very rough approximation. If we only execute the standard ADF to continuously integrate new entries to update the posterior (see (6)), the prior approximation term will remain the same and never be changed. However, the spike-and-slab prior is critical to sparsify and condense the network, and an inferior approximation will make it noneffective at all. To address this issue, after we process all the entries in the incoming batch, we use EP to update/improve the prior approximation term (34), with which to further update the posterior of the NN weights. Hence, as we continuously process streaming batches of the observed tensor entries, the prior approximation becomes more and more accurate, and thereby can effectively inhibit/deactivate the redundant or useless weights on the fly. The details of the updates are provided in the supplementary material, where we also summarize our streaming inference in Algorithm 2.

3.2.3 Experiments

Datasets. We examined SBDT on four real-world, large-scale datasets. (1) *DBLP* (Du et al., 2018), a binary tensor about bibliography relationships *(author conference, keyword)*, of size $10,000 \times 200 \times 10,000$, including 0.001% nonzero entries. (2) *Anime*(`https://www.kaggle.com/CooperUnion/anime-recommendations-database`), a two-mode tensor depicting binary *(user, anime)* preferences. The tensor contains $1,300,160$ observed entries, of size $25,838 \times 4,066$. (3) *ACC* (Du et al., 2018), a continuous tensor representing the three-way interactions *(user, action, file)*, of size $3,000 \times 150 \times 30,000$, including 0.9% nonzero entries. (4) *MovieLen1M* (`https://grouplens.org/datasets/movielens/`), a two-mode continuous tensor of size $6,040 \times 3,706$, consisting of *(user, movie)* ratings. We have $1,000,209$ observed entries.

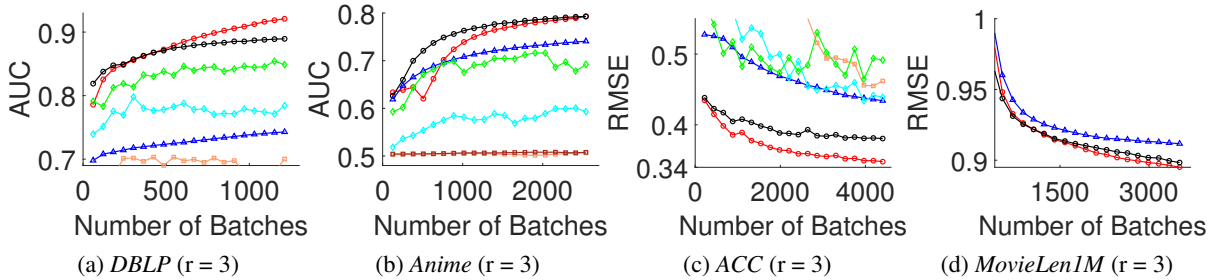
Algorithm 2 Streaming Bayesian Deep Tensor Factorization (SBDT)

Initialize the spike-and-slab prior approximation and multiply it with all the other priors to initialize $q_{\text{cur}}(\cdot)$.
while a new batch of observed tensor entries $\mathcal{B}_t$ arrives **do**
  for each entry $\mathbf{i}_n$ in $\mathcal{B}_t$ **do**
    Approximate the running model evidence with (30) (binary data) or (33) (continuous data).
    Update the posterior for $\mathcal{W}$ and the associated factors $\{u^k_{i_{nj}}\}_{k,j}$ with (6).
    if continuous data **then**
      Update the posterior for the inverse noise variance τ via conditional moment matching.
    end if
    Update the spike-and-slab prior approximation with standard EP.
  end for
end while
return the current posterior $q_{\text{cur}}(\cdot)$.



(a) *DBLP* (b) *Anime* (c) *ACC* (d) *MovieLen1M*

Figure 4: Predictive performance with different numbers of factors of SBDT. The streaming bath size is fixed to 256; The results are averaged over 5 runs.



(a) *DBLP* (r = 3) (b) *Anime* (r = 3) (c) *ACC* (r = 3) (d) *MovieLen1M* (r = 3)

Figure 5: Running prediction accuracy along with the number of processed streaming batches of SBDT. The batch size was fixed to 256.

Competing methods. We compared with the following baselines. (1) POST (Du et al., 2018), the state-of-the-art probabilistic streaming tensor decomposition algorithm based on the CP model. It uses streaming variational Bayes (SVB) (Broderick et al., 2013) to perform mean-field posterior updates upon receiving new entries. (2) SVB-DTF, streaming deep tensor factorization implemented with SVB. (3) SVB-GPTF, the streaming version of the Gaussian process(GP) nonlinear tensor factorization (Zhe et al., 2016b), implemented with SVB. Note that similar to NNs, the ELBO in SVB for GP factorization is intractable and we used stochastic optimization. (4) SS-GPTF, the streaming GP factorization implemented with the recent streaming sparse GP approximations (Bui et al., 2017). It uses SGD to optimize another intractable ELBO. (5) CP-WOPT (Acar et al., 2011b), a scalable static CP factorization algorithm implemented with gradient-based optimization.

Results. We first evaluated the prediction accuracy after all the (accessible) entries are processed. To do so, we sequentially fed the training entries into every method, each time a small batch. We then evaluated the predictive performance on the test entries. We examined the root-mean-squared-error (RMSE) and area under ROC curves (AUC) for continuous and binary data, respectively. We ran the static factorization algorithm CP-WOPT on the entire training set. On *DBLP* and *ACC*, we used the same split of the training and test entries as in (Du et al., 2018), including 320K

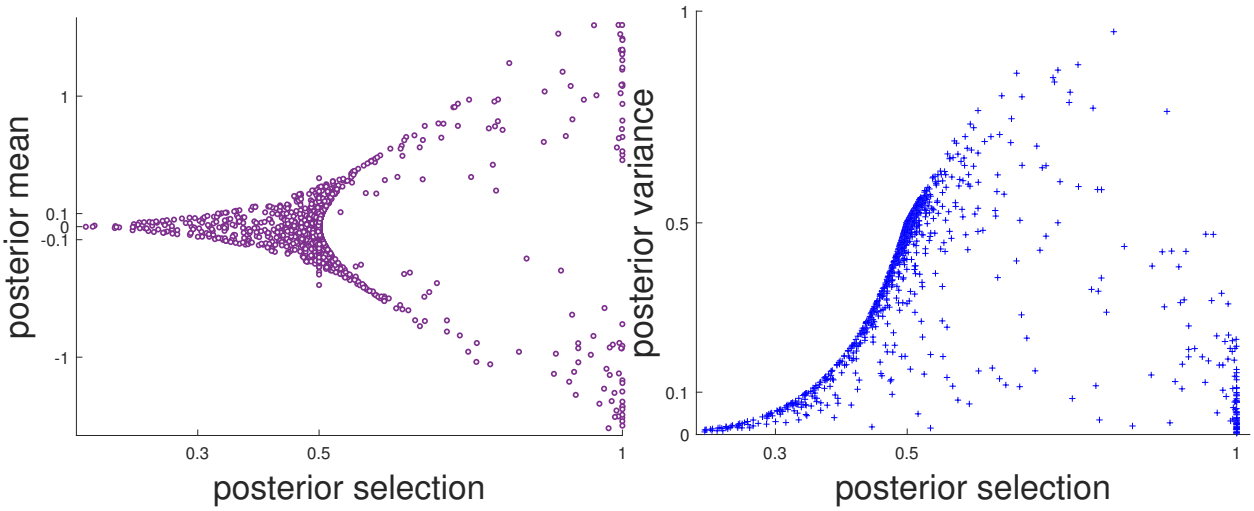


Figure 6: Posterior selection probability $c(\rho^*_{mjt})$ *vs.* the posterior mean and variance of each NN weight w_{mjt} of SBDT

and 1M training entries for *DBLP* and *ACC* respectively, and 100K test entries for both. On *Anime* and *MovieLen1M*, we randomly split the observed entries into 90% for training and 10% for test. For both datasets, the number of training entries is around one million. For each streaming approach, we randomly shuffled the training entries and then partitioned them into a stream of batches. On each dataset, we repeated the test for 5 times and calculated the average of RMSEs/AUCs and their standard deviations. For CP-WOPT, we used a different random initialization for each test. The results are reported in Fig. 4. As we can see, SBDT (with both $\tanh$ and ReLU) consistently outperforms all the competing approaches in all the cases and mostly by a large margin.

Prediction On the Fly. Next, we evaluated the dynamic performance. We randomly generated a stream of training batches from each dataset, upon which we ran each algorithm and examined the prediction accuracy after processing each batch. We set the batch size to 256 and tested with the number of latent factors $r = 3$ and $r = 8$. The running RMSE/AUC of each method are reported in Fig. 2. Note that some curves are missing or partly missing because the performance of the corresponding methods are much worse than all the other ones. In general, nearly all the methods improved the prediction accuracy with more and more batches, showing increasingly better factor estimations. However, SBDT always obtained the best AUC/RMSE on the fly, except at the beginning stage on *Anime* and *MovieLen1M* ($r = 8$). The trends of SBDT and POST are much smoother than that of SVB-DTF and SVB/SS-GPTF, which might again because the stochastic updates are unstable and unreliable. Note that in Fig. 5b, SVB-DTF has running AUC steadily around 0.5, implying that SVB actually failed to effectively update the posterior.

Network Sparsity. Finally, we looked into the estimated posterior distribution of the NN weights. We set the number of latent factors to 8 and streaming batch-size to 256, and ran SBDT on *DBLP* dataset with ReLU. In Fig. 6, we show the posterior selection probability $c(\rho^*_{mjt})$ *vs.* the posterior mean μ^*_{mjt} for each weight w_{mjt}, and $c(\rho^*_{mjt})$ *vs.* the posterior variance v^*_{mjt} for each weight. As we can see, when the posterior selection probability is less than 0.5, *i.e.,* the weight w_{mjt} is likely to be useless/redundant, both its posterior mean and variance are small and close to 0. The more the posterior selection probability approaches 0, the closer μ^*_{mjt} and v^*_{mjt} to 0, exhibiting a shrinkage effect. Thereby the corresponding weight w_{mjt} is inhibited or deactivated. By contrast, when the posterior selection probability is bigger than 0.5, the posterior mean and variance have much larger scales and ranges, implying that the corresponding weight is active and estimated from data freely. Therefore, SBDT can effectively inhibit redundant/useless NN weights to prevent overfitting during the factorization.

3.3 Bayesian Continuous-Time Tucker decomposition(BCTT)

Tensor decomposition is a dominant framework for multiway data analysis and prediction. Although practical data often contains timestamps for the observed entries, existing tensor decomposition approaches overlook or under-use this valuable temporal information. They either drop the timestamps or bin them into crude steps and hence ignore the temporal dynamics within each step or use simple parametric time coefficients. To overcome these limitations, we propose Bayesian Continuous-Time Tucker Decomposition (BCTT). We model the tensor-core of the classical Tucker

decomposition as a time-varying function, and place a Gaussian process prior to flexibly estimate all kinds of temporal dynamics. In this way, our model maintains the interpretability while is flexible enough to capture various complex temporal relationships between the tensor nodes. For efficient and high-quality posterior inference, we use the stochastic differential equation (SDE) representation of temporal GPs to build an equivalent state-space prior, which avoids huge kernel matrix computation and sparse/low-rank approximations. We then use Kalman filtering, RTS smoothing, and conditional moment matching to develop a scalable message-passing inference algorithm. We show the advantage of our method in simulation and several real-world applications.

3.3.1 SDE Representation of Temporal GP

In the literature of stochastic differential equations (SDEs) (Särkkä et al., 2006; Oksendal, 2013), it is known that the solution of linear SDEs are Gaussian processes on time, namely, temporal GPs. From the other side, for temporal GPs with certain stationary kernels, we can construct an equivalent Linear Time-Invariant (LTI) SDE through spectral analysis (Hartikainen and Särkkä, 2010). Take the Matérn kernel (8) with $\nu = m + \frac{1}{2}$ (where $m \in \mathbb{N}$) as an example. We can obtain its power spectral density as $S(\omega) = P(\mathrm{i}\omega) q_c P(-\mathrm{i}\omega)$, where $P(\mathrm{i}\omega) = \frac{1}{(\beta + \mathrm{i}\omega)^{m+1}}$, i indicates the imaginary part, $\beta = \sqrt{2\nu}/l$, and $q_c = \frac{2\sigma^2 \pi^{1/2} \beta^{2m+1} \Gamma(m+1)}{\Gamma(m+1/2)}$. This is equivalent to feeding a white noise process with diffusion q_c into a system, who transfers the signal with $P(\mathrm{i}\omega)$ to generate the output. Via inverse Fourier transform, we know the output process is the solution of the SDE

$$\frac{\mathrm{d}^{m+1} f(t)}{\mathrm{d}t^{m+1}} + a_m \frac{\mathrm{d}^m f(t)}{\mathrm{d}t^m} + \ldots + a_0 f(t) = \xi(t), \tag{35}$$

where $\xi(t)$ is the white noise process with diffusion q_c, and $a_0, \ldots, a_m$ are the coefficients of the zeroth, first, till m-th term in the polynomial of $P(\mathrm{i}\omega)$'s denominator. This can be further written as an LTI-SDE, in which we define the state as $\mathbf{y}(t) = \left(f(t), \frac{\mathrm{d}f(t)}{\mathrm{d}t}, \ldots, \frac{\mathrm{d}f^m(t)}{\mathrm{d}t} \right)^\top$, and

$$\frac{\mathrm{d}\mathbf{y}(t)}{\mathrm{d}t} = \mathbf{F}\mathbf{x}(t) + \mathbf{L}\xi(t), \tag{36}$$

where

$$\mathbf{F} = \begin{pmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & 0 & 1 \\ -a_0 & \ldots & -a_{m-1} & -a_m \end{pmatrix}, \quad \mathbf{L} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

In general, although we cannot guarantee the power spectrum $S(\omega)$ of the kernel has a polynomial form in the denominator, we can apply Taylor approximation on $1/S(\omega)$ to construct an approximately equivalent LTI-SDE.

3.3.2 Model

We model each element of the tensor-core $\mathcal{W}$ in the Tucker decomposition (2) as a time-varying (or trend) function so as to capture the temporal interactions across all the factor combinations. In order to flexibly estimate a variety of complex temporal variations, we place a GP prior over each element, $w_{\mathbf{r}}(t) \sim \mathcal{GP}\left(0, \kappa(t, t')\right)$ where $\mathbf{r} = (r_1, \ldots, r_K)$. Given the observed tensor entry values and time points, $\mathcal{D} = \{(\mathbf{i}_1, t_1, y_1), \ldots, (\mathbf{i}_N, t_N, y_N)\}$, we have a multi-variate Gaussian prior over the values of $w_{\mathbf{r}}(\cdot)$ at the observed timestamps, $p(\mathbf{w}_{\mathbf{r}}) = \mathcal{N}(\mathbf{w}_{\mathbf{r}} | \mathbf{0}, \mathbf{K}_{\mathbf{r}})$,where $\mathbf{w}_{\mathbf{r}} = [w_{\mathbf{r}}(t_1), \ldots, w_{\mathbf{r}}(t_N)]^\top$, $\mathbf{K}_{\mathbf{r}}$ is the $N \times N$ kernel matrix on the time points and each $[\mathbf{K}_r]_{n,n'} = \kappa(t_n, t_{n'})$. Given $\mathcal{W}(t_n) = \{w_{\mathbf{r}}(t_n)\}_{\mathbf{r}}$, we sample the observed entry value from $p(y_n | \mathcal{W}(t_n), \mathcal{U}) = \mathcal{N}\left(y_n | \mathrm{vec}\left(\mathcal{W}(t_n)\right)^\top \left(\mathbf{u}^1_{i_{n_1}} \otimes \ldots \otimes \mathbf{u}^K_{i_{n_K}} \right), \tau^{-1} \right)$,where τ is the inverse variance, for which we place a Gamma prior, $p(\tau) = \mathrm{Gam}(\tau | b_0, c_0)$. Here we only consider continuous observations. However, it is straightforward to extend our model and inference to other types of entry values. We further place a standard Gaussian prior over the latent factors $p(\mathcal{U}) = \prod_{k=1}^K \prod_{j=1}^{d_k} \mathcal{N}(\mathbf{u}^k_j | \mathbf{0}, \mathbf{I})$. The joint probability is

$$p(\mathcal{U}, \{\mathbf{w}_{\mathbf{r}}\}_{\mathbf{r}}, \tau, \mathbf{y}) = p(\mathcal{U}) p(\tau) \cdot \prod_{\mathbf{r}=(1,\ldots,1)}^{(R_1,\ldots,R_K)} \mathcal{N}(\mathbf{w}_{\mathbf{r}} | \mathbf{0}, \mathbf{K}_{\mathbf{r}}) \cdot \prod_{n=1}^N p(y_n | \mathcal{W}(t_n), \mathcal{U}). \tag{37}$$

However, a straightforward formulation as in (59) brings in severe computational challenges. The joint probability includes many multivariate Gaussian distributions, *i.e.,* $\mathcal{N}(\mathbf{w_r}|\mathbf{0}, \mathbf{K_r})$. When the number of time points N is large, the calculation of each kernel matrix $\mathbf{K_r}$ and its inverse (in the distribution) is extremely expensive or even infeasible ($\mathcal{O}(N^3)$ time complexity). To overcome this hurdle, we have to seek for various sparse GP approximations (Quiñonero-Candela and Rasmussen, 2005), which essentially use aggressive low-rank structures to approximate the kernel matrices.

To prevent sparse/low-rank approximations (which can be of low quality), we use SDEs to formulate our model so as to perform full GP inference with a linear cost in N. Specifically, we observe that each $w_{\mathbf{r}}(t)$ is actually a temporal GP. Therefore, we can construct an equivalent LTI-SDE. For convenience, we use the Matérn kernel with $\nu = 3/2 = 1+1/2$ for illustration. According to (36), for each $w_{\mathbf{r}}(t)$, we define a state $\boldsymbol{\gamma}_{\mathbf{r}}(t) = (w_{\mathbf{r}}, \frac{\mathrm{d}w_{\mathbf{r}}}{\mathrm{d}t})^\top$, and the SDE is

$$\frac{\mathrm{d}\boldsymbol{\gamma}_{\mathbf{r}}(t)}{\mathrm{d}t} = \mathbf{F}\boldsymbol{\gamma}_{\mathbf{r}} + \mathbf{L}\xi(t), \tag{38}$$

where $\mathbf{F} = [0, 1; -\beta^2, -2\beta]$, $\mathbf{L} = [0; 1]$, and the diffusion of the white noise $\xi(t)$ is $q_c = 4\beta^3\sigma^2$. The benefit of the LTI-SDE representation is that its discrete form (on $t_1, \ldots, t_N$) is a Gaussian Markov chain,

$$p(\boldsymbol{\gamma}_{\mathbf{r}}(t_1)) = \mathcal{N}(\boldsymbol{\gamma}_{\mathbf{r}}(t_1)|\mathbf{0}, \mathbf{P}_\infty), \tag{39}$$

$$p(\boldsymbol{\gamma}_{\mathbf{r}}(t_{n+1})|\boldsymbol{\gamma}_{\mathbf{r}}(t_n)) = \mathcal{N}\left(\boldsymbol{\gamma}_{\mathbf{r}}(t_{n+1})|\mathbf{A}_n\boldsymbol{\gamma}_{\mathbf{r}}(t_n), \mathbf{Q}_n\right) \tag{40}$$

where $\mathbf{P}_\infty = [\sigma^2, 0; 0, \beta^2\sigma^2]$ is the stationary covariance calculated by solving the matrix Riccati equation (Lancaster and Rodman, 1995), $\Delta_n = t_{n+1} - t_n$ is the time difference, $\mathbf{A}_n = \exp(\mathbf{F}\Delta_n)$, and $\mathbf{Q}_n = \mathbf{P}_\infty - \mathbf{A}_n\mathbf{P}_\infty\mathbf{A}_n^\top$.

To represent all the temporal GPs in our model, we define a joint state $\overline{\boldsymbol{\gamma}}(t)$ as the concatenation of all $\{\boldsymbol{\gamma}_{\mathbf{r}}(t)\}_{\mathbf{r}}$. Accordingly, the discrete form of the SDE for $\overline{\boldsymbol{\gamma}}(t)$ follows

$$p(\overline{\boldsymbol{\gamma}}_1) = \mathcal{N}(\overline{\boldsymbol{\gamma}}_1|\mathbf{0}, \boldsymbol{\Sigma}), \tag{41}$$

$$p(\overline{\boldsymbol{\gamma}}_{n+1}|\overline{\boldsymbol{\gamma}}_n) = \mathcal{N}(\overline{\boldsymbol{\gamma}}_{n+1}|\mathbf{B}_n\overline{\boldsymbol{\gamma}}_n, \mathbf{C}_n), \tag{42}$$

where $\overline{\boldsymbol{\gamma}}_n \triangleq \overline{\boldsymbol{\gamma}}(t_n)$, $\boldsymbol{\Sigma} = \mathrm{diag}(\mathbf{P}_\infty, \ldots, \mathbf{P}_\infty)$, $\mathbf{B}_n = \mathrm{diag}(\mathbf{A}_n, \ldots, \mathbf{A}_n)$, and $\mathbf{C}_n = \mathrm{diag}(\mathbf{Q}_n, \ldots, \mathbf{Q}_n)$. As we can see, this is essentially a state-space prior over the collection of states $\{\overline{\boldsymbol{\gamma}}_n\}$. To extract the tensor-core $\mathcal{W}(t)$, we can use a sparse $\overline{R} \times 2\overline{R}$ matrix,

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & & & & \\ & & 1 & 0 & & \\ & & & & \ddots & \\ & & & & 1 & 0 \end{pmatrix},$$

to obtain $\mathrm{vec}(\mathcal{W}(t)) = \mathbf{H} \cdot \overline{\boldsymbol{\gamma}}(t)$, where $\overline{R}$ is the size of the tensor-core, $\overline{R} = \prod_{k=1}^K R_k$.

Now, we replace the multivariate Gaussians in (59) by the state space prior in (42), and write the joint probability as

$$p(\mathcal{U}, \{\overline{\boldsymbol{\gamma}}_n\}, \tau, \mathbf{y}) = p(\mathcal{U})p(\tau) \cdot p(\overline{\boldsymbol{\gamma}}_1) \prod_{n=1}^{N-1} p(\overline{\boldsymbol{\gamma}}_{n+1}|\overline{\boldsymbol{\gamma}}_n) \prod_{n=1}^{N} \mathcal{N}\left(y_n|\left(\mathbf{H}\overline{\boldsymbol{\gamma}}_n\right)^\top \left(\mathbf{u}^1_{i_{n_1}} \otimes \ldots \otimes \mathbf{u}^K_{i_{n_K}}\right), \tau^{-1}\right). \tag{43}$$

Since each state $\overline{\boldsymbol{\gamma}}_n$ is only dependent on its previous state $\overline{\boldsymbol{\gamma}}_{n-1}$ (Markov property), we no longer need to compute a giant $N \times N$ covariance matrix nor need low-rank approximations. The state space prior enables us to develop an efficient, linear GP inference algorithm, as presented in the next section.

3.3.3 Inference by Efficient Filtering, Smoothing and Message Passing

The exact posterior of our model is infeasible to calculate, because the likelihood of each data point n (arising from the entry-wise Tucker decomposition (2)) couples the relevant latent factors $\{\mathbf{u}^1_{i_{n_1}}, \ldots \mathbf{u}^K_{i_{n_K}}\}$ and state $\overline{\boldsymbol{\gamma}}(t_n)$. To address this issue, we introduce Gaussian-Gamma likelihood approximations, and based on Kalman filtering (KF) (Kalman, 1960) and Rauch-Tung-Striebel (RTS) smoothing (Rauch et al., 1965) we develop an efficient message-passing algorithm in the expectation propagation (EP) framework (Minka, 2001a). See Fig.7.

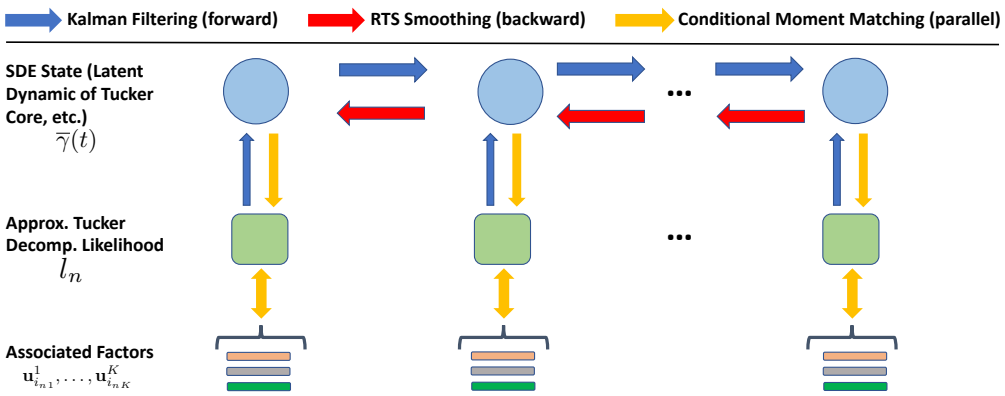


Figure 7: Graphical illustration of the message-passing inference algorithm for BCTT.

Specifically, we approximate each data likelihood with

$$\begin{aligned}&\mathcal{N}\left(y_n|\left(\mathbf{H}\overline{\boldsymbol{\gamma}}_n\right)^\top\left(\mathbf{u}^1_{i_{n_1}}\otimes\ldots\otimes\mathbf{u}^K_{i_{n_K}}\right),\tau^{-1}\right)\approx\ell_n\\&\triangleq Z_n\cdot\prod\nolimits_{k=1}^K\mathcal{N}\left(\mathbf{u}^k_{i_{n_k}}|\mathbf{m}^{k,n}_{i_{n_k}},\mathbf{V}^{k,n}_{i_{n_k}}\right)\cdot\text{Gam}\left(\tau|b_n,c_n\right)\cdot\mathcal{N}\left(\mathbf{H}\overline{\boldsymbol{\gamma}}_n\mid\boldsymbol{\beta}_n,\mathbf{S}_n\right),\end{aligned}\tag{44}$$

where Z_n is a normalization term (it will be canceled during inference). Hence we obtain the approximate posterior by

$$\begin{aligned}q(\mathcal{U},\{\overline{\boldsymbol{\gamma}}_n\},\tau)\propto&\prod\nolimits_{k=1}^K\prod\nolimits_{j=1}^{d_k}\mathcal{N}(\mathbf{u}^k_j|\mathbf{0},\mathbf{I})\text{Gam}(\tau|b_0,c_0)\prod_{n=1}^N\prod_{k=1}^K\mathcal{N}\left(\mathbf{u}^k_{i_{n_k}}|\mathbf{m}^{k,n}_{i_{n_k}},\mathbf{V}^{k,n}_{i_{n_k}}\right)\text{Gam}\left(\tau|b_n,c_n\right)\\&\cdot p(\overline{\boldsymbol{\gamma}}_1)\mathcal{N}\left(\mathbf{H}\overline{\boldsymbol{\gamma}}_1\mid\boldsymbol{\beta}_1,\mathbf{S}_1\right)\prod_{n=1}^{N-1}p(\overline{\boldsymbol{\gamma}}_{n+1}|\overline{\boldsymbol{\gamma}}_n)\mathcal{N}\left(\mathbf{H}\overline{\boldsymbol{\gamma}}_n\mid\boldsymbol{\beta}_n,\mathbf{S}_n\right).\end{aligned}\tag{45}$$

The parameters of the approximation terms, including $\{\mathbf{m}^{k,n}_{i_{n_k}},\mathbf{V}^{k,n}_{i_{n_k}},b_n,c_n,\boldsymbol{\beta}_n,\mathbf{S}_n\}$, will be updated and estimated during the message-passing inference. After that, we can obtain the (approximate) posterior of latent factors and noise inverse variance τ by merging relevant terms, $q(\mathbf{u}^k_j)\propto\mathcal{N}(\mathbf{u}^k_j|\mathbf{0},\mathbf{I})\prod_{i_{n_k}=j}\mathcal{N}(\mathbf{u}^k_{i_{n_k}}|\mathbf{m}^{k,n}_{i_{n_k}},\mathbf{V}^{k,n}_{i_{n_k}})$, and $q(\tau)\propto\text{Gam}(\tau|b_0,c_0)\prod_{n=1}^N\text{Gam}(\tau|b_n,c_n)$, which have closed forms, *i.e.,* Gaussian or Gamma.

However, the posterior of the states $\overline{\boldsymbol{\gamma}}_n$ is not easy to obtain, because $\overline{\boldsymbol{\gamma}}_n$ are chained in the state space prior. Thanks to the Gaussian term $\mathcal{N}\left(\mathbf{H}\overline{\boldsymbol{\gamma}}_n\mid\boldsymbol{\beta}_n,\mathbf{S}_n\right)$ introduced in (44) — by symmetry, we can view it as $\mathcal{N}\left(\boldsymbol{\beta}_n|\mathbf{H}\overline{\boldsymbol{\gamma}}_n,\mathbf{S}_n\right)$ — a Gaussian likelihood (emission) of the virtual observation $\boldsymbol{\beta}_n$ following the state space prior of each $\overline{\boldsymbol{\gamma}}_n$ (see the third line of (45)). Therefore, we can apply the standard KF in a forward pass and RTS smoothing in a backward pass to efficiently compute all the marginal posteriors $q(\overline{\boldsymbol{\gamma}}_n)$ and $q(\overline{\boldsymbol{\gamma}}_n,\overline{\boldsymbol{\gamma}}_{n+1})$, with a linear cost in N (*i.e.,* $\mathcal{O}(N)$ complexity). Note that the standard KF and RTS can only be used for Gaussian emissions but they give exact results. For non-Gaussian likelihoods, we have to combine with extra approximations, such as extended KF and unscented KF (Särkkä, 2013), which can be unstable and more costly.

To optimize the approximation terms in each ℓ_n (see (44)), we develop a message- passing algorithm in the EP framework. Specifically, at each step, given all ℓ_n, we first run KF and RST smoothing to calculate the posterior of each sate $q(\overline{\boldsymbol{\gamma}}_n)$. The calculation is actually the standard message passing in chain graphical models (Bishop, 2006). Each Gaussian term $\mathcal{N}\left(\mathbf{H}\overline{\boldsymbol{\gamma}}_n\mid\boldsymbol{\beta}_n,\mathbf{S}_n\right)$ is the initial message sent from data point n to the state $\overline{\boldsymbol{\gamma}}_n$, then we conduct KF to compute the message from each $\overline{\boldsymbol{\gamma}}_n$ to $\overline{\boldsymbol{\gamma}}_{n+1}$ (forward pass), and then RTS smoothing the messages from $\overline{\boldsymbol{\gamma}}_{n+1}$ to $\overline{\boldsymbol{\gamma}}_n$ (backward pass). The posterior $q(\overline{\boldsymbol{\gamma}}_n)$ is obtained by aggregating all the messages sent to $\overline{\boldsymbol{\gamma}}_n$ (*i.e.,* those from $\overline{\boldsymbol{\gamma}}_{n-1}$, $\overline{\boldsymbol{\gamma}}_{n+1}$ and data point n), which ends up with a Gaussian distribution.

Next, we use the state posteriors $\{q(\overline{\boldsymbol{\gamma}}_n)\}$ to update the likelihood approximation terms in $\{\ell_n\}$ via EP. Specifically, for each data point n, we obtain a calibrated distribution by dividing the global posterior by the current approximation,

$$q^{\backslash n}(\boldsymbol{\Theta}_n)\propto\frac{q(\overline{\boldsymbol{\gamma}}_n)q(\tau)\prod_{k=1}^K q(\mathbf{u}^k_{i_{n_k}})}{\ell_n}=\mathcal{N}(\boldsymbol{\eta}_n|\boldsymbol{\beta}^{\backslash n},\mathbf{S}^{\backslash n})\cdot\prod\nolimits_{k=1}^K\mathcal{N}\left(\mathbf{u}^k_{i_{n_k}}|\mathbf{m}^{k,\backslash n}_{i_{n_k}},\mathbf{V}^{k,\backslash n}_{i_{n_k}}\right)\text{Gam}\left(\tau|b^{\backslash n},c^{\backslash n}\right)\tag{46}$$

where $\boldsymbol{\eta}_n = \mathbf{H}\overline{\boldsymbol{\gamma}}_n = \text{vec}\left(\mathcal{W}(t_n)\right)$, and $\boldsymbol{\Theta}_n = \{\boldsymbol{\eta}_n, \{\mathbf{u}^k_{i_{n_k}}\}_k, \tau\}$ are all the random variables present in the n-th likelihood. The calibrated distribution integrates the information from all the other data points, *i.e.,* the context. To update the terms in ℓ_n, we construct a tilted distribution,

$$\tilde{p}(\boldsymbol{\Theta}_n) \propto q^{\setminus n}(\boldsymbol{\Theta}_n) \cdot \mathcal{N}\left(y_n | \boldsymbol{\eta}_n^\top \left(\mathbf{u}^1_{i_{n_1}} \otimes \ldots \otimes \mathbf{u}^K_{i_{n_K}}\right), \tau^{-1}\right). \tag{47}$$

We aim to project the tilted distribution back to our approximation family (exponential family), to obtain

$$q^*(\boldsymbol{\Theta}_n) = \mathcal{N}(\boldsymbol{\eta}_n | \boldsymbol{\beta}^*_n, \mathbf{S}^*_n) \cdot \prod\nolimits_{k=1}^K \mathcal{N}\left(\mathbf{u}^k_{i_{n_k}} | \mathbf{m}^{k,*}_{i_{n_k}}, \mathbf{V}^{k,*}_{i_{n_k}}\right) \text{Gam}\left(\tau | b^*_n, c^*_n\right) \tag{48}$$

from which we update ℓ_n terms via dividing the calibrated distribution back,

$$\ell_n \leftarrow \frac{q^*(\boldsymbol{\Theta}_n)}{q^{\setminus n}(\boldsymbol{\Theta}_n)}. \tag{49}$$

The projection essentially is to minimize the Kullback-Leibler divergence from $\tilde{p}(\boldsymbol{\Theta}_n)$ to $q^*(\boldsymbol{\Theta}_n)$, which can be done by moment matching. For example, the Gaussian posterior of $\boldsymbol{\eta}_n$ in (48) needs two moments — the expectation of $\boldsymbol{\eta}_n$ and $\boldsymbol{\eta}_n\boldsymbol{\eta}_n^\top$. So we need to compute them under the tilted distribution so as to match the parameters of $q^*(\boldsymbol{\eta}_n)$, namely $\boldsymbol{\beta}^*_n = \mathbb{E}_{\tilde{p}}\left[\boldsymbol{\eta}_n\right]$ and $\mathbf{S}^*_n = \mathbb{E}_{\tilde{p}}\left[\boldsymbol{\eta}_n\boldsymbol{\eta}_n^\top\right] - \mathbb{E}_{\tilde{p}}\left[\boldsymbol{\eta}_n\right]\mathbb{E}_{\tilde{p}}\left[\boldsymbol{\eta}\right]^\top$.

The standard EP assumes the moment matching is computationally tractable. However, this is not the case in our model. Since $\boldsymbol{\eta}_n$ and the latent factors are coupled in the product and Kronecker product in the tilted distribution (47), we do not have a closed form of the moments. To address this problem, we use the idea of the conditional moment matching (Wang and Zhe, 2019). Take $\boldsymbol{\eta}_n$ as an example. Denote the required moments by $\boldsymbol{\phi}(\boldsymbol{\eta}_n) = \left(\boldsymbol{\eta}_n, \boldsymbol{\eta}_n\boldsymbol{\eta}_n^\top\right)$. The key observation is that we can decompose the expectation into a nested structure,

$$\mathbb{E}_{\tilde{p}}\left[\boldsymbol{\phi}(\boldsymbol{\eta}_n)\right] = \mathbb{E}_{\tilde{p}(\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n})}\left[\mathbb{E}_{\tilde{p}(\boldsymbol{\eta}_n|\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n})}\left[\boldsymbol{\phi}(\boldsymbol{\eta})|\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}\right]\right] \tag{50}$$

where $\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n} \triangleq \boldsymbol{\Theta}_n \setminus \{\boldsymbol{\eta}_n\}$. Therefore, we can compute the conditional moment first, *i.e.,* the inner expectation, and then take expectation over the conditional moments, *i.e.,* the outer-expectation. Given all the other variables $\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}$ fixed, the conditioned tilted distribution $\tilde{p}(\boldsymbol{\eta}_n|\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n})$ is simply a Gaussian. Hence, the conditional moment is easy to obtain,

$$\mathbb{E}\left[\boldsymbol{\eta}_n|\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}\right] = \boldsymbol{\Sigma}_n\left(\left(\mathbf{S}^{\setminus n}\right)^{-1}\boldsymbol{\beta}^{\setminus n} + \tau y_n \mathbf{v}_n\right), \tag{51}$$

$$\mathbb{E}\left[\boldsymbol{\eta}_n\boldsymbol{\eta}_n^\top|\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}\right] = \boldsymbol{\Sigma}_n + \mathbb{E}\left[\boldsymbol{\eta}_n|\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}\right]\mathbb{E}\left[\boldsymbol{\eta}_n|\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}\right]^\top \tag{52}$$

where $\mathbf{v}_n = \mathbf{u}^1_{i_{n_1}} \otimes \ldots \otimes \mathbf{u}^K_{i_{n_K}}$ and $\boldsymbol{\Sigma}_n = \left(\left(\mathbf{S}^{\setminus n}\right)^{-1} + \tau\mathbf{v}_n\mathbf{v}_n^\top\right)^{-1}$.

Next, we need to take the outer-level expectation to obtain the moments, namely, computing the mean of the conditional moment under the marginal tilted distribution $\tilde{p}(\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n})$. However, since $\tilde{p}(\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n})$ is analytically intractable, the outer expectation does not have a closed form. To tackle this issue, we observe that the moment matching is also performed between $q(\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n})$ and $\tilde{p}(\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n})$, and hence we can assume they are close, especially in high density regions. We then use the current posterior as the surrogate to compute the expected conditional moment,

$$\mathbb{E}_{\tilde{p}}[\boldsymbol{\phi}(\boldsymbol{\eta}_n)] \approx \mathbb{E}_{q(\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n})}\left[\boldsymbol{\rho}_n\right] \tag{53}$$

where $\boldsymbol{\rho}_n$ is the conditional moment.

Nonetheless, since $\boldsymbol{\rho}_n$ is a nonlinear function of the conditioned variables $\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}$ (see (51)), we do not have a close form to compute (53) either. But we have already known the form of $q(\Theta_{\setminus\boldsymbol{\eta}_n})$, so we can use the multivariate delta method (Oehlert, 1992; Bickel and Doksum, 2015) to compute the expectation easily. Specifically, we use a first-order Taylor approximation to represent the conditional moments,

$$\boldsymbol{\rho}_n(\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}) \approx \boldsymbol{\rho}_n\left(\mathbb{E}_q\left[\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}\right]\right) + \mathbf{J}\cdot\left(\text{vec}\left(\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}\right) - \text{vec}\left(\mathbb{E}_q\left[\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}\right]\right)\right) \tag{54}$$

Algorithm 3 BCTT

Input: $\mathcal{D} = \{(\mathbf{i}_1, t_1, y_1), \ldots, (\mathbf{i}_N, t_N, y_N)\}$, kernel hyper-parameters l, σ^2
Initialize approximation terms in (44) for each likelihood.
repeat
  Go through all modes, sequentially compute $\{q(\mathbf{Z}^k(i_k^s))\}_{k=1:K}$ by KF and smoother.
  for $n = 1$ **to** N in parallel **do**
    Simultaneously update $\mathcal{N}(\mathbf{H}\overline{\boldsymbol{\gamma}}_n|\boldsymbol{\beta}_n, \mathbf{S}_n)$, $\text{Gam}(\tau|b_n, c_n)$ and $\left\{\mathcal{N}\left(\mathbf{u}^k_{i_{n_k}}|\mathbf{m}^{k,n}_{i_{n_k}}, \mathbf{V}^{k,n}_{i_{n_k}}\right)\right\}_k$ in (44) with conditional moment matching and multi-variate delta method.
  end for
until Convergence
Return: $\{q(\mathcal{W}(t_n))\}_{n=1}^N$, $\{q(\mathbf{u}_j^k)\}_{1\le k\le K, 1\le j\le d_k}$, $q(\tau)$

where $\mathbf{J}$ is the Jacobian at $\mathbb{E}_q\left[\boldsymbol{\Theta}_{\setminus \boldsymbol{\eta}_n}\right]$. Then taking the expectation over the Taylor approximation gives

$$\mathbb{E}_{q(\boldsymbol{\Theta}_{\setminus \boldsymbol{\eta}_n})}\left[\boldsymbol{\rho}_n\right] \approx \boldsymbol{\rho}_n\left(\mathbb{E}_q\left[\boldsymbol{\Theta}_{\setminus \boldsymbol{\eta}_n}\right]\right). \tag{55}$$

We refer to (Oehlert, 1992; Wolter, 2007) for the theoretical justifications and guarantees of the delta method.

With the same approach, we can compute the moments for other random variables in $\boldsymbol{\Theta}$, including $\{\mathbf{u}^k_{i_{n_k}}\}$ and τ, and obtain their posterior in (48). Finally, we apply (49) to update the approximation terms in the likelihood.

While the derivation of the conditional moment matching is a bit lengthy, the implementation is straightforward. From (55) and (53), we just need to derive the form of the conditional moments (in our case, it is either Gaussian or Gamma), and then plug in the expectation of the conditioned variables under the current poster. For efficiency, we update the approximation factors of all the likelihoods in parallel, and then perform damping to be stable (Minka, 2001b). We repeatedly do message passing and conditional moment matching until convergence. The model inference is summarized in Algorithm 3.

In each iteration, our algorithm runs KF and RTS smoothing to go through data twice, so as to calculate the posterior of each state $\overline{\boldsymbol{\gamma}}_n$, and then conduct conditional moment matching in parallel to update the likelihood approximation for each data point. The overall time complexity is $\mathcal{O}(N\overline{R})$, where $\overline{R}$ is the size of the tensor-core. The space complexity is $\mathcal{O}\left(N(\overline{R}^2 + \sum_{k=1}^K R_k^2)\right)$ which is to store the posterior of each state and the likelihood approximation terms at each data point. Hence, our algorithm enjoys a linear scalability with the growth of data. Note that our algorithm fulfills the full GP inference, without the need for any sparse or low-rank approximations.

3.3.4 Experiments

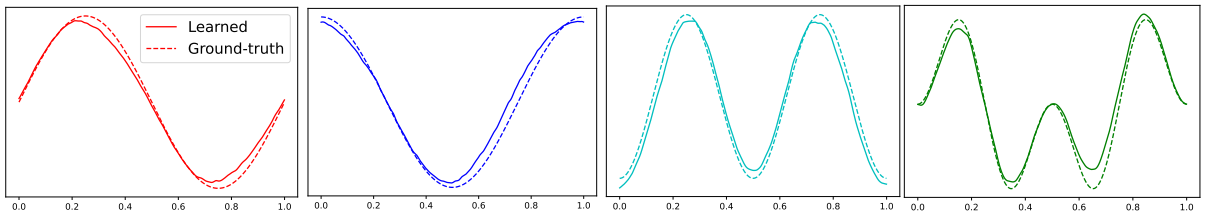
Ablation Study We first evaluated BCTT on a synthetic task. We simulated a two-mode tensor, where each mode includes 50 nodes. For each node, we generated two latent factors that reflect a clustering structure in each mode. Specifically, for the nodes in mode 1, we sampled the latent factors $\mathbf{u}_j^1$ from $\mathcal{N}([-1; 1], 0.1\mathbf{I})$ for $1 \le j \le 25$, and from $\mathcal{N}([1; -1], 0.1\mathbf{I})$ for $26 < j \le 50$. Similarly, for the nodes in mode 2, we sampled $\mathbf{u}_j^2 \sim \mathcal{N}([1; 1], 0.1\mathbf{I})$ when $1 \le j \le 25$, and $\mathcal{N}([-1; -1], 0.1\mathbf{I})$ for $26 < j \le 50$. Given the latent factors, we generate the tensor entry values at any time t from

$$y_{\mathbf{i}}(t) = u^1_{i_1,1}u^2_{i_2,1}w_{(1,1)}(t) + u^1_{i_1,1}u^2_{i_2,2}w_{(1,2)}(t) + u^1_{i_1,2}u^2_{i_2,1}w_{(2,1)}(t) + u^1_{i_1,2}u^2_{i_2,2}w_{(2,2)}(t), \tag{56}$$

where $w_{(1,1)}(t) = \sin(2\pi t)$, $w_{(1,2)}(t) = \cos(2\pi t)$, $w_{(2,1)}(t) = \sin(2\pi t)\sin(2\pi t)$, and $w_{(2,2)}(t) = \cos(2\pi t)\sin^2(2\pi t)$. These weight functions represent the four temporal interaction patterns between factors across the two modes, corresponding to the tensor-core $\mathcal{W}(t)$ in our model. We generated 2K observed entries from $t \in [0, 1]$.

We implemented our method BCTT with PyTorch (Paszke et al., 2019). We use the Matérn kernel with $\nu = 3/2$, and set $l = \sigma^2 = 0.1$. We ran our message-passing inference until convergence. The tolerance level was set to 10^{-3}. Then we compared the learned tensor-core $\mathcal{W}(t)$ with the ground-truth interaction functions between every pair of the factors across the two modes[1]. As we can see from Fig. 8, our approach recovered each function pretty accurately, showing that BCTT has successful captured all the temporal dynamics within the factor interactions.

[1]We normalized each learned interaction function by the maximum posterior mean of the corresponding state. This is to address the identifiablility issue, since scaling $\mathcal{W}$ arbitrarily then re-scaling $\mathcal{U}$ accordingly do not change the Tucker decomposition loss (or likelihood).



(a) $w_{(1,1)}(t)$ (b) $w_{(1,2)}(t)$ (c) $w_{(2,1)}(t)$ (d) $w_{(2,2)}(t)$

Figure 8: Recovered temporal dynamics within factor interactions by BCTT.

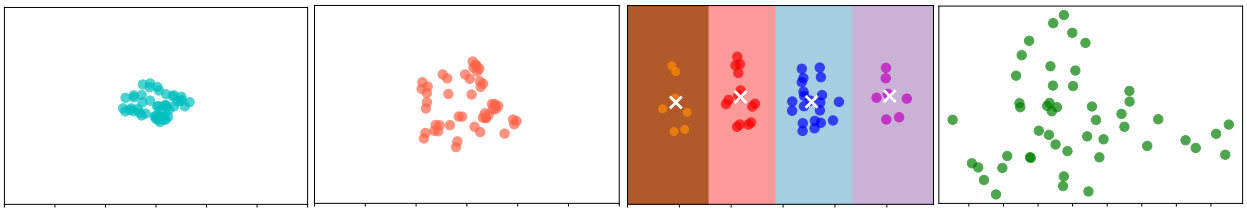
RMSE	*MovieLens*	*AdsClicks*	*DBLP*	RMSE	*MovieLens*	*AdsClicks*	*DBLP*
CT-CP	1.113 ± 0.004	1.337 ± 0.013	0.240 ± 0.007	CT-CP	1.165 ± 0.008	1.324 ± 0.013	0.263 ± 0.006
CT-GP	0.949 ± 0.008	1.422 ± 0.008	0.227 ± 0.009	CT-GP	0.965 ± 0.019	1.410 ± 0.015	0.227 ± 0.007
DT-GP	0.963 ± 0.008	1.436 ± 0.015	0.227 ± 0.007	DT-GP	0.949 ± 0.007	1.425 ± 0.015	0.225 ± 0.008
DDT-GP	0.957 ± 0.008	1.437 ± 0.010	0.225 ± 0.006	DDT-GP	0.948 ± 0.005	1.421 ± 0.012	0.220 ± 0.006
DDT-CP	1.022 ± 0.003	1.420 ± 0.020	0.245 ± 0.004	DDT-CP	1.141 ± 0.007	1.623 ± 0.013	0.282 ± 0.011
DDT-TD	1.059 ± 0.006	1.401 ± 0.022	0.232 ± 0.09	DDT-TD	0.944 ± 0.003	1.453 ± 0.035	0.312 ± 0.072
BCTT	$\mathbf{0.922 \pm 0.002}$	$\mathbf{1.322 \pm 0.012}$	$\mathbf{0.214 \pm 0.009}$	BCTT	$\mathbf{0.895 \pm 0.007}$	$\mathbf{1.304 \pm 0.018}$	$\mathbf{0.202 \pm 0.009}$
MAE				MAE			
CT-CP	0.788 ± 0.004	0.787 ± 0.006	0.105 ± 0.001	CT-CP	0.835 ± 0.006	0.792 ± 0.007	0.128 ± 0.001
CT-GP	0.714 ± 0.004	0.891 ± 0.011	0.092 ± 0.004	CT-GP	0.717 ± 0.012	0.883 ± 0.016	0.092 ± 0.002
DT-GP	0.722 ± 0.008	0.893 ± 0.008	0.084 ± 0.003	DT-GP	0.714 ± 0.005	0.886 ± 0.012	0.084 ± 0.001
DDT-GP	0.720 ± 0.003	0.894 ± 0.009	0.083 ± 0.001	DDT-GP	0.707 ± 0.004	0.882 ± 0.015	0.082 ± 0.003
DDT-CP	0.755 ± 0.002	0.901 ± 0.011	0.114 ± 0.002	DDT-CP	0.843 ± 0.003	1.082 ± 0.013	0.141 ± 0.004
DDT-TD	0.742 ± 0.006	0.866 ± 0.012	0.101 ± 0.001	DDT-TD	0.712 ± 0.002	0.903 ± 0.024	0.221 ± 0.047
BCTT	$\mathbf{0.698 \pm 0.002}$	$\mathbf{0.777 \pm 0.016}$	$\mathbf{0.084 \pm 0.001}$	BCTT	$\mathbf{0.679 \pm 0.001}$	$\mathbf{0.785 \pm 0.010}$	$\mathbf{0.080 \pm 0.001}$

(a) $R = 3$ (b) $R = 7$

Table 1: Prediction error and standard deviation for $R = 3$and $R = 7$ of BCTT. The results were averaged over five runs.

Real-World Applications Next, we examined BCTT on three real-world benchmark datasets. (1) *MovieLen100K* (`https://grouplens.org/datasets/movielens/`), a two-mode (user, movie) tensor, of size 610×9729. The entry values are movie ratings at different time points. We have $100,208$ observed entries and their timestamps. (2) *AdsClick* (`https://www.kaggle.com/c/avazu-ctr-prediction`), a three-mode mobile ads click tensor, *(banner-position, site domain, app)*, of size $7 \times 2842 \times 4127$. We collected 50K observed entry values (number of clicks) at different time points (in ten days). *DBLP* (`https://dblp.uni-trier.de/xml/`), a three-mode tensor about bibliographic records in computer science from 2011 to 2021, (author, conference, keyword), of size $3731 \times 1935 \times 169$. The entry values are the numbers of publications. There are 50k entry values and their timestamps.

We compared with following state-of-the-art multilinear and nonparametric tensor decomposition algorithms with time information integrated. (1) CT-CP (Zhang et al., 2021), continuous-time CP decomposition, which uses polynomial splines to estimates $\boldsymbol{\lambda}$ in (1) as a trend function. (2) CT-GP, continuous-time GP decomposition, which extends (Zhe et al., 2016a) to use GPs to learn tensor element as a function of the latent factors and time $y_{\mathbf{i}}(t) = g(\mathbf{u}^1_{i_1}, \ldots, \mathbf{u}^K_{i_K}, t) \sim$



(a) $\mathcal{W}(1)$ (b) $\mathcal{W}(4)$ (c) $\mathcal{W}(7)$ (d) DDT-TD

Figure 9: The structures of learned tensor-core at different time points by BCTT (a-c) and the static tensor-score learned by dynamic discrete-time Tucker decomposition (DDT-TD).

$\mathcal{GP}(0, \kappa(\cdot,\cdot))$. (3) DT-GP, discrete-time GP decomposition, which expands the tensor with a discrete time mode and then applies GP decomposition. (4) DDT-CP, dynamic discrete-time CP decomposition, which on top of DT-CP, places an RNN-like dynamic prior over the time factors, $p(\mathbf{t}_j|\mathbf{t}_{j-1}) = \mathcal{N}(\mathbf{t}_j|\sigma(\mathbf{A}\mathbf{t}_{j-1}) + \mathbf{b}, v\mathbf{I})$ where $\sigma(\cdot)$ is a nonlinear activation, (5) DDT-TD and (6) DDT-GP, dynamic discrete-time Tucker and GP decomposition, which place the same dynamic prior as in DDT-CP.

As shown in Table 1, our approach BCTT outperforms the competing methods in all the cases except that in Table 1d, on *AdsClicks*, BCTT was the second best, and its MAE is slightly worse than CP-CT. In most cases, the improvement obtained by BCTT is large and significant ($p < 0.05$). It shows that our semi-parametric model BCTT not only maintains the interpretable structure as in Tucker decomposition, but also achieves a superior performance, even to full nonparametric models, *e.g.,* CT-GP and DDT-GP. This might because BCTT uses the state-space representation to enable full GP inference, without any low-rank/sparse approximation as needed in those GP baselines.

Furthermore, we investigated if our learned tensor-core $\mathcal{W}(t)$ can reflect temporal structural variations. To do so, we set $R = 7$ and ran BCTT on *DBLP* dataset. We looked at the tensor-core at three time points $t = 1, 4, 7$. The size of the tensor-core is $7 \times 7 \times 7$. We followed (Fang et al., 2021a) to fold the tensor-core to a 49×7 interaction matrix for each mode. Thus, each row expresses how strongly the combination of factors in other modes interact with the factors in the current mode. To reflect the structure, we ran Principled Component Analysis (PCA), and show the first and second principled components in a plane. We also tested DDT-TD which learns a static tensor-core but using time factors and nonlinear dynamics. We looked at the results at mode 1. As shown in Fig. 9 a-c, we can see a clear structural variation. At $t = 1$, the tensor-core elements are quite concentrated, showing somewhat homogeneous interactions. The case is similar at $t = 4$ but the interaction strengths are more scattered. However, at $t = 7$, the strengths clearly formed four groups, exhibiting heterogeneous interaction patterns — a major shift. Together these imply the interaction between factors evolve with time. As a comparison, the tensor-core learned by DDT-TD do not reflect apparent structures or temporal patterns. It is inconvenient to examine how the interactions between the factors of the tensor nodes evolve.

4 Proposed Research

As our research has thus far addressed the challenges of streaming tensor data (BASS (Fang et al., 2021a), SBDT (Fang et al., 2021b)) and temporal tensor data (BCTT (Fang et al., 2022)) independently, the natural next step is to develop a unified framework that combines both concepts of "dynamic" modeling: streaming and temporal. To this end, we propose the concept of Streaming Factor Trajectory Learning (SFTL) for temporal tensor decomposition, which aims to model temporal Bayesian tensors in a streaming manner. By incorporating the benefits of both streaming and temporal modeling into a unified framework, SFTL has the potential to offer superior modeling performance, increased interpretability, and broader applicability to real-world problems.

4.1 Streaming Factor Trajectory Learning for Temporal Tensor Decomposition(SFTL)

Practical tensor data is often along with time information. Most existing temporal decomposition approaches estimate a set of fixed factors for the objects in each tensor mode, and hence cannot capture the temporal evolution of the objects' representation. More important, we lack an effective approach to capture such evolution from streaming data, which is very common in real-world applications. To address these issues, we propose Streaming Factor Trajectory Learning (SFTL) for temporal tensor decomposition. We use Gaussian processes (GPs) to model the trajectory of factors so as to flexibly estimate their temporal evolution. To address the computational challenges in handling streaming data, we convert the GPs into a state-space prior by constructing an equivalent stochastic differential equation (SDE). We develop an efficient online filtering algorithm to estimate a decoupled running posterior of the involved factor states upon receiving new data. The decoupled estimation enables us to conduct standard RTS smoothing to compute the full posterior of all the trajectories in parallel, without the need for revisiting any previous data.

4.1.1 Model

In real-world applications, tensor data is often associated with time information, namely, the timestamps at which the objects of different modes interact to generate the entry values.

To capture the potential evolution of the objects' inner properties, we propose a Bayesian temporal tensor decomposition model that can estimate a trajectory of the factor representation. Specifically, for each object j in mode m, we model

the factors as a function of time, $\mathbf{u}_j^m : [0, \infty] \to \mathbb{R}^R$. To flexibly capture a variety of temporal evolution, we assign a GP prior over each element of $\mathbf{u}_j^m(t) = [u_{j1}^m(t), \ldots, u_{jR}^m(t)]^\top$,

$$u_{jr}^m(t) \sim \mathcal{GP}(0, \kappa(t, t')) \tag{57}$$

where $1 \leq r \leq R$. Given the factor trajectories, we then use the CP or Tucker form to sample the entry values at different time points. For the CP form, we have

$$p(y_{\boldsymbol{\ell}}(t)|\mathcal{U}(t)) = \mathcal{N}(y_{\boldsymbol{\ell}}(t)|\mathbf{1}^\top(\mathbf{u}_{\ell_1}^1(t) \circ \ldots \circ \mathbf{u}_{\ell_M}^K(t)), \tau^{-1}), \tag{58}$$

where $\mathcal{U}(t) = \{\mathbf{u}_j^m(t)\}$ includes all the factor trajectories, and τ is the inverse noise variance, for which we assign a Gamma prior, $p(\tau) = \text{Gam}(\tau|\alpha_0, \alpha_1)$. Note that for convenience, we absorb the CP coefficients $\boldsymbol{\lambda}$ (see (1)) into the factors. For the Tucker form, we have $p(y_{\boldsymbol{\ell}}(t)|\mathcal{U}(t), \mathcal{W}) = \mathcal{N}(y_{\boldsymbol{\ell}}(t)|\text{vec}(\mathcal{W})^\top(\mathbf{u}_{\ell_1}^1(t) \otimes \ldots \otimes \mathbf{u}_{\ell_M}^K(t)), \tau^{-1})$ where we place a standard normal prior over the tensor-core, $p(\text{vec}(\mathcal{W})) = \mathcal{N}(\text{vec}(\mathcal{W})|\mathbf{0}, \mathbf{I})$. In this work, we focus on continuous observations. It is straightforward to extend our method for other types of observations.

Suppose we have a collection of observed entry values and timestamps, $\mathcal{D} = \{(\boldsymbol{\ell}_1, y_1, t_1), \ldots, (\boldsymbol{\ell}_N, y_N, t_N)\}$ where $t_1 \leq \cdots \leq t_N$. We denote the sequence of timestamps when a particular object j of mode m participated in the observed entries by $s_{j,1}^m < \ldots < s_{j,c_j^m}^m$, where c_j^m is the participation count of the object. Note that it is a sub-sequence of $\{t_n\}$. From the GP prior (57), the values of each $u_{jr}^m(t)$ at these timestamps follow a multi-variate Gaussian distribution, $p(\mathbf{u}_{jr}^m) = \mathcal{N}(\mathbf{u}_{jr}^m|\mathbf{0}, \mathbf{K}_j^m)$ where $\mathbf{u}_{jr}^m = [u_{jr}^m(s_{j,1}^m), \ldots, u_{jr}^m(s_{j,c_j^m}^m)]^\top$ and $\mathbf{K}_j^m$ is the kernel matrix computed at these timestamps. The joint probability of our model with the CP form is

$$p(\{\mathbf{u}_{jr}^m\}, \tau, \mathbf{y}) = \prod_{m=1}^{M} \prod_{j=1}^{d_m} \prod_{r=1}^{R} \mathcal{N}(\mathbf{u}_{jr}^m|\mathbf{0}, \mathbf{K}_j^m)\text{Gam}(\tau|\alpha_0, \alpha_1) \cdot \prod_{n=1}^{N} \mathcal{N}(y_n|\mathbf{1}^\top(\mathbf{u}_{\ell_{n1}}^1(t_n) \circ \ldots \circ \mathbf{u}_{\ell_{nM}}^M(t_n)), \tau^{-1}). \tag{59}$$

The joint probability with the Tucker form is the same except that we use the Tucker likelihood instead and multiply with the prior of tensor-core $p(\mathcal{W})$.

While this formulation is straightforward, it can introduce computational challenges. There are many multi-variate Gaussian distributions in the joint distribution (59), *i.e.,* $\{\mathcal{N}(\mathbf{u}_{jr}^m|\mathbf{0}, \mathbf{K}_j^m)\}$. The time and space complexity to compute each $\mathcal{N}(\mathbf{u}_{jr}^m|\mathbf{0}, \mathbf{K}_j^m)$ is $\mathcal{O}\left(\left(c_j^m\right)^3\right)$ and $\mathcal{O}\left(\left(c_j^m\right)^2\right)$, respectively. With the increase of N, the appearance count c_j^m for many objects can grow as well, making the computation cost very expensive or even infeasible. The issue is particularly severe when we handle streaming data — the number of timestamps grows rapidly with new data keeps coming in, so does the size of each covariance matrix.

4.1.2 Running Posterior Inference

The components of SFTL inference applied similar techniques as temporal tensor(BCTT (Fang et al., 2022)), saying, efficient sequential inference of factor trajectories by Kalman filter and smoothing. As for the streaming update of other parameters, *e.g.,* noise level τ, Tucker core $\mathcal{W}$, we apply the same online moment-matching framework as in BASS (Fang et al., 2021a). We illustrate the inference procedure in Figure 10, but omit the details here for brevity.

4.1.3 Expected Outcome

By exploring the temporal trajectory of latent factors with streaming inference, the proposed method can capture the dynamic interactions between factors more accurately and lead to more accurate and reliable models, with increased interpretability and usefulness for a wide range of applications.

References

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

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

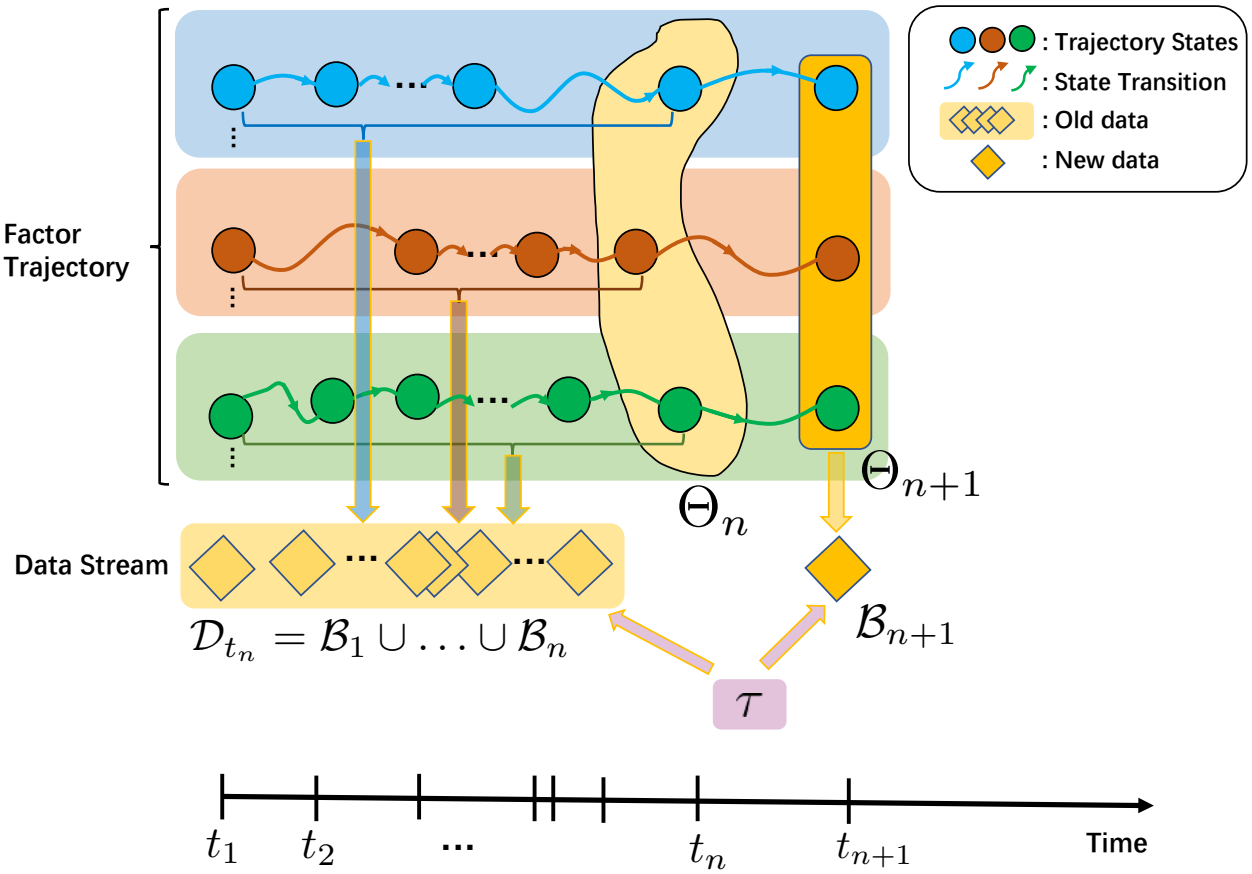


Figure 10: A graphical model representation of SFTL, from which we can see $\{\Theta_{n+1}, \mathcal{B}_{n+1}\}$ are independent to $\mathcal{D}_{t_n}$ conditioned on Θ_n and the noise inverse variance τ, namely, $\Theta_{n+1}, \mathcal{B}_{n+1} \perp \mathcal{D}_{t_n} | \Theta_n, \tau$.

Bader, B. W., Kolda, T. G., et al. (2015). Matlab tensor toolbox version 2.6. Available online.

Bickel, P. J. and Doksum, K. A. (2015). Mathematical statistics: basic ideas and selected topics, volume I, volume 117. CRC Press.

Bishop, C. M. (2006). Pattern recognition and machine learning. springer.

Boyen, X. and Koller, D. (2013). Tractable inference for complex stochastic processes. arXiv preprint arXiv:1301.7362.

Broderick, T., Boyd, N., Wibisono, A., Wilson, A. C., and Jordan, M. I. (2013). Streaming variational bayes. Advances in neural information processing systems, 26.

Bui, T. D., Nguyen, C., and Turner, R. E. (2017). Streaming sparse gaussian process approximations. Advances in Neural Information Processing Systems, 30.

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

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

Du, Y., Zheng, Y., Lee, K.-c., and Zhe, S. (2018). Probabilistic streaming tensor decomposition. In 2018 IEEE International Conference on Data Mining (ICDM), pages 99–108. IEEE.

Fang, S., Kirby, R. M., and Zhe, S. (2021a). Bayesian streaming sparse tucker decomposition. In Uncertainty in Artificial Intelligence, pages 558–567. PMLR.

Fang, S., Narayan, A., Kirby, R., and Zhe, S. (2022). Bayesian continuous-time tucker decomposition. In International Conference on Machine Learning, pages 6235–6245. PMLR.

Fang, S., Wang, Z., Pan, Z., Liu, J., and Zhe, S. (2021b). Streaming bayesian deep tensor factorization. In International Conference on Machine Learning, pages 3133–3142. PMLR.

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

Hartikainen, J. and Särkkä, S. (2010). Kalman filtering and smoothing solutions to temporal gaussian process regression models. In 2010 IEEE international workshop on machine learning for signal processing, pages 379–384. IEEE.

Ishwaran, H. and Rao, J. S. (2005). Spike and slab variable selection: frequentist and bayesian strategies.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems.

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

Kolda, T. G. (2006). Multilinear operators for higher-order decompositions, volume 2. United States. Department of Energy.

Lancaster, P. and Rodman, L. (1995). Algebraic riccati equations. Clarendon press.

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

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

Minka, T. P. (2001b). A family of algorithms for approximate Bayesian inference. PhD thesis, Massachusetts Institute of Technology.

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

Oh, S., Park, N., Lee, S., and Kang, U. (2018). Scalable Tucker factorization for sparse tensors-algorithms and discoveries. In 2018 IEEE 34th International Conference on Data Engineering (ICDE), pages 1120–1131. IEEE.

Oksendal, B. (2013). Stochastic differential equations: an introduction with applications. Springer Science & Business Media.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32:8026–8037.

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate gaussian process regression. The Journal of Machine Learning Research, 6:1939–1959.

Rasmussen, C. E. and Williams, C. K. I. (2006). Gaussian Processes for Machine Learning. MIT Press.

Rauch, H. E., Tung, F., and Striebel, C. T. (1965). Maximum likelihood estimates of linear dynamic systems. AIAA journal, 3(8):1445–1450.

Särkkä, S. (2013). Bayesian filtering and smoothing. Number 3. Cambridge University Press.

Särkkä, S. et al. (2006). Recursive Bayesian inference on stochastic differential equations. Helsinki University of Technology.

Titsias, M. and Lázaro-Gredilla, M. (2011). Spike and slab variational inference for multi-task and multiple kernel learning. Advances in neural information processing systems, 24.

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

Wainwright, M. J. and Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. Now Publishers Inc.

Wang, Z. and Zhe, S. (2019). Conditional expectation propagation. In UAI, page 6.

Wolter, K. (2007). Introduction to variance estimation. Springer Science & Business Media.

Zhang, Y., Bi, X., Tang, N., and Qu, A. (2021). Dynamic tensor recommender systems. Journal of Machine Learning Research, 22(65):1–35.

Zhe, S., Qi, Y., Park, Y., Xu, Z., Molloy, I., and Chari, S. (2016a). Dintucker: Scaling up gaussian process models on large multidimensional arrays. In Thirtieth AAAI conference on artificial intelligence.

Zhe, S., Zhang, K., Wang, P., Lee, K.-c., Xu, Z., Qi, Y., and Ghahramani, Z. (2016b). Distributed flexible nonlinear tensor factorization. In Advances in Neural Information Processing Systems, pages 928–936.