BAYESIAN TENSOR LEARNING FOR DYNAMIC HIGH-ORDER DATA

by

Shikai Fang

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

School of Computing

The University of Utah

August 2024

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Shikai Fang**
has been approved by the following supervisory committee members:

Shandian Zhe ,	Chair(s)	**6/13/2024** Date Approved
Robert Michael Kirby II ,	Member	**5/31/2024** Date Approved
Akil Narayan ,	Member	**5/31/2024** Date Approved
Tucker Hermans ,	Member	**5/31/2024** Date Approved
Bei Wang Phillips ,	Member	**5/31/2024** Date Approved

by **Mary W. Hall** , Chair/Dean of
the Department/College/School of **Computing**
and by **Darryl P. Butt** , Dean of The Graduate School.

ABSTRACT

Tensor decomposition is an essential technique in high-order data analysis and prediction, serving as a fundamental tool for uncovering the multifaceted structures inherent in tensor data. Traditional methods like CANDECOMP/PARAFAC (CP) and Tucker decomposition have pioneered this area. However, these methods struggle with the sparsity and noise common in real-world data, and they lack mechanisms to handle the dynamic nature of pratical applications, including streaming data, temporal variations, and continuously indexed tensor, i.e., functional data.

This dissertation presents a comprehensive suite of advancements in Bayesian tensor learning that address these challenges across various forms of dynamic tensor data: streaming tensors, temporal tensors, and functional tensors. For streaming tensor data, the dissertation introduces the Bayesian Streaming Sparse Tucker Decomposition (BASS) and the Streaming Bayesian Deep Tensor Factorization (SBDT), providing efficient and scalable solutions for streaming tensor analysis with sparsification mechanisms for avoiding overfitting and promoting pattern discovery. As for temporal tensors, the dissertation presents Bayesian Continuous-Time Tucker Decomposition (BCTT), and Streaming Factor Trajectory Learning for Temporal Tensor Decomposition (SFTL), which can dynamically capture the evolving temporal factors. Further extending the scope, the dissertation proposes Functional Bayesian Tucker Decomposition for Continuous-Indexed Tensor (FunBAT), that adapts tensor decomposition to infinite-dimensions, i.e., functional data.

Throughout this dissertation, each contribution is explored to underpin a robust and scalable Bayesian framework, demonstrating significant real-world implications for handling the complexities of data across various applications.

To my parents.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this era where artificial intelligence is rapidly reshaping the world, machine learning faces an ever-expanding array of challenges and opportunities derived from real-world data. One significant challenge is the shift from traditional data types like text, images, and sequences to more structured, interaction-based data. Such data demands more sophisticated modeling and computational strategies. Tensors, with their inherent ability to represent high-order interactions and multi-way relationships, are crucial in this landscape.

Tensor data is a type of multidimensional data that arises in a multitude of real-world applications, providing a natural representation of information that spans multiple modes, such as time, space, and class. For instance, in recommendation systems, a three-mode tensor comprising *user, item,* and *website* is often utilized, where the entry values signify the three-way interactions. In the realm of climate modeling, four-mode tensors encapsulating *longitude, latitude, altitude,* and *time* are employed to represent spatiotemporal data, such as satellite measurements of temperature variations. In neuroscience, tensors model brain activating across various modalities, reflecting the complexity and dynamic nature of neural interactions.

The analysis of tensor data poses unique challenges due to its high dimensionality and complex structures. Tensor decomposition stands out as a promising approach by breaking down a tensor into a set of low-rank representations, known as factors. This decomposition facilitates efficient processing, analysis, and interpretation of tensor data, leading to significant advancements in machine learning, data mining, and signal processing.

Classical tensor decomposition methods, such as the CANDECOMP/PARAFAC (CP) [32] and Tucker decomposition [91], are foundational in this analysis. Rooted in the concept of learning latent factors for real-world objects from their observed interactions, these methods have become cornerstones in machine learning and data science. They are

extensively applied in a variety of domains including recommendation systems, signal processing, data compression, and computer vision.

Despite the broad application of classical tensor decomposition methods, they frequently encounter challenges in real-world scenarios, especially when dealing with dynamic tensor data. These methods often struggle with the inherent sparsity and noise within tensor data. Moreover, they lack robust mechanisms to handle the dynamic nature of real-world data, which includes streaming updates, temporal variations, and function tensors with continuous indices. For example, in streaming data scenarios, the data arrives sequentially, and the model must adapt to these changes in real time. In temporal data, the data evolves over time, and the model must capture these temporal patterns continuously. In functional data, the data is indexed continuously, and the model must adapt to these continuous indices, which correspond to infinite dimension in each mode. Classical tensor decomposition methods are not well-suited to handle these features, limiting their effectiveness in real-world applications. Such limitations significantly hinder their effectiveness in scenarios where data evolves or updates continuously and dynamically.

1.1 Dissertation Statement

In this dissertation, we aim to develop efficient and effective probabilistic tensor models within a Bayesian framework, specifically designed to address a series of challenges associated with real-world tensor data. Our goal is to push the boundaries and influence of tensor learning to a broader range of applications. To achieve this objective, we will now enumerate our thesis statements associated with each subtask.

- **Goal 1: Online Inference for Streaming Tensor Data**:

  We hypothesize that developing tensor models capable of performing online inference directly addresses the limitations of traditional batch processing methods by enabling real-time, incremental updates to tensor decompositions. This approach not only accommodates the dynamic and incremental nature of streaming data but also ensures data privacy and meets the computational efficiency demands of modern applications such as social media platforms.

  Traditional tensor decomposition methods are predominantly designed to operate on static datasets, requiring repeated access to the entire dataset for performing iterative updates or gradient computations. This batch processing approach is not

feasible in many practical scenarios where data is generated dynamically and incrementally. In such cases, tensor entries arrive in streams, leading to a dynamic growth of the tensor in each mode. This continuous data flow makes it impractical to either retrain models from scratch with every new data increment or delay training until sufficient data accumulates.

Furthermore, in environments where data privacy and security are paramount, such as applications in social media platforms like Snapchat or Instagram—data retention policies may prevent the storage of data after initial access. This restriction necessitates models that can update their embeddings or factors immediately upon receiving new data, without the ability to revisit previously seen entries. The challenge lies not only in achieving reliable and incremental updates to tensor factors but also in maintaining the expressive power of the model, accommodating the noise inherent in real-world data, and effectively handling the sparsity often observed in tensor data.

Consequently, there is a critical need for tensor models capable of efficient online learning and inference, as opposed to traditional offline learning methodologies. The task demands the development of innovative tensor decomposition techniques that can adapt in real-time to new information without the need for batch processing. This involves balancing model complexity and computational efficiency to ensure timely updates while preserving the quality of the tensor decomposition. Addressing this challenge will involve rethinking the foundational algorithms of tensor analysis to support the dynamic and incremental nature of modern data streams.

- **Goal 2: Modeling Continuous-Time Evolution in Temporal Tensor Data**:

  We hypothesize that by developing tensor models capable of continuous-time modeling, we can significantly enhance our ability to capture and utilize the rich temporal dynamics embedded in real-world tensor data. This approach will allow for more effective and flexible decompositions and completions, greatly improving the interpretability of evolving patterns in data.

  Real-world tensor data often include explicit time information, reflecting the moments when interactions occur across different modes. This time-stamped data holds potential insights into the evolving patterns that are crucial for understanding

dynamic processes. Despite the availability of numerous temporal decomposition techniques, most current methods either treat time as a static, separate mode or incorporate it superficially—often by discretizing it into crude intervals or outright discarding it during the analysis. Such approaches significantly limit the ability to capture and utilize the full spectrum of information embedded in the temporal dynamics of the data, leading to a loss of valuable insights and potential misinterpretations of the underlying processes.

Addressing this gap necessitates the development of tensor models capable of continuous-time modeling. These models should not only leverage and thoroughly exploit temporal information to perform more effective and flexible decompositions and completions but also enhance the interpretability of the data. By understanding how the interactions among real-world objects evolve over time, we can gain deeper insights from the applications, which are pivotal for making decisions in many applications, such as in financial markets, weather forecasting, and dynamic social networks.

- **Goal 3: Extending Tensor Learning Methods to Continuous-Indexed Tensor Data**:

  We hypothesize that by extending classical tensor decomposition methods to accommodate continuous-indexed tensor data, we can more authentically model the intricacies of natural data structures present in fields like climate and geographic modeling. This advancement will enable the capture and analysis of complex, infinite dimensional relationships with enhanced granularity and accuracy, preserving the integrity of the original data information.

  The standard tensor decomposition framework primarily caters to data structured in discrete, finite-dimensional grids. Each mode of the tensor is indexed by integers, representing a finite set of objects like "user 1, user 2," etc., with each tensor entry specified by a tuple of these discrete indices. This structural requirement significantly limits the application of tensor methods in fields where data naturally aligns with continuous coordinates, such as in climate and geographic modeling. In these domains, data points are often associated with continuous indexes like latitude, longitude, and time, making traditional tensor decomposition inapplicable without adjustments.

  Currently, a common workaround to apply tensor models to such data involves

discretizing these continuous modes—binning timestamps into defined intervals or segmenting latitude and longitude into discrete ranges. However, this approach inevitably results in the loss of fine-grained details and often imposes the challenge of choosing an optimal discretization strategy. The granularity lost during such discretization can substantially alter the insights gained from the data, potentially leading to less accurate or meaningful models.

To overcome these limitations, there is a compelling need to extend classical tensor decomposition methods to embrace the functional field. Specifically, this would involve decomposing continuous-indexed tensor data through interactions among groups of latent functions. By successfully modeling the continuous indices and latent dynamics of each mode, tensor learning can be significantly expanded to broader scenarios, particularly those involving data structures with continuous spatio-temporal coordinates, such as point clouds in 3D modeling or dynamic patterns in weather modeling. This extension would not only preserve the integrity of the original data but also enhance the applicability and effectiveness of tensor decomposition in capturing complex, multi-dimensional relationships in a wide array of real-world applications.

- **Goal 4: Efficient and Scalable Bayesian Inference for Large-Scale Tensor Data**:

  We hypothesize that by developing efficient and scalable Bayesian inference strategies tailored specifically to dynamic tensor learning models, we can significantly enhance the practical application of these models to large-scale data scenarios. This approach not only addresses the computational and resource challenges posed by Bayesian inference's complexity but also preserves the probabilistic framework's advantages such as robustness against noise and accurate uncertainty quantification.

  While the initial challenges and tasks focused primarily on developing new Bayesian tensor learning models to address various dynamic data structures, another overarching challenge remains in the domain of inference—ensuring efficiency and scalability. The probabilistic framework employed in our models offers significant advantages in handling noise, enhancing model robustness, and providing measures of uncertainty. However, a major challenge lies in the complexity of Bayesian inference, especially when compared to deterministic algorithms. Typically, Bayesian

methods do not yield closed-form posterior distributions and rely on sampling techniques, which may not scale well or prove feasible for large-scale tensor data due to computational and resource constraints.

Therefore, it is imperative to devise efficient and scalable inference strategies tailored to the dynamic tensor learning models developed in this dissertation. These strategies must not only accommodate the computational demand of large-scale data but also maintain the fidelity of the probabilistic framework. Addressing this challenge is crucial for ensuring that our models can be practically applied to real-world problems, where the ability to process vast amounts of data swiftly and accurately can significantly impact the outcomes of machine learning applications. Our approach aims to bridge the gap between advanced Bayesian modeling techniques and the practical necessities of applied data science, facilitating the deployment of these sophisticated models in real-world tasks.

1.2 Contributions

In this dissertation, we present a comprehensive suite of advancements in Bayesian tensor learning that address the aforementioned challenges across various forms of dynamic tensor data: streaming tensors, temporal tensors, and functional tensors. Our contributions span a range of innovative models and efficient algorithms designed to handle the complexities of dynamic data across various applications. Our major contributions are listed as follows:

- **Bayesian Streaming Sparse Tucker Decomposition (BASS-Tucker)** [23]

  We address the challenges of streaming tensor data by introducing a Bayesian Streaming Sparse Tucker decomposition method. We employ spike-and-slab priors over the core tensor elements to identify meaningful interactions and prevent overfitting. The novel usage of conditional moment matching and delta methods enables efficient one-shot incremental updates of latent factors and the core tensor with each new data batch, significantly improving efficiency over traditional methods that update iteratively.

- **Streaming Bayesian Deep Tensor Factorization (SBDT)** [25]

  The SBDT method is another innovative solution for streaming tensor data analy-

sis with deep neural network-based decomposition models. This approach leverages Bayesian neural networks to develop a deep tensor factorization model that adapts to streaming data. By assigning spike-and-slab priors to neural network weights, it promotes sparsity and prevents overfitting, while responsive incremental updates facilitate real-time learning. We develop an integration of multivariate delta methods and moment matching for the predictive distribution of the neural network output, which ensures our model dynamically adjusts to new data, maintaining robustness and superior completion accuracy.

- **Bayesian Continuous-Time Tucker Decomposition (BCTT)** [24]

  BCTT introduces a time-varying function model for the tensor core within the classical Tucker decomposition framework. By incorporating Gaussian process priors, our model adeptly captures complex temporal dynamics, while maintaining interpretability and flexibility. This approach utilizes a stochastic differential equation representation of temporal Gaussian processes to enable efficient and high-quality posterior inference.

- **Streaming Factor Trajectory Learning for Temporal Tensor Decomposition (SFTL)** [27]

  SFTL is a novel method that combines the two meanings of "dynamic"—streaming and temporal modeling—to provide a unified framework for temporal Bayesian tensor decomposition. This model uses Gaussian processes to estimate the trajectories of tensor decomposition factors over time, adapting them to a state-space representation via stochastic differential equations for computational efficiency. Our innovative online filtering algorithm updates a decoupled running posterior approximation of factor states as new data arrives, allowing for concurrent, efficient trajectory analysis through Rauch-Tung-Striebel smoothing. The learned trajectories provide valuable insights into the temporal evolution of tensor factors, enhancing the interpretability and utility of the model.

- **Functional Bayesian Tucker Decomposition for Continuous-Indexed Tensor (FunBaT)** [28]

  FunBaT extends tensor decomposition to continuous-indexed data by modeling interactions between the Tucker core and a set of latent functions represented as

Gaussian processes. Each process is converted into a state-space prior through an equivalent stochastic differential equation, enhancing the model's computational efficiency while preserving its ability to capture intricate multi-way relationships within infinite-dimensional space.

1.3 Organization

This dissertation is organized as follows:

In Chapter 2, we cover the necessary background of tensor decomposition, Bayesian inference, and Gaussian processes.

Chapter 3 introduces the Bayesian Streaming Sparse Tucker Decomposition (BASS-Tucker) method, which addresses the challenges of streaming tensor data by employing spike-and-slab priors over the core tensor elements to selectively enhance meaningful interactions and prevent overfitting.

In Chapter 4, we demonstrate the Streaming Bayesian Deep Tensor Factorization (SBDT) method, which leverages Bayesian neural networks to develop a deep tensor factorization model that adapts to streaming tensor data.

In Chapter 5, we propose the Bayesian Continuous-Time Tucker Decomposition (BCTT) method, which introduces a time-varying function model for the tensor core within the classical Tucker decomposition framework.

In Chapter 6, we design Streaming Factor Trajectory Learning for Temporal Tensor Decomposition (SFTL), a novel method that combines the two meanings of "dynamic"—streaming and temporal modeling—to provide a unified framework for temporal Bayesian tensor decomposition by estimating the trajectories of tensor decomposition factors over time.

In Chapter 7, we presents the Functional Bayesian Tucker Decomposition for Continuous-indexed Tensor (FunBaT) method, which extends tensor decomposition to continuous-indexed data by modeling interactions between the Tucker core and a set of latent functions represented as Gaussian processes.

In Chapter 8, we summarize the dissertation and discuss potential directions for future research.

CHAPTER 2

BACKGROUND

In this chapter, we provide background on Bayesian tensor decomposition, Bayesian online learning, and Gaussian processes. We first introduce the standard tensor decomposition methods, including CANDECOMP/PARAFAC (CP), Tucker decomposition and non-linear tensor decomposition. We then discuss the Bayesian framework for tensor decomposition. Next, we introduce Assumed Density Filtering (ADF), a Bayesian online learning framework that allows for efficient online posterior updates. Finally, we introduce Gaussian processes, a powerful non-parametric Bayesian model that can be used to estimate a wide range of functions.

2.1 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 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 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 $\mathbf{u}_j^k \in \mathbb{R}^{r_k}$ 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.

We can estimate the optimal factors $\mathcal{U}$ to reconstruct the tensor $\mathcal{Y}$ by minimizing a loss on the observed entries. This involves different tensor decomposition approaches that design various factorization forms on how these latent factors interact. Based on the assumption that each entry value $y_{\mathbf{i}} = y_{(i_1,\ldots,i_K)}$ represents the interaction of corresponding latent factors in each mode $\mathbf{u}_{i_1}^1, \ldots, \mathbf{u}_{i_K}^K$, the specific form of interaction function f can vary. We will provide a detailed introduction to some popular tensor decomposition approaches, including CP decomposition, Tucker decomposition, and other non-linear

decomposition.

1. **CANDECOMP/PARAFAC (CP) Decomposition** CP decomposition, also known as Canonical Decomposition or PARAFAC [32], is a popular tensor decomposition method that expresses a high-dimensional tensor $\mathcal{Y} \in \mathbb{R}^{d_1 \times \cdots \times d_K}$ as the sum of rank-1 tensors, each of which is the outer product of vectors from the corresponding factor matrices. Mathematically, it assumes $r_1 = r_2 \cdots = r_K$ and designs f as a rank-wise product of all latent factors, followed by a summation:

$$y_{\mathbf{i}} \approx f(\mathbf{u}_{i_1}^1, \ldots, \mathbf{u}_{i_K}^K) = \left(\mathbf{u}_{i_1}^1 \circ \ldots \circ \mathbf{u}_{i_K}^K\right)^\top \mathbf{1}, \tag{2.1}$$

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

One of the main advantages of CP decomposition is its interpretability and efficiency, as the factor matrices represent the latent factors for each mode of the tensor. This allows for easy interpretation and visualization of the tensor structure, making it a popular choice for a wide range of applications. However, one of the main disadvantages of CP decomposition is its overly simplified expression, which accounts for the interaction between every r-th factor in different modes, but overlooks all other possible interactions. This can lead to a lack of flexibility in handling complex data and application.

2. **Tucker Decomposition**

Tucker decomposition [91] takes one more step towards a more expressive interaction by considering all the possible interactions between the factors across the tensor modes. It parameterizes 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:

$$y_{\mathbf{i}} \approx f(\mathbf{u}_{i_1}^1, \ldots, \mathbf{u}_{i_K}^K) = \mathrm{vec}(\mathcal{W})^\top \left(\mathbf{u}_{i_1}^1 \otimes \ldots \otimes \mathbf{u}_{i_K}^K\right), \tag{2.2}$$

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

Tucker decomposition is a more general form than CP decomposition, as it can capture more complex interactions between the latent factors. The core tensor $\mathcal{W}$ in Tucker decomposition provides a more flexible representation of the tensor structure, allowing for a more accurate approximation and interpretability of the tensor data. However, Tucker decomposition is also more computationally expensive than CP decomposition, as it requires the estimation of a core tensor $\mathcal{W}$ in addition to the factor matrices. This can make Tucker decomposition more challenging in large-scale datasets and/or high-mode tensors.

3. **Nonlinear Decomposition**

   As classical tensor decomposition methods (e.g., CP and Tucker) assume multi-linear interactions between the factors, they may be incapable of estimating more complex, nonlinear relationships. Recent work has proposed non-linear tensor decomposition methods that take parameterized model functions to capture the non-linear interactions between the latent factors,

$$y_{\mathbf{i}} \approx f_\theta(\mathbf{u}^1_{i_1}, \ldots, \mathbf{u}^K_{i_K}), \tag{2.3}$$

   where f_θ is a non-linear function parameterized by θ. The non-linear tensor decomposition methods can be used to capture more complex relationships between the latent factors, allowing for a more accurate representation of the underlying tensor structure. For example, *GPTF* [107] adopts the Gaussian Process with a stationary kernel, *POND* [86] extend it with the neural kernel. *CoSTco* [51] applies deep neural networks to model the interaction between factors. Our research *SBDT* [25] adopts the Bayesian neural network to model the decomposition.

In real-world tensor data, the observations of tensor always come with noise and uncertainty. The Bayesian framework enables a principled way to incorporate the uncertainty into the tensor decomposition. In the Bayesian framework, the observed entry values of tensor $\mathcal{Y}$ is assumed to be generated from a decomposition model f with some noise, e.g., $y_{\mathbf{i}} \approx \mathcal{N}(y_{\mathbf{i}} \mid f(\mathbf{u}^1_{i_1}, \ldots, \mathbf{u}^K_{i_K}), \tau^{-1})$, where τ is the precision of the noise. We further assign a prior distribution over the latent factors, e.g., $p(\mathbf{u}^k_j) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and a prior over precision τ, e.g., $p(\tau) = \text{Gamma}(\alpha, \beta)$. Then the joint probability of Bayesian tensor decomposition is:

$$p_f(y_{\mathbf{i}}, \mathcal{U}, \tau) = p(\mathcal{U})p(\tau)\mathcal{N}(y_{\mathbf{i}} \mid f(\mathbf{u}^1_{i_1}, \ldots, \mathbf{u}^K_{i_K}), \tau^{-1}). \tag{2.4}$$

In the Bayesian framework, since the latent factors $\mathcal{U}$ and the noise precision τ are treated as random variables, their posterior distribution can be inferred through Bayesian inference. This allows for the estimation of the uncertainty in the latent factors and the noise precision, providing more robust decomposition and prediction results.

2.2 Bayesian Sequential Inference

Bayesian framework also provides a principled way to handle dynamic data, such as streaming data, temporal data, and general latent dynamics. In the Bayesian framework, the posterior distribution of the latent factors and the noise precision can be updated sequentially as new data arrives, allowing for the efficient and scalable analysis of dynamic sequence data, which is known as Bayesian sequential inference or online learning, based on the incremental version of Bayes' rule:

$$p\left(\boldsymbol{\theta} \mid \mathcal{D}_{\text{old}} \cup \mathcal{D}_{\text{new}}\right) \propto p\left(\boldsymbol{\theta} \mid \mathcal{D}_{\text{old}}\right) p\left(\mathcal{D}_{\text{new}} \mid \boldsymbol{\theta}\right), \tag{2.5}$$

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

There are several methods that can be used to implement Bayesian sequential inference via the incremental version of Bayes' rule (2.5), such as Assumed Density Filtering (ADF) [9], streaming variational inference methods [10], and classical filtering methods like Kalman filtering [41].

Among these methods, Assumed Density Filtering (ADF) is a popular choice for Bayesian online learning, as it provides a tractable and efficient way to approximate the posterior distribution. ADF can be viewed as an online version of expectation propagation (EP) [55].

The key idea of ADF is to approximate the posterior distribution $p\left(\boldsymbol{\theta} \mid \mathcal{D}_{\text{old}}\right)$ of the parameters $\boldsymbol{\theta}$ by a tractable distribution $q_{\text{old}}\left(\boldsymbol{\theta}\right)$ in the exponential family, e.g., Gaussian distribution. When the new data $\mathcal{D}_{\text{new}}$ arrives, ADF integrates $q_{\text{old}}\left(\boldsymbol{\theta}\right)$ with the data likelihood $p\left(\mathcal{D}_{\text{new}} \mid \boldsymbol{\theta}\right)$ and builds 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 $\tilde{p}(\boldsymbol{\theta})$ back to the exponential family to obtain the updated posterior approximation $q_{\text{new}}\left(\boldsymbol{\theta}\right)$ through moment matching. Finally,

ADF updates the current approximation $q_{\text{old}}(\boldsymbol{\theta})$ to $q_{\text{new}}(\boldsymbol{\theta})$, which serves as an approximation to $p(\boldsymbol{\theta} \mid \mathcal{D}_{\text{old}} \cup \mathcal{D}_{\text{new}})$. Thanks to the property of exponential family distribution, when we use Gaussian, the projection of $\tilde{p}(\boldsymbol{\theta})$ to the $q_{\text{new}}(\boldsymbol{\theta})$ can be done analytically [56]:

$$\mu^* = \mu + v\frac{\partial \log Z}{\partial \mu}, \tag{2.6}$$

$$v^* = v - v^2\left[\left(\frac{\partial \log Z}{\partial \mu}\right)^2 - 2\frac{\partial \log Z}{\partial v}\right], \tag{2.7}$$

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

2.3 Gaussian Process

Gaussian Process (GP) [71] is a powerful non-parametric Bayesian model that can flexibly estimate 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 the target function $f(\cdot)$ as a random process, i.e., Gaussian Process,

$$f \sim \mathcal{GP}(m(\cdot), \kappa(\cdot,\cdot)), \tag{2.8}$$

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 target function, often 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 inputs, often chosen as a Mercer kernel function. One popular and powerful kernel is the Matérn kernel:

$$\kappa(x,x') = \sigma^2\frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}}{l}\Delta\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}}{l}\Delta\right), \tag{2.9}$$

where Δ is the distance between x and x', often set as the Euclidean distance: $\Delta = |x - x'|$, ν is the degree of freedom controlling the smoothness, often chosen as a half-integer positive number, e.g., $\frac{5}{2}$, $\{l, \sigma^2\}$ are the length-scale and magnitude parameters, respectively, Γ is the gamma function, and K_ν is the modified Bessel function of the second kind. With a finite number of N observations of $\{(x_i, y_i)\}_{i=1}^N$, GP prior (2.8) will become a multivariate Gaussian distribution: $p(f(x)) = \mathcal{N}(\mathbf{0}, \mathbf{K})$, where $\mathbf{K} \in \mathbb{R}^{N\times N}$ is the kernel matrix, whose element $[K]_{i,j}$ is computed through the kernel function $\kappa(x_i, x_j)$.

GP is a powerful tool for modeling complex functions and capturing uncertainty in prediction. It is widely used in machine learning, statistics, and other fields for regression, classification, and other tasks [20, 52, 81, 80]. It is particularly useful in scenarios where the target function is unknown or difficult to model using traditional parametric methods.

However, a major challenge of GP is the computational complexity of inference, which can be expensive for large datasets or high-dimensional input spaces. However, recent advances in scalable GP, such as sparse GP [83], variational GP [90], and state-space GP [33, 74], have enabled GP to large-scale datasets and high-dimensional input spaces. Our research adopts GP to model latent dynamics in tensor learning, and we will introduce the details in the following chapters.

CHAPTER 3

BAYESIAN STREAMING SPARSE TUCKER DECOMPOSITION

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

3.1 Introduction

Multiway or multi-relation data are ubiquitous in real-world applications. Those data are naturally represented by tensors. For example, we can use a three-mode tensor, *(customer, product, page)*, to describe online shopping records. In order to analyze these tensors, tensor factorization provides a fundamental framework, which introduces a set of latent factors (or properties) to represent the nodes/entities in each mode. These latent factors are estimated by minimizing the reconstruction error of the observed tensor entries. Given

the latent factors, we can discover the hidden patterns among the tensor nodes, such as communities and outliers. We can also use these latent factors as effective features for downstream tasks, such as commodity recommendation and click-through-rate prediction.

The most commonly used tensor decomposition approach is (probably) the CANDECOMP/PARAFAC (CP) decomposition [32], which demands all the tensor nodes (across different modes) have the same number of latent factors R. CP assumes each tensor element is calculated by a summation of R products, where the r-th product is computed from the r-th latent factor of each involved node ($1 \leq r \leq R$). Despite the convenience, CP model can be quite restrictive in that it only considers R possible interactions between the latent factors. By contrast, Tucker decomposition [91] is much more expressive. It allows the nodes in each mode k to have a different factor number R_k, and uses a linear combination of all possible $R_1 \times \cdots \times R_K$ interactions across the factors to model each entry value. The weights can be organized as an $R_1 \times \cdots \times R_K$ core tensor $\mathcal{W}$, and jointly estimated with the latent factors. CP decomposition can therefore be viewed as a special instance of Tucker decomposition with $R_1 = \ldots = R_K$ and $\mathcal{W}$ being diagonal. Hence, Tucker decomposition enables more flexibility and interpretability than CP decomposition.

However, in the mean time, Tucker decomposition also brings challenges in both modeling and computation. First, assuming that every possible interaction of the latent factors is taking effect might make the model overly complex, and enhance the risk of overfitting. Second, the model estimation is much more expensive (considering a relatively large core tensor $\mathcal{W}$), especially in the case of fast streaming data, which are ubiquitous in real-world applications [21]. It is very costly to estimate the latent factors and core tensor from scratch every time when a batch of new entries are received. Some applications that stress on privacy (*e.g.*, SnapChat) even forbid us from re-accessing previous data. Therefore, an efficient incremental update is in urgent need.

To address these issues, we develop BASS-Tucker, a Bayesian streaming sparse Tucker decomposition approach. We propose a Bayesian formulation of the Tucker decomposition, and assign a spike-and-slab prior over each element of the core tensor $\mathcal{W}$ so as to identify the effective factor interactions in the generation (reconstruction) of the entry values. Accordingly, we can reduce the model complexity and further enhance the interpretability.

Second, to efficiently deal with streaming data, we extend the assumed density filtering (ADF) framework [9] to jointly process each batch of incoming tensor entries. We use the conditional moment matching and delta method [6] to address the intractable moment calculation, and develop one-shot incremental posterior update for the latent factors and core tensor upon receiving each streaming batch. In so doing, we avoid sequentially processing every data point as in standard ADF, which requires us to repeatedly construct new approximations, match the moments, and is much more expensive. Next, to enable effective sparse estimation, we explicitly introduce a spike-and-slab approximation in the running posterior. By updating the approximation every a few batches, the sparse regularization of the spike-and-slab prior is constantly transmitted and reinforced in the running posterior, with which we can effectively select meaningful factor interactions during the streaming decomposition. Our incremental updates are closed-form, reliable and efficient.

We examined BASS-Tucker in four real-world applications. We compared with the state-of-the-art multilinear streaming decomposition algorithm, POST [21], and static CP/-Tucker decomposition algorithms that need to repeatedly access the data. In both running and final predictive performance, BASS-Tucker consistently outperforms POST, often by a large margin. The final prediction accuracy of BASS-Tucker is even significantly better than the state-of-the-art static decomposition algorithms. BASS-Tucker also enjoys a linear scalability in the data size. Finally, BASS-Tucker identifies sparse structures in the core tensor, which reflects interesting patterns within the factor interactions in reconstructing the tensor entries.

3.2 Background and Notation

3.2.1 Tensor Decomposition

Denote a K-mode tensor by $\mathcal{Y} \in \mathbb{R}^{d_1 \times \ldots \times d_K}$, where mode k contains d_k nodes (or entities). Each tensor entry is indexed by a K-element tuple, $\mathbf{i} = (i_1, \ldots, i_K)$, where each i_k ($1 \leq k \leq K$) is the index of the involved node in mode k. The entry value is denoted by $y_{\mathbf{i}}$. To conduct tensor decomposition, we introduce a set of latent factors to represent the nodes in each mode. These factors may correspond to intrinsic properties or features of the nodes. Suppose in each mode k, we have R_k latent factors for every node. We can

arrange these latent factors into K factor matrices, $\mathcal{U} = \{\mathbf{U}^1, \ldots, \mathbf{U}^K\}$, where each $\mathbf{U}^k$ is of size $d_k \times R_k$ and each row corresponds to one node. Our goal is to estimate $\mathcal{U}$ to recover the observed entry values in $\mathcal{Y}$. The most commonly used tensor decomposition model is CANDECOMP/PARAFAC (CP) decomposition [32] which assumes $R_1 = \ldots = R_K = R$, and $\mathcal{Y} \approx \sum_{r=1}^{R} \lambda_r \cdot \big(\mathbf{U}^1(:, r) \otimes \ldots \otimes \mathbf{U}^K(:, r)\big)$, where $\otimes$ is the Kronecker product, and $\mathbf{U}^k(:, r)$ is the r-th column of $\mathbf{U}^k$. The element-wise form is

$$y_{\mathbf{i}} \approx \sum_{r=1}^{R} \lambda_r \prod_{k=1}^{K} u^k_{i_k, r}, \tag{3.1}$$

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

A much more expressive model is Tucker decomposition model [91], which takes into account all possible factor interactions. Specifically, we introduce an $R_1 \times \ldots \times R_K$ core tensor $\mathcal{W}$ and assume that

$$\mathcal{Y} \approx \mathcal{W} \times_1 \mathbf{U}^1 \times_2 \ldots \times_K \mathbf{U}^K, \tag{3.2}$$

where $\times_k$ the mode-k tensor matrix product [45], $\mathcal{W} \times_k \mathbf{U}^k$ results in an $R_1 \times \ldots \times R_{k-1} \times d_k \times R_{k+1} \times \ldots \times R_K$ tensor and each element is calculated by:

$$\Big[\mathcal{W} \times_k \mathbf{U}^k\Big]_{i_1,\ldots,i_K} = \sum_{j=1}^{R_k} w_{(i_1,\ldots,i_{k-1},j,i_{k+1},\ldots,i_K)} \cdot u^k_{i_k,j}. \tag{3.3}$$

The element-wise form is given by

$$y_{\mathbf{i}} \approx \sum_{r_1=1}^{R_1} \cdots \sum_{r_K=1}^{R_K} \left[w_{(r_1,\ldots,r_K)} \cdot \prod_{k=1}^{K} u^k_{i_k,r_k} \right]. \tag{3.4}$$

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

3.2.2 Streaming Model Inference

Streaming variational Bayes (SVB) [10] is a popular approach to conduct incremental posterior inference given data streams. It is based upon the incremental Bayes' rule:

$$p(\boldsymbol{\theta}|\mathcal{D}_{\text{old}} \cup \mathcal{D}_{\text{new}}) \propto p(\boldsymbol{\theta}|\mathcal{D}_{\text{old}})p(\mathcal{D}_{\text{new}}|\boldsymbol{\theta}), \tag{3.5}$$

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

$$\tilde{p}(\boldsymbol{\theta}) = q_{\text{cur}}(\boldsymbol{\theta})p(\mathcal{D}_{\text{new}}|\boldsymbol{\theta}), \tag{3.6}$$

which is considered as approximately proportional to the joint distribution $p(\boldsymbol{\theta}, \mathcal{D}_{\text{old}} \cup \mathcal{D}_{\text{new}})$. The latter, however, is infeasible to compute in the streaming scenario. Therefore, SVB uses $\tilde{p}(\boldsymbol{\theta})$ to construct a variational evidence lower bound (ELBO) [92], $\mathcal{L}(q(\boldsymbol{\theta})) = \mathbb{E}_q\left[\log\left(\frac{\tilde{p}(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right)\right]$, and maximizes the ELBO to update the posterior, $q_{\text{cur}} \leftarrow \text{argmax}_q \mathcal{L}(q)$. This is equivalent to minimizing Kullback-Leibler (KL) divergence between q and $\tilde{p}$. We continue this procedure to conduct incremental updates for the new incoming data. At the beginning, we set $q_{\text{cur}} = p(\boldsymbol{\theta})$, the original prior in the model. For efficiency and convenience, we usually use a mean-field variational posterior, $q(\boldsymbol{\theta}) = \prod_j q(\theta_j)$, and alternately update each $q(\boldsymbol{\theta}_j)$, which is often closed-form. The state-of-the-art streaming CP decomposition algorithm, POST [21], applies SVB in a Bayesian CP model to incrementally update the posterior of the factors upon receiving new tensor entries.

3.3 Model

Tucker decomposition is much more flexible than the most commonly used CP decomposition in that it combines all possible (multiplicative) factor interactions across the modes (see (3.4)) to reconstruct the tensor entries. Therefore, Tucker decomposition can capture as many effective and interpretable patterns as possible. However, such flexibility can also cause significant challenges in modeling and computation. First, assuming every

factor interaction is taking effect in generating the entry values can make the model overly complex, especially given that in most applications, the data are extremely sparse, *i.e.*, the number of observed entries is far less than the tensor size. This modeling assumption can increase the risk of overfitting. Second, we have to estimate an extra parametric core-tensor $\mathcal{W}$, of size $R_1 \times \ldots \times R_K$, which is relatively large. The joint estimation of $\mathcal{W}$ and $\mathcal{U}$ can be much more costly, especially when we handle streaming data, where each newly observed entry might incur a whole update of $\mathcal{W}$. To address these challenges, we propose a Bayesian sparse Tucker decomposition model and develop an efficient one-shot streaming posterior inference algorithm (see Section 3.4).

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

$$p(\mathcal{U}) = \prod_{k=1}^{K} \prod_{j=1}^{d_k} \mathcal{N}(\mathbf{u}_j^k | \mathbf{0}, \mathbf{I}). \tag{3.7}$$

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

$$p(\mathcal{S}|\rho_0) = \prod_{\mathbf{j}=(1,\ldots,1)}^{(R_1,\ldots,R_K)} \text{Bern}(s_{\mathbf{j}}|\rho_0), \tag{3.8}$$

$$p(\mathcal{W}|\mathcal{S}) = \prod_{\mathbf{j}} s_{\mathbf{j}} \mathcal{N}(w_{\mathbf{j}}|0, \sigma_0^2) + (1 - s_{\mathbf{j}})\delta(w_{\mathbf{j}}), \tag{3.9}$$

where $\mathcal{S}$ is a binary tensor of the same size with $\mathcal{W}$, $\text{Bern}(\cdot)$ 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

$$\begin{aligned} p(y_{\mathbf{i}}|\mathcal{U}, \mathcal{W}, \tau) &= \mathcal{N}\left(y_{\mathbf{i}}|\mathcal{W} \times_1 \left(\mathbf{u}_{i_1}^1\right)^\top \times_2 \ldots \times_K \left(\mathbf{u}_{i_K}^K\right)^\top, \tau^{-1}\right), \\ &= \mathcal{N}\left(y_{\mathbf{i}}| \sum_{j_1=1}^{R_1} \ldots \sum_{j_K=1}^{R_K} \left[w_{(j_1,\ldots,j_K)} \cdot \prod_{k=1}^{K} u_{i_k,j_k}^k\right], \tau^{-1}\right), \end{aligned} \tag{3.10}$$

where τ is the inverse of the noise variance. Now, combining with (3.9), 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, $p(\tau) = \text{Gam}(\tau|a_0, b_0)$, where a_0, b_0 are the hyperparameters of the prior distribution. Denote the observed tensor entries by $\mathcal{F}$. The joint probability of our model is given by

$$\begin{aligned} p(\mathcal{S}, \mathcal{W}, \mathcal{U}, \mathcal{Y}, \tau) = & \prod_k \prod_j \mathcal{N}(\mathbf{u}_j^k | \mathbf{0}, \mathbf{I}) \text{Gam}(\tau | a_0, b_0) \\ & \cdot \prod_{\mathbf{j}} \text{Bern}(s_{\mathbf{j}} | \rho_0) \left(s_{\mathbf{j}} \mathcal{N}(w_{\mathbf{j}} | 0, \sigma_0^2) + (1 - s_{\mathbf{j}}) \delta(w_{\mathbf{j}}) \right) \\ & \cdot \prod_{\mathbf{i} \in \mathcal{F}} \mathcal{N}(y_{\mathbf{i}} | \mathcal{W} \times_1 \left(\mathbf{u}_{i_1}^1 \right)^\top \times_2 \ldots \times_K \left(\mathbf{u}_{i_K}^K \right)^\top, \tau^{-1}). \end{aligned} \tag{3.11}$$

3.4 Streaming Inference

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

One might consider applying the SVB framework discussed in Section 5.2. However, a severe problem is that SVB cannot well integrate the spike-and-slab (SS) prior in (3.9) with the streaming data to effectively estimate the sparse posterior of the core tensor $\mathcal{W}$. The reason is that the spike-and-slab prior is a mixture prior and we cannot set q_{cur} to the SS prior at the beginning. Instead, we have to choose a much simpler form for q_{cur} (in the exponential family) to ensure a tractable update, *e.g.*, a mean-field form $q_{\text{cur}}(\mathcal{W}, \mathcal{S}, \mathcal{U}) = q_{\text{cur}}(\mathcal{W}) q_{\text{cur}}(\mathcal{S}) q_{\text{cur}}(\mathcal{U})$. The SS prior will only take effect in dealing with the very first batch of entries to obtain the first estimate of q_{cur}. After that, the prior will be replaced by q_{cur}, which will repeatedly integrate with the subsequent streaming batches to update itself (see (3.5)). Hence, in the subsequent updates, the SS prior is separated from the data likelihoods, and cannot perform any sparse regularization in the model update. The regularization is performed by q_{cur} itself, which is much weaker than the original SS prior. For example, if we use the aforementioned mean-field form for q_{cur}, the selection indicators $\mathcal{S}$ and core tensor elements $\mathcal{W}$ are assumed to be independent.

It does not encourage any sparsity in $\mathcal{W}$. In addition, for each streaming batch, SVB needs to iteratively update each component in the mean-field posterior until convergence, and hence can be quite expensive, especially for the relatively large core-tensor $\mathcal{W}$.

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

3.4.1 Refining Spike-and-Slab Prior Approximation in the Running Posterior

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

$$q_{\text{cur}}(\mathcal{W},\mathcal{U},\tau) \propto p(\mathcal{S})\xi(\mathcal{W},\mathcal{S}) \cdot \prod_{k=1}^{K}\prod_{j=1}^{d_k}\mathcal{N}(\mathbf{u}_j^k|\boldsymbol{\mu}_j^k,\mathbf{V}_j^k) \cdot \mathcal{N}\left(\text{vec}(\mathcal{W})|\boldsymbol{\mu},\boldsymbol{\Sigma}\right)\text{Gam}(\tau|a,b), \tag{3.12}$$

where $\xi(\mathcal{W},\mathcal{S})$ is an approximation to the SS prior in (3.9),

$$\xi(\mathcal{W},\mathcal{S}) = \prod_{\mathbf{j}}\text{Bern}\left(s_{\mathbf{j}}|c(\rho_{\mathbf{j}})\right)\mathcal{N}(w_{\mathbf{j}}|m_{\mathbf{j}},\eta_{\mathbf{j}}) \lessapprox p(\mathcal{W}|\mathcal{S}), \tag{3.13}$$

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 (3.11), we can see that the terms other than $p(\mathcal{S})\xi(\mathcal{S},\mathcal{W})$ in (3.12) essentially integrate (or summarize) the prior of $\mathcal{U}$ and τ and the likelihood of all the data points that have been seen so far.

Similar to (3.6), given an incoming batch $\mathcal{B}_n$, we combine the current posterior with the new data likelihood to construct a blending distribution,

$$\tilde{p}(\mathcal{U},\mathcal{W},\tau) = q_{\text{cur}}(\mathcal{U},\mathcal{W},\tau)\prod_{\mathbf{i}\in\mathcal{B}_n}p(y_{\mathbf{i}}|\mathcal{U},\mathcal{W},\tau), \tag{3.14}$$

which is an approximation to the joint distribution of all the data (see (3.11)). We use the idea of moment matching and projection to update $\mathcal{N}\left(\text{vec}(\mathcal{W})|\mathbf{m},\boldsymbol{\Sigma}\right)$, $\text{Gam}(\tau|a,b)$, and the associated $\mathcal{N}(\mathbf{u}_j^k|\boldsymbol{\mu}_j^k,\mathbf{V}_j^k)$. The details will be discussed in Section 3.4.2. Then after

every a few batches, we update $\xi(\mathcal{W}, \mathcal{S})$ to find the best approximation of the SS prior under the current data context (reflected in the other terms in q_{cur}). In this way, the sparse regularization effect of the SS prior can be injected and reinforced in q_{cur}, and leveraged in the subsequent posterior updates.

The update of $\xi(\mathcal{W}, \mathcal{S})$ resembles standard expectation propagation [55]. We update each ξ_j in parallel. We first divide the marginal posterior by the prior approximation to obtain the calibrated (or context) distribution,

$$q^{\setminus}(w_{\mathbf{j}}, s_{\mathbf{j}}) \propto \frac{q_{\text{cur}}(w_{\mathbf{j}}, s_{\mathbf{j}})}{\xi_j(w_{\mathbf{j}}, s_{\mathbf{j}})} = \text{Bern}(s_{\mathbf{j}}|\rho_0)\mathcal{N}(w_{\mathbf{j}}|\mu_{\mathbf{j}}^{\setminus}, v_{\mathbf{j}}^{\setminus}). \tag{3.15}$$

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

$$\tilde{p}(w_{\mathbf{j}}, s_{\mathbf{j}}) \propto q^{\setminus}(w_{\mathbf{j}}, s_{\mathbf{j}})\big(s_{\mathbf{j}}\mathcal{N}(w_{\mathbf{j}}|0, \sigma_0^2) + (1 - s_{\mathbf{j}})\delta(w_{\mathbf{j}})\big). \tag{3.16}$$

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

$$q^*(w_{\mathbf{j}}, s_{\mathbf{j}}) = \text{Bern}(s_{\mathbf{j}}|c(\rho_{\mathbf{j}}^*))\mathcal{N}(w_{\mathbf{j}}|\mu_{\mathbf{j}}^*, v_{\mathbf{j}}^*), \tag{3.17}$$

where $\rho_{\mathbf{j}}^* = \log\Big(\mathcal{N}(\mu_{\mathbf{j}}^{\setminus}|0, \sigma_0^2 + v_{\mathbf{j}}^{\setminus})/\mathcal{N}(\mu_{\mathbf{j}}^{\setminus}|0, v_{\mathbf{j}}^{\setminus})\Big) + c^{-1}(\rho_0)$, $\mu_{\mathbf{j}}^* = c(\rho_{\mathbf{j}}^*)\widehat{\mu}_{\mathbf{j}}$, $v_{\mathbf{j}}^* = c(\rho_{\mathbf{j}}^*)\big(\widehat{v}_{\mathbf{j}} + (1 - c(\rho_{\mathbf{j}}^*))\widehat{\mu}_{\mathbf{j}}^2\big)$, where, $\widehat{v}_{\mathbf{j}} = \big((v_{\mathbf{j}}^{\setminus})^{-1} + \sigma_0^{-2}\big)^{-1}$, 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

$$\xi_{\mathbf{j}}(\mathbf{w}_{\mathbf{j}}, \mathbf{s}_{\mathbf{j}}) \leftarrow q^*(w_{\mathbf{j}}, s_{\mathbf{j}})/q^{\setminus}(w_{\mathbf{j}}, s_{\mathbf{j}}). \tag{3.18}$$

3.4.2 One-Shot Incremental Update

Now, let us look at how to update the remaining terms in (3.12). As mentioned in Section 3.4.1, upon the arrival of a streaming batch $\mathcal{B}_n$, we will construct the blending distribution in (3.14), which combines the prior, the information of the previous data (reflected in q_{cur}), and the information of new data. We use the ADF framework that projects the blending distribution onto the exponential family to obtain the updated posterior. This essentially is to minimize $\text{KL}(\frac{1}{Z}\tilde{p}\|q)$ where Z is the normalization constant.

The optimal q is obtained by moment matching. Standard ADF only subsumes one data point at each step to construct the blending distribution and perform projection. Hence, it needs to sequentially process each entry in $\mathcal{B}_n$ and repeatedly match moments. This one-by-one strategy is inefficient, because we need to match the moments of the core tensor $\mathcal{W}$ many times, which can be very costly. In addition, repeatedly constructing approximations by projection can affect the quality of posterior updates. Therefore, we jointly integrate the whole streaming batch $\mathcal{B}_n$ via (3.14), and perform moment-matching only once. Our one-shot posterior update not only is much more efficient, but also avoids multiple approximation steps and hence can improve the quality of posterior estimation.

However, a critical bottleneck is that the moment of the blending distribution $\tilde{p}$ in (3.14) is analytically intractable, due to the complex coupling of $\mathcal{U}$ and $\mathcal{W}$ in the likelihood (see (3.10)). To overcome this problem, we first perform conditional moment matching, and then seek to calculate the expected conditional moments. Specifically, let us consider the update of $\mathcal{N}\left(\text{vec}(\mathcal{W})|\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$ in (3.12). We denote by Θ all the latent random variables in our model, and define $\Theta_{\backslash \mathcal{W}} = \Theta \backslash \{\mathcal{W}\}$. Our key observation is

$$\mathbb{E}_{\tilde{p}}\left[\boldsymbol{\phi}(\mathcal{W})\right] = \mathbb{E}_{\tilde{p}(\Theta_{\backslash \mathcal{W}})}\mathbb{E}_{\tilde{p}(\mathcal{W}|\Theta_{\backslash \mathcal{W}})}\left[\boldsymbol{\phi}(\mathcal{W})|\Theta_{\backslash \mathcal{W}}\right], \tag{3.19}$$

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

$$\begin{aligned}\tilde{p}(\mathcal{W}|\Theta_{\backslash \mathcal{W}}) &\propto \mathcal{N}\left(\text{vec}(\mathcal{W})|\mathbf{m}, \text{diag}(\boldsymbol{\eta})\right)\mathcal{N}\left(\text{vec}(\mathcal{W})|\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \\ &\cdot \prod_{\mathbf{i}\in\mathcal{F}} \mathcal{N}\left(y_{\mathbf{i}}|\left(\left(\mathbf{u}^K_{i_K}\right)^\top \otimes \ldots \otimes \left(\mathbf{u}^1_{i_1}\right)^\top\right)\text{vec}(\mathcal{W}), \tau^{-1}\right),\end{aligned} \tag{3.20}$$

where $\mathbf{m}$ is the concatenation of all $\{m_{\mathbf{j}}\}$ and $\boldsymbol{\eta}$ is the concatenation of all $\{\eta_{\mathbf{j}}\}$ (see (3.13)). Note that the mean in the Gaussian of $y_{\mathbf{i}}$ is obtained from the tensor algebra [45]. Together this gives another Gaussian distribution, and we can immediately derive the closed-form conditional moments,

$$\mathbb{E}\left[\text{vec}(\mathcal{W})|\Theta_{\backslash \mathcal{W}}\right] \propto \Omega^{-1}\left(\sum_{\mathbf{i}} \tau y_{\mathbf{i}}\mathbf{b}_i + \text{diag}\left(\frac{\mathbf{m}}{\boldsymbol{\eta}}\right) + \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right), \tag{3.21}$$

$$\mathbb{E}\left[\text{vec}(\mathcal{W})\text{vec}(\mathcal{W})^\top|\Theta_{\backslash \mathcal{W}}\right] \propto \Omega^{-1} + \mathbb{E}\left[\text{vec}(\mathcal{W})|\Theta_{\backslash \mathcal{W}}\right]\mathbb{E}\left[\text{vec}(\mathcal{W})|\Theta_{\backslash \mathcal{W}}\right]^\top, \tag{3.22}$$

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

To obtain the moment, we need to take the expectation of the conditional moment under the marginal blending distribution, $\tilde{p}(\Theta_{\setminus\mathcal{W}})$, namely the outer expectation in (3.19). However, this is infeasible, because $\tilde{p}(\Theta_{\setminus\mathcal{W}})$ is analytically intractable. To overcome this problem, we observe that due to the factorization structure of q_{cur} in (3.12), we also maintain the moment matching between $q_{\text{cur}}(\Theta_{\setminus\mathcal{W}})$ and $\tilde{p}(\Theta_{\setminus\mathcal{W}})$, hence we can assume they are close in high-density regions. In addition, since $\tilde{p}$ is an integration of q_{cur} and a small batch of new data points, when many streaming batches have been processed, adding one more batch is unlikely to significantly change the current posterior, especially in the high-density regions. Therefore, we can use $q_{\text{cur}}(\Theta_{\setminus\mathcal{W}})$ to approximate $\tilde{p}(\Theta_{\setminus\mathcal{W}})$, and calculate the expected conditional moments. For notional convenience, we denote all the conditional moments of $\mathcal{W}$ by $\mathbf{h}(\Theta_{\setminus\mathcal{W}})$. We now resort to calculate

$$\mathbb{E}_{\tilde{p}}\left[\boldsymbol{\phi}(\mathcal{W})\right] \approx \mathbb{E}_{q_{\text{cur}}(\Theta_{\setminus\mathcal{W}})}\left[\mathbf{h}(\Theta_{\setminus\mathcal{W}})\right]. \tag{3.23}$$

This is still intractable, because the conditional moments $\mathbf{h}$ is a nonlinear function of the remaining variables $\Theta_{\setminus\mathcal{W}}$ (see (3.22)). However, with the nice form of q_{cur}, we can use the delta method [59, 6] to calculate the expectation. We present $\mathbf{h}$ with the first-order Taylor approximation at the mean,

$$\mathbf{h}(\Theta_{\setminus\mathcal{W}}) \approx \mathbf{h}\left(\mathbb{E}_{q_{\text{cur}}}[\Theta_{\setminus\mathcal{W}}]\right) + \mathbf{J}\left(\mathbb{E}_{q_{\text{cur}}}[\Theta_{\setminus\mathcal{W}}]\right)\left(\text{vec}\left(\Theta_{\setminus\mathcal{W}}\right) - \text{vec}\left(\mathbb{E}_{q_{\text{cur}}}[\Theta_{\setminus\mathcal{W}}]\right)\right), \tag{3.24}$$

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

$$\mathbb{E}_{q_{\text{cur}}(\Theta_{\setminus\mathcal{W}})}\left[\mathbf{h}(\Theta_{\setminus\mathcal{W}})\right] \approx \mathbf{h}\left(\mathbb{E}_{q_{\text{cur}}}[\Theta_{\setminus\mathcal{W}}]\right). \tag{3.25}$$

Therefore, we can simply substitute the current expectation or moments for $\Theta_{\setminus\mathcal{W}}$ in the conditional moments of $\mathcal{W}$. The computation is convenient and efficient. The theoretical analysis and guarantees about the delta method are discussed in previous work [59, 98].

We then use the moments calculated from (3.25) to construct the updated marginal posterior $q^*(\mathcal{W}) = \mathcal{N}\left(\text{vec}(\mathcal{W})|\boldsymbol{\beta}^*, \mathbf{G}^*\right)$. Contrasting to (3.12), we update with

$$\mathcal{N}\left(\text{vec}(\mathcal{W})|\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \propto \frac{\mathcal{N}(\text{vec}(\mathcal{W})|\boldsymbol{\beta}^*, \mathbf{G}^*)}{\mathcal{N}\left(\text{vec}(\mathcal{W})|\mathbf{m}, \text{diag}(\boldsymbol{\eta})\right)}. \tag{3.26}$$

In parallel, we use the same approach to update $\text{Gam}(\tau|a, b)$ and all the Gaussian terms $\mathcal{N}(\mathbf{u}_j^k|\boldsymbol{\mu}_j^k, \mathbf{V}_j^k)$ in (3.12) associated with the current batch $\mathcal{B}_n$, namely, those corresponding

to the entries of $\mathcal{B}_n$. Note that if a latent factor has no relevant tensor entries observed in $\mathcal{B}_n$, we do not update its posterior, because we do not have data to improve it.

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

3.4.3 Algorithm Complexity

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

3.5 Related Work

As a natural extension of matrix decomposition, tensor decomposition is a classical and important topic in machine learning. Canonical approaches include CP [32] and Tucker [91] decomposition. Although numerous other methods have also been developed, such as [82, 16, 4, 85, 1, 37, 42, 102, 69, 15, 39, 77, 78, 60, 21], the majority of these methods still inherit the CP or Tucker form, not only for their elegance and convenience, but also due to the model transparency and interpretability. Recently, a few efforts have been made to develop nonlinear decomposition models based on Gaussian processes [71] or neural networks, such as [101, 108, 107, 106, 51]. Despite their power, these methods rely on

Algorithm 1 BASS-Tucker

1: Initialize the spike-and-slab prior approximation and multiply it with all the other priors to initialize $q_{\text{cur}}(\cdot)$.
2: **while** a new batch of tensor entries $\mathcal{B}_n$ arrives **do**
3: In parallel update $\mathcal{N}(\text{vec}(\mathcal{W})|\mathbf{m}, \mathbf{\Sigma})$, $\text{Gam}(\tau|a, b)$, and related $\mathcal{N}(\mathbf{u}_j^k|\boldsymbol{\mu}_j^k, \mathbf{V}_j^k)$ in (3.12) 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 (3.17) and (3.18).
6: **end if**
7: **end while**
8: **return** the current posterior $q_{\text{cur}}(\cdot)$.

black-box models and hence sacrifice the interpretability.

Expectation propagation [55] is a deterministic approximate inference approach that unifies assumed-density-filtering (ADF) [9] and (loopy) belief propagation [57]. EP approximates the prior and likelihood of each data point by a term from the exponential family. ADF can be considered as an online version of EP — it maintains a holistic posterior, and applies EP to update the model with one incoming data point at a time. Hence, ADF does not need to keep approximation terms for the past data points. EP is problematic when the moment computation is intractable. Recently, [96] proposed conditional EP to address this problem by conditional moment matching, Taylor approximation and numerical quadrature with fully factorized posteriors. Our work extends ADF to perform one-shot posterior update upon receiving a batch of streaming entries. Hence, we do not need to repeatedly conduct moment matching, which is quite expensive and might also impair the inference quality. To fulfill an efficient, tractable moment calculation, we use similar ideas as in [96]. In addition, another important difference from ADF is that we explicitly maintain the approximation term of the spike and slab prior in the running posterior, rather than absorb everything in a singleton term in the exponential family. In so doing, we are able to constantly update the prior approximation, transmit and reinforce the regularization effect of the original prior to ensure a high-quality sparse posterior estimate under streaming data, which is different from existing work on online sparse learning with spike and slab prior [29].

3.6 Experiment

3.6.1 Predictive Performance

3.6.1.1 Datasets

We evaluated BASS-Tucker in four real-world applications. (1) *Alog* [108], 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* [21], 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.

3.6.1.2 Competing Methods

We compared with the state-of-the-art (SOA) multilinear streaming tensor decomposition algorithm (1) POST [21], which is based on SVB [10]. In addition, we compared with SOA static multilinear decomposition methods, including (2) P-Tucker [60], a highly efficient Tucker decomposition algorithm that conducts row-wise updates in parallel, (3) CP-WOPT [1], CP decomposition by conjugate gradient descent, (4) CP-ALS [4], CP decomposition via alternating least square updates, (5) NNCP [47], non-negative CP decomposition with multiplicative updates, and (6) Tucker-ALS [17], Tucker decomposition with alternating least square updates.

3.6.1.3 Settings and Results

We implemented BASS-Tucker with MATLAB. To estimate a sparse core tensor $\mathcal{W}$, we set $\rho_0 = 0.5$ and $\sigma_0^2 = 1$ (see (3.9)). We used the original implementation of all the competing approaches and their default settings (*e.g.,* the maximum number of iterations for static decomposition). We first evaluated the final predictive performance, namely when all the streamed entries have been processed. To this end, we randomly split the observed entries of *Alog* into 75% for training and 25% for test and the other datasets 90% for training and 10% for test. For BASS-Tucker and POST, we randomly partitioned the training entries into a stream of small batches. The other methods conducted iterative static decomposition and need to repeatedly access the whole data. We repeated the experiment for 5 times, and calculated the average root-mean-squared-error (RMSE) for the real-valued datasets and area under ROC curve (AUC) for the binary dataset. The average RMSE/AUC and standard deviation are reported in Fig. 3.1. In Fig. 3.1a-d, we fixed the streaming batch size to 512, and show how the final predictive performance of each method varied with different factor numbers, $\{3, 5, 7, 9\}$. In Fig. 3.1e-h, we fixed the factor number to 5, and examined the performance under different batch sizes, $\{2^8, 2^9, 2^{10}, 2^{11}\}$. The more the factors/larger the streaming batch size, the more expensive for BASS-Tucker to factorize each batch. Hence these settings examined the trade-off between the accuracy and computational complexity.

As we can see, in all the cases BASS-Tucker outperforms SOA streaming approach POST and SOA static decomposition methods except that in Fig. 3.1c, BASS-Tucker is slightly worse than P-Tucker when the number factor is 9. In most cases, BASS-Tucker shows an significant improvement ($p < 0.05$). Note that P-Tucker estimates a dense core-tensor and needs to access the data many time. The results demonstrate the advantage of BASS-Tucker (including sparse core tensor estimation) in prediction accuracy.

3.6.2 Prediction on the Fly

Next, we evaluated the running predictive performance of BASS-Tucker. We fixed the batch size to 512, and randomly streamed the training entries to BASS-Tucker and POST. We tested the prediction accuracy after each streaming batch was processed, with the factor number $R = 5$ and $R = 7$. The running RMSE/AUC is shown in Fig. 3.2. In the top row, the factor number is fixed to 5; in the bottom row, the factor number is fixed to 7. The batch size is fixed to 512. As we can see, at the beginning, POST is close to or even better than BASS-Tucker in prediction accuracy. This is reasonable, because our Tucker model is much more complex than the CP model. However, at later stages, BASS-Tucker consistently outperforms POST, mostly by a large margin. Even in Fig. 3.2c and g, while POST and BASS-Tucker overlapped quite a long time, BASS-Tucker beat POST when the data stream is about to be finished.

In addition, we examined the scalability of BASS-Tucker. On *ACC*, we fixed the batch size to 512, and streamed 30%, 50% and 100% of observed entries to BASS-Tucker and tested the streaming decomposition time. We varied the factor number R from $\{5, 7, 9, 11\}$. As we can see in Fig. 3.3c, the running time of BASS-Tucker grows linearly in the number of streamed entries, and the factor number R determines the slope. Therefore, BASS-Tucker enjoys a linear scalability to the data size.

To evaluate if BASS-Tucker can indeed estimate a sparse core-tensor. We varied ρ_0 from (0.1, 0.9) and ran BASS-Tucker on *Alog* dataset. An element of core-tensor $\mathcal{W}$ is viewed as selected if the posterior mean of its selection probability is no less than 0.5. We showed how the ratio of the unselected elements varies with ρ_0 in Fig. 3.3d under different factor numbers. We can see that small ρ_0 pruned most of the elements while large ρ_0 preserved most, showing that BASS-Tucker can effectively achieve sparsity in the streaming setting.

3.6.3 Sparse Structure in Core Tensor

Finally, we examined if the estimated sparse core-tensor by BASS-Tucker can reflect some structure, as compared with the standard Tucker decomposition. To this end, we set the number of latent factors to 9, and ran BASS-Tucker and P-Tucker on *Alog* dataset. The core tensor is of size $9 \times 9 \times 9$. Then, in each mode, we fold the core tensor to an 81×9 matrix, where each row indicates how strongly each factor combination from the other modes interact with the 9 factors in the current mode. We then ran Principled Component Analysis (PCA) and projected the interaction matrix onto a plane. The positions of the points represent the first and second principle components, which summarize how every factor combination from the other modes interact with the factors in the current mode. We show the results for the first mode in Fig. 3.3a and b. As we can see, from BASS-Tucker, the interaction between the factors in the first mode and in the other modes clearly exhibit clustering structures, implying different interaction patterns. To show the structures, we ran the k-means algorithm and filled the cluster regions with different colors. By contrast, the core-tensor estimated by P-Tucker do not reflect apparent structures, and the interaction strengths are distributed like a symmetric Gaussian. While all the datasets have been completely anonymized and we are unable to look into the meaning of the patterns discovered by BASS-Tucker, these results have demonstrated that BASS-Tucker, as compared with standard Tucker decomposition, can potentially discover more interesting patterns and knowledge, and enhance the interpretability.

3.7 Conclusion

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

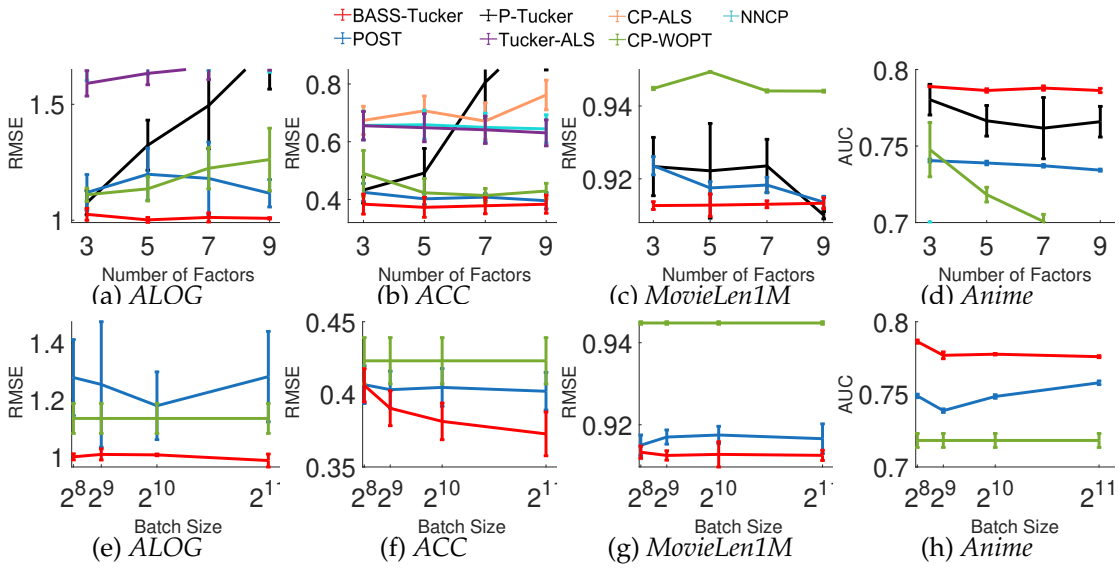


Figure 3.1: Predictive performance with different numbers of factors (top row) and streaming batch sizes (bottom row).

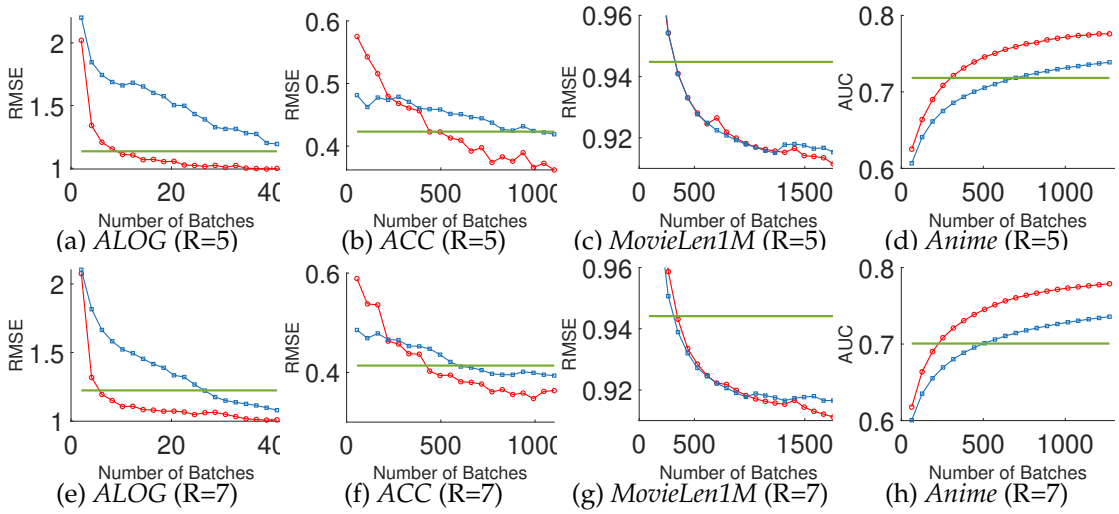
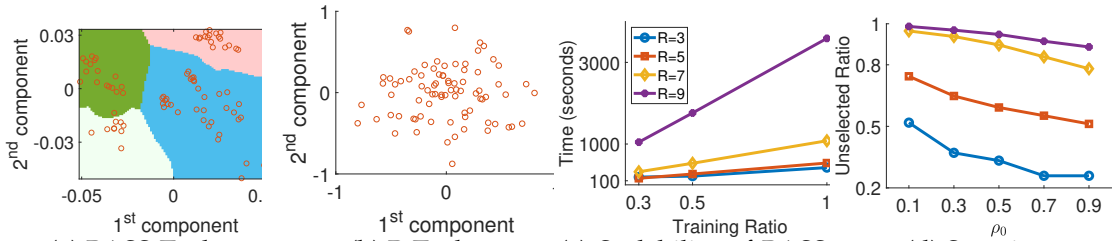


Figure 3.2: Running prediction accuracy along with the number of processed streaming batches.



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

Figure 3.3: The structures of the estimated core-tensor by BASS-Tucker and P-Tucker (a, b), the scalability of BASS-Tucker (c) and the sparsity achieved by BASS-Tucker(d).

CHAPTER 4

STREAMING BAYESIAN DEEP TENSOR FACTORIZATION

Despite the success of existing tensor factorization methods, 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.

4.1 Introduction

Tensor factorization is a fundamental tool for multiway data analysis. While many tensor factorization methods have been developed [91, 32, 16, 42, 15], most of them conduct a mutilinear decomposition and are incapable of capturing complex, nonlinear relationships in data. Deep neural networks (NNs) are a class of very flexible and powerful modeling framework, known to be able to estimate all kinds of complicated (*e.g.*, highly nonlinear) mappings. The most recent work [50, 51] have attempted to incorporate NNs into tensor factorization and shown a promotion of the performance, in spite of the risk of overfitting

the tensor data that are typically sparse.

Nonetheless, one critical bottleneck for NN based factorization is the lack of effective approaches for streaming data. In practice, many applications produce huge volumes of data at a fast pace [21]. It is extremely costly to run the factorization from scratch every time when we receive a new set of entries. Some privacy-demanding applications (*e.g.,* SnapChat) even forbid us from revisiting the previously seen data. Hence, given new data, we need an effective and efficient way to incrementally update the model.

A general and popular approach is streaming variational Bayes (SVB) [10], which integrates the current posterior with the new data, and then estimates a variational approximation as the updated posterior. Although SVB has been successfully used to develop the state-of-the-art multilinear streaming tensor factorization [21], it does not perform well for deep NN based factorization. Due to the nested nonlinear coupling of the latent factors and NN weights, the variational model evidence lower bound (ELBO) that SVB maximizes is analytically intractable and we have to seek for stochastic optimization, which is unstable and hard to diagnose the convergence. We cannot use common tricks, *e.g.,* cross-validation and early stopping, to alleviate the issue, because we cannot store or revisit the data in the streaming scenario. Consequently, the posterior updates are often unreliable and inferior, which in turn hurt the subsequent updates and tend to result in a poor final model estimation.

To address these issues, we propose SBDT, a streaming Bayesian deep tensor factorization method that not only exploits NNs' expressive power to capture intricate relationships, but also provides efficient, high-quality posterior updates for streaming data. Specifically, we first use Bayesian neural networks to build a deep tensor factorization model, where the input is the concatenation of the factors associated with each tensor entry and the NN output predicts the entry value. To reduce the risk of overfitting, we place a spike-and-slab prior over each NN weight to encourage sparsity. For streaming inference, we use the multivariate delta method [6] that employs a Taylor expansion of the NN output to analytically compute its moments, and match the moments to obtain the its current posterior and the running model evidence. We then use back-propagation to calculate the gradient of the log evidence, with which we match the moments and update the posterior of the latent factors and NN weights in the assumed-density-filtering [9] framework. After

processing all the newly received entries, we update the spike-and-slab prior approximation with expectation propagation [55] to identify and inhibit redundant/useless weights. In this way, the incremental posterior updates are deterministic, reliable and efficient.

For evaluation, we examined SBDT in four real-world large-scale applications, including both binary and continuous tensors. We compared with the state-of-the-art streaming tensor factorization algorithm [21] based on a multilinear form, and streaming nonlinear factorization methods implemented with SVB. In both running and final predictive performance, our method consistently outperforms the competing approaches, mostly by a large margin. The running accuracy of SBDT is also much more stable and smooth than the SVB based methods.

4.2 Background and Notations

4.2.1 Streaming Variational Bayes

A general and popular framework for incremental model estimation is streaming variational Bayes(SVB) [10], which is grounded on the incremental version of Bayes' rule,

$$p(\boldsymbol{\theta}|\mathcal{D}_{\text{old}} \cup \mathcal{D}_{\text{new}}) \propto p(\boldsymbol{\theta}|\mathcal{D}_{\text{old}})p(\mathcal{D}_{\text{new}}|\boldsymbol{\theta}), \tag{4.1}$$

where $\boldsymbol{\theta}$ are the latent random variables in the probabilistic model we are interested in, $\mathcal{D}_{\text{old}}$ all the data that have been seen so far, and $\mathcal{D}_{\text{new}}$ the incoming data. SVB approximates the current posterior $p(\boldsymbol{\theta}|\mathcal{D}_{\text{old}})$ with a variational posterior $q_{\text{cur}}(\boldsymbol{\theta})$. When the new data arrives, SVB integrates $q_{\text{cur}}(\boldsymbol{\theta})$ with the likelihood of the new data to obtain an unnormalized, blending distribution,

$$\tilde{p}(\boldsymbol{\theta}) = q_{\text{cur}}(\boldsymbol{\theta})p(\mathcal{D}_{\text{new}}|\boldsymbol{\theta}), \tag{4.2}$$

which can be viewed as approximately proportional to the joint distribution $p(\boldsymbol{\theta}, \mathcal{D}_{\text{old}} \cup \mathcal{D}_{\text{new}})$. To conduct the incremental update, SVB uses $\tilde{p}(\boldsymbol{\theta})$ to construct a variational ELBO [92], $\mathcal{L}(q(\boldsymbol{\theta})) = \mathbb{E}_q[\log(\tilde{p}(\boldsymbol{\theta})/q(\boldsymbol{\theta}))]$, and maximizes the ELBO to obtain the updated posterior, $q^* = \text{argmax}_q \mathcal{L}(q)$. This is equivalent to minimizing the Kullback-Leibler (KL) divergence between q and the normalized $\tilde{p}(\boldsymbol{\theta})$. We then set $q_{\text{cur}} = q^*$ and prepare the update for the next batch of new data. At the beginning (when we do not receive any data), we set $q_{\text{cur}} = p(\boldsymbol{\theta})$, the original prior in the model. For efficiency and convenience, a

factorized variational posterior $q(\boldsymbol{\theta}) = \prod_j q(\theta_j)$ is usually adopted to fulfill cyclic, closed-form updates. For example, the state-of-the-art streaming tensor factorization, POST [21], uses the CP form to build a Bayesian model, and applies SVB to update the posterior of the factors incrementally when receiving new tensor entries.

4.3 Bayesian Deep Tensor Factorization

Despite the elegance and convenience of the popular Tucker and CP factorization, their multilinear form can severely limit the capability of estimating complicated, highly nonlinear/nonstationary relationships hidden in data. While numerous other methods have also been proposed, *e.g.,* [16, 42, 15], most are still inherently based on the CP or Tucker form. Enlightened by the expressive power of (deep) neural networks [30], we propose a Bayesian deep tensor factorization model to overcome the limitation of traditional methods and flexibly estimate all kinds of complex relationships.

Specifically, for each tensor entry $\mathbf{i}$, we construct an input $\mathbf{x}_\mathbf{i}$ by concatenating all the latent factors associated with $\mathbf{i}$, namely, $\mathbf{x}_\mathbf{i} = [\left(\mathbf{u}^1_{i_1}\right)^\top, \ldots, \left(\mathbf{u}^K_{i_K}\right)^\top]^\top$. We assume that there is an unknown mapping between the input factors $\mathbf{x}_\mathbf{i}$ 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 $\mathbf{h}_m$ ($1 \leq m \leq M-1$) and define $\mathbf{h}_0 = \mathbf{x}_\mathbf{i}$. 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 $f_\mathcal{W}(\mathbf{x}_\mathbf{i}) = \mathbf{W}_M[\mathbf{h}_{M-1}; 1]/\sqrt{V_{M-1}+1}$. Given the output, we sample the observed entry value $y_\mathbf{i}$ via a noisy model. For continuous data, we use a Gaussian noise model, $p(y_\mathbf{i}|\mathcal{U}) = \mathcal{N}\left(y_\mathbf{i}|f_\mathcal{W}(\mathbf{x}_\mathbf{i}), \tau^{-1}\right)$ where τ is the inverse noise variance. We further assign τ a Gamma prior, $p(\tau) = \text{Gamma}(\tau|a_0, b_0)$. For binary data, we use the Probit model, $p(y_\mathbf{i}|\mathcal{U}) = \Phi\big((2y_\mathbf{i} - 1)f_\mathcal{W}(\mathbf{x}_\mathbf{i})\big)$ where $\Phi(\cdot)$ is the cumulative density function (CDF) of the standard normal distribution.

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 [40, 89] (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 $p(s_{mij}|\rho_0) = \text{Bern}(s_{mjt}|\rho_0) = \rho_0^{s_{mjt}}(1-\rho_0)^{1-s_{mjt}}$. The weight is then sampled from:

$$p(w_{mjt}|s_{mjt}) = s_{mjt}\mathcal{N}(w_{mjt}|0,\sigma_0^2) + (1-s_{mjt})\delta(w_{mjt}), \tag{4.3}$$

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

$$\begin{aligned} &p(\mathcal{U},\mathcal{W},\mathcal{S},\tau) = \\ &\textstyle\prod_{m=1}^{M}\prod_{j=1}^{V_m}\prod_{t=1}^{V_{m-1}+1}\text{Bern}(s_{mjt}|\rho_0)\big(s_{mjt}\mathcal{N}(w_{mjt}|0,\sigma_0^2) + (1-s_{mjt})\delta(w_{mjt})\big) \\ &\cdot\textstyle\prod_{k=1}^{K}\prod_{j=1}^{d_k}\mathcal{N}(\mathbf{u}_j^k|\mathbf{0},\mathbf{I})\text{Gamma}(\tau|a_0,b_0)\prod_{n=1}^{N}\mathcal{N}(y_{\mathbf{i}_n}|f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n}),\tau^{-1}), \end{aligned} \tag{4.4}$$

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

$$\begin{aligned} &p(\mathcal{U},\mathcal{W},\mathcal{S}) = \\ &\textstyle\prod_{m=1}^{M}\prod_{j=1}^{V_m}\prod_{t=1}^{V_{m-1}+1}\text{Bern}(s_{mjt}|\rho_0)\big(s_{mjt}\mathcal{N}(w_{mjt}|0,\sigma_0^2) + (1-s_{mjt})\delta(w_{mjt})\big) \\ &\cdot\textstyle\prod_{k=1}^{K}\prod_{j=1}^{d_k}\mathcal{N}(\mathbf{u}_j^k|\mathbf{0},\mathbf{I})\prod_{n=1}^{N}\Phi\big((2y_{\mathbf{i}_n}-1)f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})\big). \end{aligned} \tag{4.5}$$

4.4 Streaming Posterior Inference

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 $\{\mathcal{B}_j\}_{j<t}$. While we can apply SVB,

the variational ELBO that integrates the current posterior and the new entry batch will be analytically intractable. Take binary tensors as an example. Given a new entry batch $\mathcal{B}_t$, the EBLO constructed based on the blending distribution (see (4.2)) is:

$$\mathcal{L} = -\mathrm{KL}\big(q(\mathcal{U},\mathcal{S},\mathcal{W})\|q_{\mathrm{cur}}(\mathcal{U},\mathcal{S},\mathcal{W})\big) + \sum_{n\in\mathcal{B}_t} \mathbb{E}_q\big[\log\Phi\big((2y_{\mathbf{i}_n}-1)f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})\big)\big]. \tag{4.6}$$

Due to the nested, nonlinear coupling of the factors (in each $\mathbf{x}_{\mathbf{i}_n}$) and NN weights $\mathcal{W}$ in calculating $f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})$, the expectation terms in $\mathcal{L}$ are intractable, without any closed form. Obviously, the same conclusion applies to the continuous data. Therefore, to maximize $\mathcal{L}$ so as to obtain the updated posterior, we have to use stochastic gradient descent (SGD), typically with the re-parameterization trick [44]. However, without the explicit form of $\mathcal{L}$, it is hard to diagnose the convergence of SGD — it may stop at a place far from the (local) optimums. Note that we cannot use hold-out datasets or cross-validation to monitor/control the training because we cannot store or revisit the data. The inferior posterior estimation in one batch can in turn influence the posterior updates in the subsequent batches, and finally result in a very poor model estimation.

4.4.1 Online Moment Matching for Posterior Update

To address these problems, we exploit the assumed-density-filtering (ADF) framework [9], which can be viewed as an online version of expectation propagation (EP) [55], a general approximate Bayesian inference algorithm. ADF is also based on the incremental version of Bayes' rule (see (4.1)). It uses a distribution in the exponential family [92] to approximate the current posterior. When the new data arrive, instead of maximizing a variational ELBO, ADF projects the (unnormalized) blending distribution (4.2) 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

$$q_{\text{cur}}(\mathcal{W},\mathcal{U},\mathcal{S}) = \prod_{m=1}^{M}\prod_{j=1}^{V_m}\prod_{t=1}^{V_{m-1}+1} \text{Bern}(s_{mjt}|\rho_{mjt})\mathcal{N}(w_{mjt}|\mu_{mjt},v_{mjt}) \cdot \prod_{k=1}^{K}\prod_{j=1}^{d_k}\prod_{t=1}^{r_k}\mathcal{N}(u_{jt}^k|\psi_{kjt},\nu_{kjt}). \tag{4.7}$$

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,

$$Z_n = \int q_{\text{cur}}(\mathcal{W},\mathcal{U},\mathcal{S})\Phi\big((2y_{\mathbf{i}_n}-1)f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})\big)\text{d}\mathcal{W}\text{d}\mathcal{U}\text{d}\mathcal{S}. \tag{4.8}$$

Under the Gaussian form, according to [56], 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

$$\mu^* = \mu + v\frac{\partial \log Z_n}{\partial \mu}, \tag{4.9}$$

$$v^* = v - v^2\big[\big(\frac{\partial \log Z_n}{\partial \mu}\big)^2 - 2\frac{\partial \log Z_n}{\partial v}\big], \tag{4.10}$$

where μ and v are the current posterior mean and variance of the corresponding weight or factor. 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 [59, 6] that expands the NN output at the mean of $\mathcal{W}$ and $\mathcal{U}$ with a Taylor approximation,

$$f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n}) \approx f_{\mathbb{E}[\mathcal{W}]}(\mathbb{E}[\mathbf{x}_{\mathbf{i}_n}]) + \mathbf{g}_n^\top(\boldsymbol{\eta}_n - \mathbb{E}[\boldsymbol{\eta}_n]), \tag{4.11}$$

where the expectation is under $q_{\text{cur}}(\cdot)$, $\boldsymbol{\eta}_n = \text{vec}(\mathcal{W}\cup\mathbf{x}_{\mathbf{i}_n})$, $\mathbf{g}_n = \nabla f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})|_{\boldsymbol{\eta}_n=\mathbb{E}[\boldsymbol{\eta}_n]}$. Note that $\mathbf{x}_{\mathbf{i}_n}$ is the concatenation of the latent factors associated with $\mathbf{i}_n$. The rationale of the approximation (4.11) 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 (see (4.4) and (4.5)). Therefore, we can ignore the second-order term that involves the posterior variances. Note that despite the seemingly simpler structure, our approximation in (4.11) 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 (4.11), we can calculate the first and second moments of $f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})$,

$$\alpha_n = \mathbb{E}_{q_{\text{cur}}}[f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})] \approx f_{\mathbb{E}[\mathcal{W}]}(\mathbb{E}[\mathbf{x}_{\mathbf{i}_n}]), \tag{4.12}$$

$$\beta_n = \text{Var}_{q_{\text{cur}}}(f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})) \approx \mathbf{g}_n^\top \text{diag}(\gamma_n)\mathbf{g}_n, \tag{4.13}$$

where each $[\gamma_n]_j = \text{Var}_{q_{\text{cur}}}([\boldsymbol{\eta}_n]_j)$. Due to the fully factorized posterior form, we have $\text{cov}(\boldsymbol{\eta}_n) = \text{diag}(\boldsymbol{\eta}_n)$. 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 (4.8) by

$$\begin{aligned} Z_n &= \mathbb{E}_{q_{\text{cur}}(\mathcal{W},\mathcal{U},\mathcal{S})}[\Phi\big((2y_{\mathbf{i}_n} - 1)f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})\big)] = \mathbb{E}_{q_{\text{cur}}(f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n}))}[\Phi\big((2y_{\mathbf{i}_n} - 1)f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})\big)] \\ &\approx \int \mathcal{N}(f_o|\alpha_n, \beta_n)\Phi\big((2y_{\mathbf{i}_n} - 1)f_o\big)\mathrm{d}f_o = \Phi\big(\frac{(2y_{\mathbf{i}_n} - 1)\alpha_n}{\sqrt{1+\beta_n}}\big), \end{aligned} \tag{4.14}$$

where we redefine $f_o = f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n})$ for simplicity. With the nice analytical form, we can immediately apply (4.10) 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 (4.11) and (4.13) to obtain the posterior of the NN output, we derive the running model evidence by

$$\begin{aligned} Z_n &= \mathbb{E}_{q_{\text{cur}}(\mathcal{W},\mathcal{U},\mathcal{S},\tau)}[\mathcal{N}\big(y_{\mathbf{i}_n}|f_{\mathcal{W}}(\mathbf{x}_{\mathbf{i}_n}), \tau^{-1}\big)] = \mathbb{E}_{q_{\text{cur}}(f_o)q_{\text{cur}}(\tau)}[\mathcal{N}(y_{\mathbf{i}_n}|f_o, \tau^{-1})] \\ &\approx \mathbb{E}_{q_{\text{cur}}(\tau)}\big[\int \mathcal{N}(f_o|\alpha_n, \beta_n)\mathcal{N}(y_{\mathbf{i}_n}|f_o, \tau^{-1})\mathrm{d}f_o\big] = \mathbb{E}_{q_{\text{cur}}(\tau)}[\mathcal{N}(y_{\mathbf{i}_n}|\alpha_n, \beta_n + \tau^{-1})]. \end{aligned} \tag{4.15}$$

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

$$\mathcal{N}(y_{\mathbf{i}_n}|\alpha_n, \beta_n + \tau^{-1}) \approx$$
$$\mathcal{N}(y_{\mathbf{i}_n}|\alpha_n, \beta_n + \mathbb{E}_{q_{\text{cur}}}[\tau]^{-1}) + (\tau - \mathbb{E}_{q_{\text{cur}}}[\tau])\partial\mathcal{N}(y_{\mathbf{i}_n}|\alpha_n, \beta_n + \tau^{-1})/\partial\tau|_{\tau=\mathbb{E}_{q_{\text{cur}}}[\tau]}. \quad (4.16)$$

Taking expectation over the Taylor expansion gives

$$Z_n \approx \mathcal{N}(y_{\mathbf{i}_n}|\alpha_n, \beta_n + \mathbb{E}_{q_{\text{cur}}}(\tau)^{-1}) = \mathcal{N}(y_{\mathbf{i}_n}|\alpha_n, \beta_n + b/a). \quad (4.17)$$

We now can use (4.10) 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:

$$\tilde{p}(f_o, \tau) \propto q_{\text{cur}}(f_o)q_{\text{cur}}(\tau)\mathcal{N}(y_{\mathbf{i}_n}|f_o, \tau^{-1}) = \mathcal{N}(f_o|\alpha_n, \beta_n)\text{Gamma}(\tau|a, b)\mathcal{N}(y_{\mathbf{i}_n}|f_o, \tau^{-1}). \quad (4.18)$$

Then we follow CEP [96] to first derive the conditional moments and then approximate the expectation of the conditional moments to obtain the moments. The updated posterior is given by $q^*(\tau) = \text{Gamma}(\tau|a^*, b^*)$, where $a^* = a + \frac{1}{2}$ and $b^* = b + \frac{1}{2}((y_{\mathbf{i}_n} - \alpha_n)^2 + \beta_n)$.

4.4.2 Prior Approximation Refinement

Ideally, at the beginning (*i.e.,* when we have not received any data), we should set q_{cur} to the prior of the model (see (4.4) and (4.5)). This is feasible for the factors $\mathcal{U}$, selection indicators $\mathcal{S}$ and the inverse noise variance τ (for continuous data only), because their Gaussian, Bernoulli and Gamma priors are all members of the exponential family. However, the spike-and-slab prior for each NN weight w_{mjt} (see (4.3)) is a mixture prior and does not belong to the exponential family. Hence, we introduce an approximation term,

$$p(w_{mjt}|s_{mjt}) \overset{\propto}{\sim} A(w_{mjt}, s_{mjt}) = \text{Bern}\big(s_{mjt}|c(\rho_{mjt})\big)\mathcal{N}(w_{mjt}|\mu^0_{mjt}, v^0_{mjt}), \quad (4.19)$$

where $\overset{\propto}{\sim}$ 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 (4.10)), 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 (4.19), 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. We also summarize our streaming inference in Algorithm 2.

4.4.3 Algorithm Complexity

The time complexity of our streaming inference is $\mathcal{O}(NV + \sum_{k=1}^{K} d_k r_k)$ where V is the total number of weights in the NN. Therefore, the computational cost is proportional to N, the size of the streaming batch. The space complexity is $\mathcal{O}(V + \sum_{k=1}^{K} d_k r_k)$, including the storage of the posterior for the factors $\mathcal{U}$, NN weights $\mathcal{W}$ and selection indicators $\mathcal{S}$, and the approximation term for the spike-and-slab prior.

4.5 Related Work

Classical CP [32] and Tucker [91] factorization are multilinear and therefore are incapable of estimating complex, nonlinear relationships in data. While numerous other

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 (4.14) (binary data) or (4.17) (continuous data).
        Update the posterior for $\mathcal{W}$ and the associated factors $\{u^k_{i_{nj}}\}_{k,j}$ with (4.10).
        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 whilereturn the current posterior $q_{\text{cur}}(\cdot)$.

approaches have been proposed [82, 16, 85, 37, 42, 102, 15, 39], most of them are still based on the CP or Tucker form. To overcome these issues, recently, several Bayesian nonparametric tensor factorization models [101, 107, 106, 108] were proposed to estimate the nonlinear relationships with Gaussian processes [71]. The most recent work, NeuralCP [50] and CoSTCo [51] have shown the advantage of NNs in tensor factorization. CoSTCo also concatenates the factors associated with each entry to construct the input and use the NN output to predict the entry value; but to alleviate overfitting, CoSTCo introduces two convolutional layers to extract local features and then feed them into dense layers. By contrast, with the spike-and-slab prior and Bayesian inference, we found that our model can also effectively prevent overfitting, without the need for extra convolutional layers. NeuralCP uses two NNs, one to predict the entry value, the other the (log) noise variance. Hence, NeuralCP only applies to continuous data. Our model can be used for both continuous and binary data. Finally, both CoSTCo and NerualCP are trained with stochastic optimization, need to pass the data many times (epochs), and hence cannot handle streaming data. [86] inserted NNs into kernels to conduct Gaussian process based factorization. Apart from this, there are interesting works using tensor factorization to compress NNs [103, 3], which, however, are off our topic.

Expectation propagation [55] is an approximate Bayesian inference algorithm that generalizes assumed-density-filtering (ADF) [9] and (loopy) belief propagation [57]. EP employs an exponential-family term to approximate the prior and likelihood of each data point, and cyclically updates each approximation term via moment matching. ADF can be considered as applying EP in a model including only one data point. Because ADF only maintains the holistic posterior, without the need for keeping individual approximation terms, it is very appropriate for streaming learning. EP can meet a practical barrier when the moment matching is intractable. To address this problem, CEP [96] proposed conditional EP that uses conditional moment matching, quadrature and Taylor approximations to provide a high-quality, analytical solution. Based on EP and ADF, [36] proposed probabilistic back-propagation, a batch inference algorithm for Bayesian neural networks. PBP conducts ADF to pass the dataset many times and re-update the prior approximation after each pass. A key difference from our work is that PBP conducts a forward, layer by layer moment matching, to approximate the posterior of each hidden neuron in the network,

until it reaches the output. The computation is limited to fully connected, feed-forward networks and ReLU activation function. By contrast, our method computes the moments of the NN output via the delta method (*i.e.,* Taylor expansions) and does not need to approximate the posterior of the hidden neurons. Therefore, our method is free to use any NN architecture and activation function. Note that the multi-variate delta method was also used in Laplace's approximation [53] for NNs and non-conjugate variational inference [93]. Furthermore, we employ spike-and-slab priors over the NN weights to control the complexity of the model and to prevent overfitting in the streaming inference.

4.6 Experiment

4.6.1 Predictive Performance

4.6.1.1 Datasets

We examined SBDT on four real-world, large-scale datasets. (1) *DBLP* [21], 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* [21], 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.

4.6.1.2 Competing Methods

We compared with the following baselines. (1) POST [21], the state-of-the-art probabilistic streaming tensor decomposition algorithm based on the CP model. It uses streaming variational Bayes (SVB) [10] 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 [108], 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 [11]. It uses SGD to optimize another intractable ELBO. (5) CP-WOPT [1], a scalable static CP

factorization algorithm implemented with gradient-based optimization.

4.6.1.3 Parameter Settings

We implemented our method, SBDT with Theano, and SVB-DTF, SVB/SS-GPTF with TensorFlow. For POST, we used the original MATLAB implementation (https://github.com/yishuaidu/POST). For SVB/SS-GPTF, we set the number of pseudo inputs to 128 in their sparse approximations. We used Adam [43] for the stochastic optimization in SVB-DTF and SVB/SS-GPTF, where we set the number of epochs to 100 in processing each streaming batch and tuned the learning rate from $\{10^{-5}, 5\times10^{-5}, 10^{-4}, 3\times10^{-4}, 5\times10^{-4}, 10^{-3}, 3\times10^{-3}, 5\times10^{-3}, 10^{-2}\}$. For SBDT and SVB-DTF, We used a 3-layer NN, with 50 nodes in each hidden layer. We tested ReLU and tanh activations.

4.6.1.4 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 [21], including 320K 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.

We conducted two groups of evaluations. In the first group, we fixed the batch size to 256 and examined the predictive performance with different numbers of the factors, {3, 5, 8 10}. In the second group, we fixed the factor number to 8, and examined how the prediction accuracy varies with the size of the streaming batches, $\{2^6, 2^7, 2^8, 2^9\}$. Obviously, the more the factors/larger the batch size, the more expensive for SBDT to factorize in

each streaming batch. *Hence both settings examined the trade-off between the accuracy and computational complexity.* The results are reported in Fig. 4.1. 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. First, SBDT significantly improves upon POST and CP-WOPT — the streaming and static multilinear factorization, confirming the advantages of the deep tensor factorization. It worth noting that CP-WOPT performed much worse than POST on *ACC* and *MovieLen1M*, which might be due to the poor local optimums CP-WOPT converged to. Second, SVB/SS-GPTF are generally far worse than our method, and in many cases even inferior to POST (see Fig. 4.1b, d, f and h). SVB-DTF is even worse. Only on Fig. 4.1c, the performance of SVB-DTF is comparable to or better than CP-WOPT, and in all the other cases, SVB-DTF is much worse than all the other methods and we did not show its curve in the figure (similar for CP-WOPT in Fig. 4.1g). Those results might be due to the inferior/unreliable stochastic posterior updates. Lastly, although both SBDT-ReLU and SBDT-tanh outperform all the baselines, SBDT-ReLU is overall better than SBDT-tanh (especially Fig. 4.1g).

4.6.2 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. 4.2. 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. 4.2b, SVB-DTF has running AUC steadily around 0.5, implying that SVB actually failed to effectively update the posterior.

4.6.3 Efficiency

We implemented SBDT by Theano and SVB-DTF, SVB-GPTF, SS-GPTF by TensorFlow. POST was implemented with Matlab. We ran all the methods on a desktop machine with Intel i9-9900K CPU and 32GB memory. We did not use GPU acceleration to run SBDT for a fair comparison. SBDT is faster than POST on *MovieLen1M* but slower on the other datasets. For example, for $r = 8$ and batch-size 128, the running time (in seconds) are {SBDT-ReLU: 6,928, SBDT-tanh: 8,566, POST: 40,551} on *MovieLen1M*, {SBDT-ReLU: 13,742, SBDT-tanh: 15,641, POST: 1,442} on *ACC*, {SBDT-ReLU: 5,255, SBDT-tanh: 5,356, POST: 2,459} on *Anime* and {SBDT-ReLU: 1,349, SBDT-tanh: 1,481, POST: 371} on *DBLP*. Note that on different datasets, POST may need a different number of iterations to converge in the mean-field variational updates. The tolerance level was set to 10^{-5} and the maximum number of iterations 500. The results are reasonable, because SBDT is based on neural networks, including much more parameters than CP factorization, and the computation is much more complex. The other methods are in general faster than SBDT. This might be partly due to the difference in efficiency between Theano and TensorFlow libraries. Nonetheless, the predictive performance of those methods are much worse than SBDT and even often worse than POST.

4.6.4 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. 4.3, we show the posterior selection probability $c(\rho^*_{mjt})$ versus the posterior mean μ^*_{mjt} for each weight w_{mjt}, and $c(\rho^*_{mjt})$ versus 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.

4.7 Conclusion

We have presented SBDT, a streaming probabilistic deep tensor factorization approach, which can effectively leverage neural networks to capture complicated relationships for streaming factorization. Experiments on real-world applications have demonstrated the advantage of SBDT.

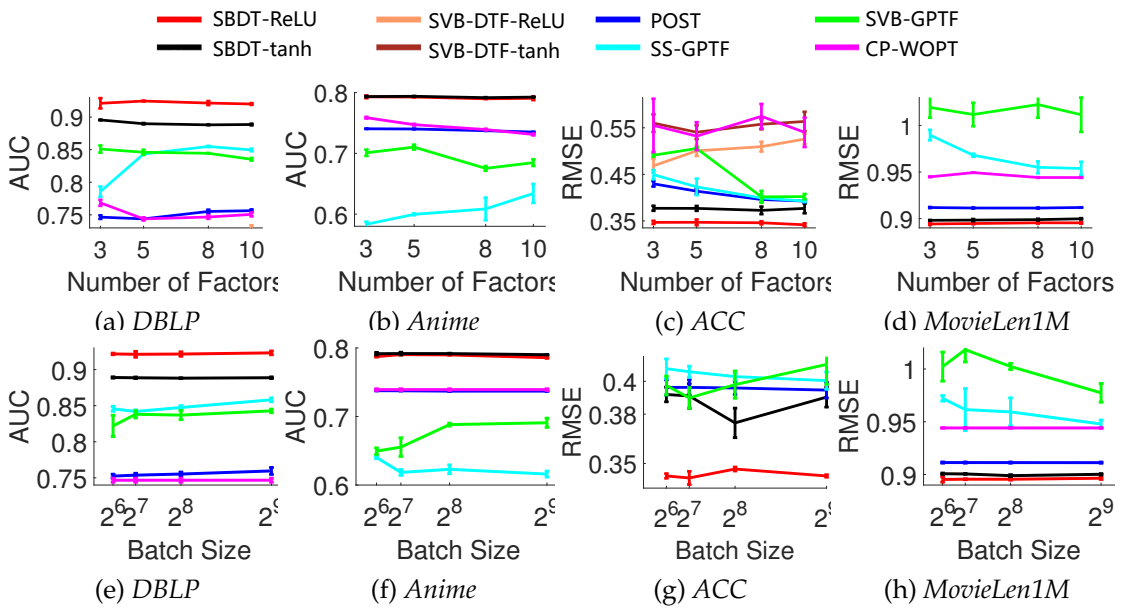


Figure 4.1: Predictive performance with different numbers of factors (top row) and streaming batch sizes (bottom row).

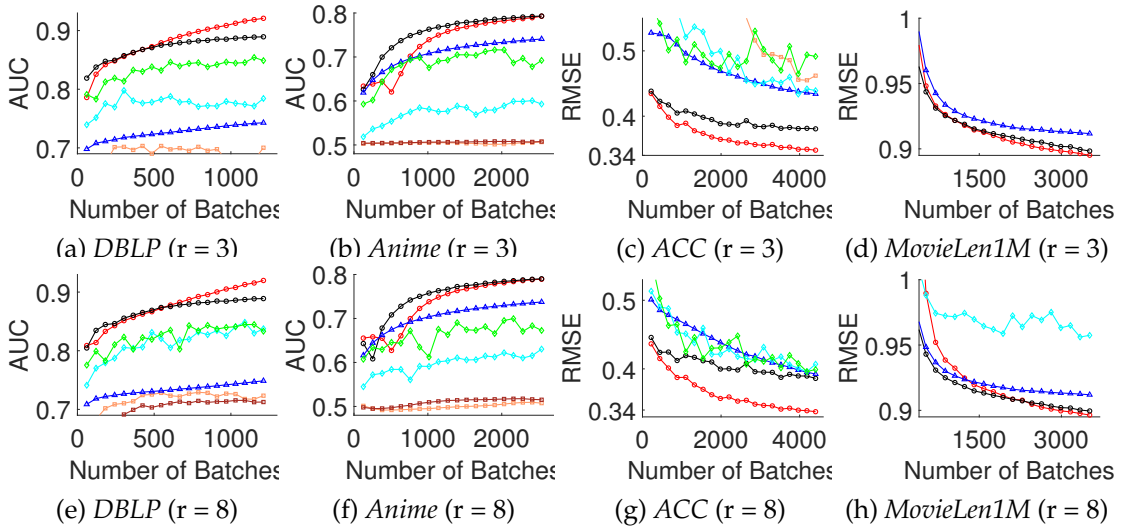


Figure 4.2: Running prediction accuracy along with the number of processed streaming batches, where the batch size was fixed to 256.

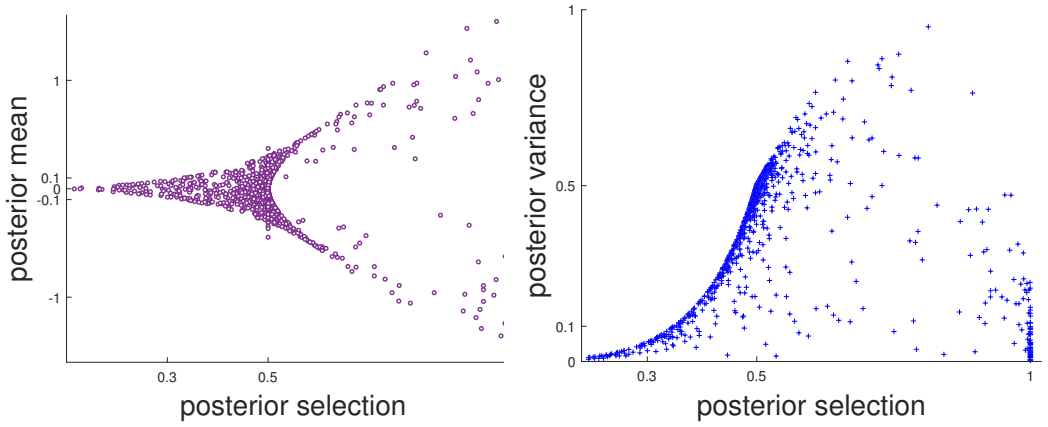


Figure 4.3: Posterior selection probability $c(\rho^*_{mjt})$ versus the posterior mean and variance of each NN weight w_{mjt}.

CHAPTER 5

BAYESIAN CONTINUOUS-TIME TUCKER DECOMPOSITION

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.

5.1 Introduction

Multiway interaction data is omnipresent in real-world applications, such as in online advertising, e-commerce and social networking. A popular and powerful framework for multiway interaction analysis and prediction is tensor decomposition, which aims to estimate a set of latent factors to represent the interaction nodes, and use the factors to reconstruct the observed tensor elements. The factors can reflect unknown patterns in data, such as communities across the nodes, and provide effective features to build downstream

predictive tools, such as product rating for recommendation and clicks for advertisement display.

While many successful tensor decomposition methods have been developed [91, 32, 16, 42, 15], these methods ignore or under-exploit the valuable time information, which often comes along with the tensor data, *e.g.,* at which time point a *user* purchased an *item* at a specific *Amazon store*. Current methods often throw out the timestamps or bin the timestamps into crude steps, *e.g.,* weeks or months, and augment the tensor with a time step mode [100, 73, 107, 106, 21]. While between the steps we can use conditional priors and/or nonlinear dynamics to model their transition, the temporal dependencies within each step are overlooked. The most recent work [104] although introduces continuous-time coefficients into the CANDECOMP/PARAFAC (CP) decomposition [32], its parametric modeling of the coefficients, *i.e.,* polynomial splines, might not be flexible enough to capture a variety of different temporal dynamics in data (*e.g.,* from simple linear to highly nonlinear).

To overcome these limitations, we propose BCTT, a novel continuous-time Bayesian dynamic decomposition model. We extend the classical Tucker decomposition, which accounts for every multiplicative interaction between the factors across different tensor modes and is highly interpretable and quite expressive. We model the tensor-core — weights of the factor interactions — as a time-varying function. We place a Gaussian process (GP) prior, a nonparametric function prior that can flexibly estimate all kinds of functions, not restricted to any specific parametric form. In this way, our model not only maintains the interpretability, but also can automatically capture different, complex temporal dynamics from data. For efficient and high-quality posterior inference, we construct a linear time-invariant (LTI) stochastic differential equation (SDE) [33] as an equivalent representation of the temporal GP. Based on the LTI-SDE, we build a state-space prior, which is essentially a Gaussian Markov chain but is equivalent to the GP prior. In this way, we circumvent the expensive kernel matrix computation in the original GP, and do not need any low-rank or sparse approximations. Next, we develop a message-passing posterior inference algorithm in the expectation propagation framework. We use Kalman filtering and Rauch–Tung–Striebel (RTS) smoothing [74] to efficiently compute the posterior of the SDE states, and use conditional moment matching [96] and multi-variate delta method [6]

to overcome the intractability in moment matching. Both the time and space complexity of our inference algorithm is linear in the number of observed data points.

For evaluation, we examined our approach in both ablation study and real-world applications. On synthetic datasets, BCTT successful learned different temporal dynamics and recovered the clustering structures of the tensor nodes from their factor estimation. On three real-world temporal tensor datasets, BCTT significantly outperforms the competing dynamic decomposition methods, including discrete time factors and continuous time coefficients, often by a large margin. The structure of the learned tensor-core also shows interesting temporal evolution.

5.2 Background and Notation

5.2.1 Tucker Decomposition

Consider a K-mode tensor $\mathcal{Y} \in \mathbb{R}^{d_1 \times \cdots \times d_K}$, where d_k is the number of nodes in mode k. We use a K-elements tuple $\mathbf{i} = (i_1, \ldots, i_K)$ to index each entry of the tensor, and denote the entry value by $y_{\mathbf{i}}$. To factorize $\mathcal{Y}$ into a concise structure, we introduce a set of latent factors for the tensor nodes, $\mathcal{U} = \{\mathbf{U}^1, \ldots, \mathbf{U}^K\}$, where each $\mathbf{U}^k = [\mathbf{u}^k_1, \ldots, \mathbf{u}^k_{d_k}]^\top$ is a factor matrix, in which each row consists of the factors for a node j in mode k, namely $\mathbf{u}^k_j$ $(1 \leq j \leq d_k)$. Given the factorization form, we estimate the optimal factors $\mathcal{U}$ to reconstruct the tensor $\mathcal{Y}$, by minimizing a loss on the observed entries.

Tucker decomposition [91] is an interpretable and expressive decomposition method that considers all the possible interactions between the factors across the tensor modes. Specifically, Tucker decomposition assumes $\mathcal{Y} \approx \mathcal{W} \times_1 \mathbf{U}^1 \times_2 \ldots \times_K \mathbf{U}^K$ where $\mathcal{W} \in \mathbb{R}^{R_1 \times \ldots \times R_K}$ is parametric tensor-core and $\times_k$ is mode k tensor-matrix product [45]. The entry-wise form is therefore given by

$$y_{\mathbf{i}} \approx \text{vec}(\mathcal{W})^\top \left(\mathbf{u}^1_{i_1} \otimes \ldots \otimes \mathbf{u}^K_{i_K}\right) = \sum_{r_1=1}^{R_1} \ldots \sum_{r_K=1}^{R_K} \left[w_{(r_1,\ldots,r_K)} \cdot \prod_{k=1}^{K} u^k_{i_k,r_k} \right], \tag{5.1}$$

where $\text{vec}(\cdot)$ is the vectorization and $\otimes$ is the Kronecker product. As we can see from (5.1), every interaction between the factors across the K modes is accounted for, $\{\prod_{k=1}^{K} u^k_{i_k,r_k} | 1 \leq r_1 \leq R_1, \ldots, 1 \leq r_K \leq R_K\}$. Each interaction is weighted by an element of the tensor-core. It is easy to see that CP is a special case of Tucker decomposition when we set all $R_k = R$ and $\mathcal{W}$ to be diagonal.

5.2.2 Gaussian Processes (GPs) and SDE Representation

GPs are powerful Bayesian function estimators. Due to the nonparametric nature, GPs can automatically grasp the complexity of the target function underlying the data (*e.g.*, from linear to highly linear), not restricted to any parametric form. Specifically, suppose given N training examples, $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top$ and $\mathbf{y} = (y_1, \ldots, y_N)^\top$, we want to learn a function $f : \mathbb{R}^d \to \mathbb{R}$. We place a GP prior over the target function, and then any finite set of the function values follow a multivariate Gaussian distribution. Consider $\mathbf{f} = (f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N))^\top$ and we have $p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{m}, \mathbf{K})$, where $\mathbf{m}$ is the mean function value at the inputs and usually set to $\mathbf{0}$, and $\mathbf{K}$ is an $N \times N$ kernel matrix — each element $[\mathbf{K}]_{n,n'} = \kappa(\mathbf{x}_n, \mathbf{x}_{n'})$ and $\kappa(\cdot, \cdot)$ is a kernel function. A commonly used, powerful kernel is Matérn kernel,

$$k(\mathbf{x}_n, \mathbf{x}_{n'}) = \sigma^2 \frac{\left(\frac{\sqrt{2\nu}}{l}\alpha(\mathbf{x}_n, \mathbf{x}_{n'})\right)^\nu}{\Gamma(\nu)2^{\nu-1}} K_\nu \left(\frac{\sqrt{2\nu}}{l}\alpha(\mathbf{x}_n, \mathbf{x}_{n'})\right), \tag{5.2}$$

where $\alpha(\cdot, \cdot)$ is the distance function (usually the Euclidean distance), $\Gamma(\cdot)$ is the gamma function, $K_\nu(\cdot)$ is the modified Bessel function of the second kind, ν is the degree of freedom, l is length-scale and σ^2 magnitude. Given $\mathbf{f}$, we use a noise model $p(\mathbf{y}|\mathbf{f})$ to fit the observed function outputs, *e.g.*, $p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{y}|\mathbf{f}, \tau^{-1}\mathbf{I})$. We can then conduct Bayesian inference. The predictive distribution of the function value at a new input $\mathbf{x}^*$ is straightforward to obtain: since $[\mathbf{f}; f(\mathbf{x}^*)]$ follows a joint Gaussian distribution as well and $p(f(\mathbf{x}^*)|\mathbf{f})$ is a conditional Gaussian distribution.

In the literature of stochastic differential equations (SDEs) [75, 61], 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 [33]. Take the Matérn kernel 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

$$\frac{\mathrm{d}^{m+1} f(t)}{\mathrm{d}t^{m+1}} + a_m \frac{\mathrm{d}^m f(t)}{\mathrm{d}t^m} + \ldots + a_0 f(t) = \xi(t), \tag{5.3}$$

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

$$\frac{\mathrm{d}\mathbf{y}(t)}{\mathrm{d}t} = \mathbf{F}\mathbf{x}(t) + \mathbf{L}\xi(t), \tag{5.4}$$

where

$$\mathbf{F} = \begin{pmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & 0 & 1 \\ -a_0 & \ldots & -a_{m-1} & -a_m \end{pmatrix}, \quad \mathbf{L} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}. \tag{5.5}$$

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.

5.3 Model

While useful, existing tensor decomposition methods use discrete time steps, and hence can miss the temporal variations within each step. Although the latest work [104] employs continuous-time coefficients in the CP decomposition, it uses polynomial splines to model these coefficients and might not be sufficient to capture more complex dynamics. The CP form can further restrict its capability of capturing temporal interactions between the factors across different tensor modes. To overcome these limitations, we propose BCTT, a Bayesian continuous-time Tucker decomposition approach.

Specifically, we model each element of the tensor-core $\mathcal{W}$ in the Tucker decomposition (5.1) 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, $w_{\mathbf{r}}(t) \sim \mathcal{GP}(0, \kappa(t, t'))$ 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:

$$p(y_n|\mathcal{W}(t_n),\mathcal{U}) = \mathcal{N}\left(y_n|\mathrm{vec}\left(\mathcal{W}(t_n)\right)^\top \left(\mathbf{u}^1_{i_{n_1}} \otimes \ldots \otimes \mathbf{u}^K_{i_{n_K}}\right), \tau^{-1}\right), \tag{5.6}$$

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

$$p(\mathcal{U}, \{\mathbf{w}_\mathbf{r}\}_\mathbf{r}, \tau, \mathbf{y}) = p(\mathcal{U})p(\tau) \cdot \prod_{\mathbf{r}=(1,\ldots,1)}^{(R_1,\ldots,R_K)} \mathcal{N}(\mathbf{w}_\mathbf{r}|\mathbf{0}, \mathbf{K}_\mathbf{r}) \prod_{n=1}^N p(y_n|\mathcal{W}(t_n),\mathcal{U}). \tag{5.7}$$

However, a straightforward formulation as in (5.7) brings in severe computational challenges. The joint probability includes many multivariate Gaussian distributions. When the number of time points N is large, the calculation of each kernel matrix $\mathbf{K}_\mathbf{r}$ 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 [67], 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 (5.4), 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

$$\frac{\mathrm{d}\boldsymbol{\gamma}_\mathbf{r}(t)}{\mathrm{d}t} = \mathbf{F}\boldsymbol{\gamma}_\mathbf{r} + \mathbf{L}\xi(t), \tag{5.8}$$

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,

$$p(\boldsymbol{\gamma}_\mathbf{r}(t_1)) = \mathcal{N}(\boldsymbol{\gamma}_\mathbf{r}(t_1)|\mathbf{0}, \mathbf{P}_\infty), \tag{5.9}$$

$$p(\boldsymbol{\gamma}_\mathbf{r}(t_{n+1})|\boldsymbol{\gamma}_\mathbf{r}(t_n)) = \mathcal{N}\left(\boldsymbol{\gamma}_\mathbf{r}(t_{n+1})|\mathbf{A}_n\boldsymbol{\gamma}_\mathbf{r}(t_n), \mathbf{Q}_n\right) \tag{5.10}$$

where $\mathbf{P}_\infty = [\sigma^2, 0; 0, \beta^2\sigma^2]$ is the stationary covariance calculated by solving the matrix Riccati equation [46], $\Delta_n = t_{n+1} - t_n$ 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 $\overline{\gamma}(t)$ as the concatenation of all $\{\gamma_{\mathbf{r}}(t)\}_{\mathbf{r}}$. Accordingly, the discrete form of the SDE for $\overline{\gamma}(t)$ follows

$$p(\overline{\gamma}_1) = \mathcal{N}(\overline{\gamma}_1|\mathbf{0}, \mathbf{\Sigma}), \tag{5.11}$$

$$p(\overline{\gamma}_{n+1}|\overline{\gamma}_n) = \mathcal{N}(\overline{\gamma}_{n+1}|\mathbf{B}_n\overline{\gamma}_n, \mathbf{C}_n), \tag{5.12}$$

where $\overline{\gamma}_n \overset{\Delta}{=} \overline{\gamma}(t_n)$, $\mathbf{\Sigma} = \text{diag}(\mathbf{P}_\infty, \ldots, \mathbf{P}_\infty)$, $\mathbf{B}_n = \text{diag}(\mathbf{A}_n, \ldots, \mathbf{A}_n)$, and $\mathbf{C}_n = \text{diag}(\mathbf{Q}_n, \ldots, \mathbf{Q}_n)$. As we can see, this is essentially a state-space prior over the collection of states $\{\overline{\gamma}_n\}$. 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 $\text{vec}(\mathcal{W}(t)) = \mathbf{H} \cdot \overline{\gamma}(t)$, 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 (5.7) by the state space prior in (5.12), and write the joint probability as

$$p(\mathcal{U}, \{\overline{\gamma}_n\}, \tau, \mathbf{y}) = p(\mathcal{U})p(\tau) \cdot p(\overline{\gamma}_1) \prod_{n=1}^{N-1} p(\overline{\gamma}_{n+1}|\overline{\gamma}_n) \cdot \prod_{n=1}^{N} \mathcal{N}\left(y_n | \left(\mathbf{H}\overline{\gamma}_n\right)^\top \left(\mathbf{u}^1_{i_{n_1}} \otimes \ldots \otimes \mathbf{u}^K_{i_{n_K}}\right), \tau^{-1}\right). \tag{5.13}$$

Since each state $\overline{\gamma}_n$ is only dependent on its previous state $\overline{\gamma}_{n-1}$ (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.

5.4 Algorithm

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 (5.1)) couples the relevant latent factors $\{\mathbf{u}^1_{i_{n_1}}, \ldots \mathbf{u}^K_{i_{n_K}}\}$ and state $\overline{\gamma}(t_n)$. To address this issue, we introduce Gaussian-Gamma likelihood approximations, and based on Kalman filtering (KF) [41] and Rauch-Tung-Striebel (RTS) smoothing [72] we develop an efficient message-passing algorithm in the expectation propagation (EP) framework [55]. See Fig. 5.1.

5.4.1 Gaussian-Gamma Approximations for Efficient Filtering and Smoothing

Specifically, we approximate each data likelihood with

$$\begin{aligned}&\mathcal{N}\left(y_n|\left(\mathbf{H}\overline{\boldsymbol{\gamma}}_n\right)^\top\left(\mathbf{u}^1_{i_{n_1}}\otimes\ldots\otimes\mathbf{u}^K_{i_{n_K}}\right),\tau^{-1}\right)\approx\ell_n\\&\stackrel{\Delta}{=}Z_n\cdot\textstyle\prod_{k=1}^K\mathcal{N}\left(\mathbf{u}^k_{i_{n_k}}|\mathbf{m}^{k,n}_{i_{n_k}},\mathbf{V}^{k,n}_{i_{n_k}}\right)\cdot\mathrm{Gam}\left(\tau|b_n,c_n\right)\cdot\mathcal{N}\left(\mathbf{H}\overline{\boldsymbol{\gamma}}_n\mid\boldsymbol{\beta}_n,\mathbf{S}_n\right),\end{aligned}\tag{5.14}$$

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

$$\begin{aligned}&q(\mathcal{U},\{\overline{\boldsymbol{\gamma}}_n\},\tau)\propto\\&\textstyle\prod_{k=1}^K\prod_{j=1}^{d_k}\mathcal{N}(\mathbf{u}^k_j|\mathbf{0},\mathbf{I})\mathrm{Gam}(\tau|b_0,c_0)\prod_{n=1}^N\prod_{k=1}^K\mathcal{N}\left(\mathbf{u}^k_{i_{n_k}}|\mathbf{m}^{k,n}_{i_{n_k}},\mathbf{V}^{k,n}_{i_{n_k}}\right)\mathrm{Gam}\left(\tau|b_n,c_n\right)\\&\cdot p(\overline{\boldsymbol{\gamma}}_1)\mathcal{N}\left(\mathbf{H}\overline{\boldsymbol{\gamma}}_1\mid\boldsymbol{\beta}_1,\mathbf{S}_1\right)\prod_{n=1}^{N-1}p(\overline{\boldsymbol{\gamma}}_{n+1}|\overline{\boldsymbol{\gamma}}_n)\mathcal{N}\left(\mathbf{H}\overline{\boldsymbol{\gamma}}_n\mid\boldsymbol{\beta}_n,\mathbf{S}_n\right).\end{aligned}\tag{5.15}$$

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 with the closed forms:

$$q(\mathbf{u}^k_j)\propto\mathcal{N}(\mathbf{u}^k_j|\mathbf{0},\mathbf{I})\prod_{i_{n_k}=j}\mathcal{N}(\mathbf{u}^k_{i_{n_k}}|\mathbf{m}^{k,n}_{i_{n_k}},\mathbf{V}^{k,n}_{i_{n_k}}),\tag{5.16}$$

$$q(\tau)\propto\mathrm{Gam}(\tau|b_0,c_0)\prod_{n=1}^N\mathrm{Gam}(\tau|b_n,c_n).\tag{5.17}$$

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 (5.14) — 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 (5.15)). 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 [74], which can be unstable and more costly.

5.4.2 Message Passing with Conditional Moment Matching

To optimize the approximation terms in each ℓ_n (see (5.14)), 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 [8]. 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,

$$\begin{aligned} q^{\setminus n}(\boldsymbol{\Theta}_n) &\propto \frac{q(\overline{\boldsymbol{\gamma}}_n)q(\tau)\prod_{k=1}^K q(\mathbf{u}_{i_{n_k}}^k)}{\ell_n} \\ &= \mathcal{N}(\boldsymbol{\eta}_n|\boldsymbol{\beta}^{\setminus n}, \mathbf{S}^{\setminus n}) \cdot \prod_{k=1}^K \mathcal{N}\left(\mathbf{u}_{i_{n_k}}^k | \mathbf{m}_{i_{n_k}}^{k,\setminus n}, \mathbf{V}_{i_{n_k}}^{k,\setminus n}\right) \mathrm{Gam}\left(\tau | b^{\setminus n}, c^{\setminus n}\right), \end{aligned} \tag{5.18}$$

where $\boldsymbol{\eta}_n = \mathbf{H}\overline{\boldsymbol{\gamma}}_n = \mathrm{vec}\left(\mathcal{W}(t_n)\right)$, and $\boldsymbol{\Theta}_n = \{\boldsymbol{\eta}_n, \{\mathbf{u}_{i_{n_k}}^k\}_k, \tau\}$ 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,

$$\tilde{p}(\boldsymbol{\Theta}_n) \propto q^{\setminus n}(\boldsymbol{\Theta}_n) \cdot \mathcal{N}\left(y_n | \boldsymbol{\eta}_n^\top \left(\mathbf{u}_{i_{n_1}}^1 \otimes \ldots \otimes \mathbf{u}_{i_{n_K}}^K\right), \tau^{-1}\right). \tag{5.19}$$

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

$$q^*(\boldsymbol{\Theta}_n) = \mathcal{N}(\boldsymbol{\eta}_n|\boldsymbol{\beta}_n^*, \mathbf{S}_n^*) \cdot \prod_{k=1}^K \mathcal{N}\left(\mathbf{u}_{i_{n_k}}^k | \mathbf{m}_{i_{n_k}}^{k,*}, \mathbf{V}_{i_{n_k}}^{k,*}\right) \mathrm{Gam}\left(\tau | b_n^*, c_n^*\right), \tag{5.20}$$

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

$$\ell_n \leftarrow \frac{q^*(\boldsymbol{\Theta}_n)}{q^{\setminus n}(\boldsymbol{\Theta}_n)}. \tag{5.21}$$

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 (5.20) 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:

$$\boldsymbol{\beta}_n^* = \mathbb{E}_{\tilde{p}}\left[\boldsymbol{\eta}_n\right], \tag{5.22}$$

$$\mathbf{S}_n^* = \mathbb{E}_{\tilde{p}}\left[\boldsymbol{\eta}_n\boldsymbol{\eta}_n^\top\right] - \mathbb{E}_{\tilde{p}}\left[\boldsymbol{\eta}_n\right]\mathbb{E}_{\tilde{p}}\left[\boldsymbol{\eta}\right]^\top. \tag{5.23}$$

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 (5.19), we do not have a closed form of the moments. To address this problem, we use the idea of the conditional moment matching [96]. 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,

$$\mathbb{E}_{\tilde{p}}\left[\boldsymbol{\phi}(\boldsymbol{\eta}_n)\right] = \mathbb{E}_{\tilde{p}(\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n})}\left[\mathbb{E}_{\tilde{p}(\boldsymbol{\eta}_n|\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n})}\left[\boldsymbol{\phi}(\boldsymbol{\eta})|\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}\right]\right], \tag{5.24}$$

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,

$$\mathbb{E}\left[\boldsymbol{\eta}_n|\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}\right] = \boldsymbol{\Sigma}_n\left(\left(\mathbf{S}^{\setminus n}\right)^{-1}\boldsymbol{\beta}^{\setminus n} + \tau y_n \mathbf{v}_n\right), \tag{5.25}$$

$$\mathbb{E}\left[\boldsymbol{\eta}_n\boldsymbol{\eta}_n^\top|\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}\right] = \boldsymbol{\Sigma}_n + \mathbb{E}\left[\boldsymbol{\eta}_n|\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}\right]\mathbb{E}\left[\boldsymbol{\eta}_n|\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n}\right]^\top, \tag{5.26}$$

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,

$$\mathbb{E}_{\tilde{p}}[\boldsymbol{\phi}(\boldsymbol{\eta}_n)] \approx \mathbb{E}_{q(\boldsymbol{\Theta}_{\setminus\boldsymbol{\eta}_n})}\left[\boldsymbol{\rho}_n\right], \tag{5.27}$$

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 (5.25)), we do not have a close form to compute (5.27) either. But we have already known the form of $q(\Theta_{\setminus \boldsymbol{\eta}_n})$, so we can use the multivariate delta method [59, 6] to compute the expectation easily. Specifically, we use a first-order Taylor approximation to represent the conditional moments,

$$\boldsymbol{\rho}_n(\boldsymbol{\Theta}_{\setminus \boldsymbol{\eta}_n}) \approx \boldsymbol{\rho}_n \left(\mathbb{E}_q \left[\boldsymbol{\Theta}_{\setminus \boldsymbol{\eta}_n} \right] \right) + \mathbf{J} \cdot \left(\text{vec} \left(\boldsymbol{\Theta}_{\setminus \boldsymbol{\eta}_n} \right) - \text{vec} \left(\mathbb{E}_q \left[\boldsymbol{\Theta}_{\setminus \boldsymbol{\eta}_n} \right] \right) \right), \tag{5.28}$$

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

$$\mathbb{E}_{q(\boldsymbol{\Theta}_{\setminus \boldsymbol{\eta}_n})} \left[\boldsymbol{\rho}_n \right] \approx \boldsymbol{\rho}_n \left(\mathbb{E}_q \left[\boldsymbol{\Theta}_{\setminus \boldsymbol{\eta}_n} \right] \right). \tag{5.29}$$

We refer to [59, 98] 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 $\{\mathbf{u}_{i_{n_k}}^k\}$ and τ, and obtain their posterior in (5.20). Finally, we apply (5.21) 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 (5.29) and (5.27), 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 [56]. We repeatedly do message passing and conditional moment matching until convergence. The model inference is summarized in Algorithm 3.

5.4.3 Algorithm Complexity

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. As a comparison, the naive GP

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 (5.14) 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 (5.14) with conditional moment matching and multi-variate delta method.
  end for
until Convergence
Return: $\{q(\mathcal{W}(t_n))\}_{n=1}^N$, $\{q(\mathbf{u}_j^k)\}_{1\le k\le K, 1\le j\le d_k}$, $q(\tau)$

model demands $\mathcal{O}(\overline{R}N^3)$ time and $\mathcal{O}(\overline{R}N^2)$ space complexity, and hence can be extremely expensive or infeasible for large N. In practice, it is possible that multiple entries were observed at the same time point. Adjusting our method for such cases is trivial. Since the number of states is smaller than N accordingly, the complexity is even lower.

5.5 Related Work

There are many tensor decomposition methods, *e.g.,* [16, 42, 15, 106, 50, 64, 25, 87]. To utilize time information, current methods expand the tensor with a time mode [100, 73, 21, 108, 107, 2, 99], which comprises a set of discrete time steps, *e.g.,* by hours or days. The observed entry values are then arranged into different time slices of the tensor. The factors of the time steps and tensor nodes are jointly learned during the decomposition. To better estimate the temporal relationships, a few more refined approaches model the transition between the time steps, *e.g.,* a conditional linear Gaussian prior in [100], RNN [99], and kernel smoothing and regularization in [2]. To conduct continuous-time decomposition, the latest work [104] uses polynomial splines to estimate the factor coefficients in the CP model as a time function. Another set of works [77, 78, 105, 63, 94] decompose the events between the tensor nodes. The entry values are event counts or event sequences. These methods either use Poisson processes or more complex temporal point processes, such as Hawkes processes [34]. However, these methods do not consider the result of events, *e.g.,* payment or product ratings.

Message passing is a general inference framework in probabilistic graphical mod-

els [92]. When the model has a chain or tree structure and the factors in the graph (*i.e.,* terms in probability) are tractable, message passing can perform exact inference in a highly efficient way. Kalman filter and RTS smoothing are examples. When the factors are complex (*e.g.,* not in the exponential family), the computation of the messages can be intractable. [55] proposed a more general framework, Expectation propagation (EP), to handle the message computation via moment matching. However, it can still fail when moment matching is intractable. To address this problem, [96] proposed conditional EP (CEP) that uses conditional moment matching, Taylor approximations and numerical quadrature to compute the intractable moments for fully factorized posteriors. CEP has been used in Bayesian CP decomposition [96] and shown great performance. [23] has used CEP in the streaming inference of a sparse Tucker decomposition model where a spike-and-slab prior is placed on the tensor-core and approximated on the fly to obtain the running posterior. Our work uses GPs to estimate a time-varying tensor-core to handle continuous time information in the Tucker decomposition framework. To avoid huge kernel matrix computations and/or low-rank approximations, we use LTI-SDEs to build an equivalent state-space prior, which is essentially a Gaussian Markov chain. Under the chain structure, we combine the message passing and moment matching for efficient inference. We use similar ideas as in [96, 23] to compute the Gaussian messages to the SDE states. Given these messages, we then perform KF and RTS smoothing to calculate the posterior of the SDE states in an exact way, which are in turn used to update the approximation terms in each likelihood. In this way, we achieve the linear time complexity for our Tucker-GP model.

5.6 Experiment

5.6.1 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 $1 \leq j \leq 25$, and from $\mathcal{N}([1;-1], 0.1\mathbf{I})$ for $26 < j \leq 50$. Similarly, for the nodes in mode 2, we sampled $\mathbf{u}_j^2 \sim \mathcal{N}([1;1], 0.1\mathbf{I})$ when $1 \leq j \leq 25$, and $\mathcal{N}([-1;-1], 0.1\mathbf{I})$ for $26 < j \leq 50$. Given the latent factors, we generate

the tensor entry values at any time t from

$$y_{\mathbf{i}}(t) = u^1_{i_1,1} u^2_{i_2,1} w_{(1,1)}(t) + u^1_{i_1,1} u^2_{i_2,2} w_{(1,2)}(t) + u^1_{i_1,2} u^2_{i_2,1} w_{(2,1)}(t) + u^1_{i_1,2} u^2_{i_2,2} w_{(2,2)}(t), \quad (5.30)$$

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 [66]. 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. 5.2, our approach recovered each function pretty accurately, showing that BCTT has successful captured all the temporal dynamics within the factor interactions. Next, we show the learned factors in each mode in Fig. 5.3. The colors indicate the ground-truth cluster membership of the nodes. As we can see, our learned factors clearly revealed the hidden structures of the tensor nodes.

5.6.2 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* (kaggle/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.

[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).

5.6.2.1 Methods

We compared with following state-of-the-art multilinear and nonparametric tensor decomposition algorithms with time information integrated. (1) CT-CP [104], continuous-time CP decomposition, which uses polynomial splines to estimates $\boldsymbol{\lambda}$ in CP as a trend function. (2) CT-GP, continuous-time GP decomposition, which extends [106] to use GPs to learn tensor element as a function of the latent factors and time $y_{\mathbf{i}}(t) = g(\mathbf{u}^1_{i_1}, \ldots, \mathbf{u}^K_{i_K}, t) \sim \mathcal{GP}(0, \kappa(\cdot,\cdot))$. (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.

5.6.2.2 Settings and Results

All the methods were implemented by PyTorch. For {CT, DT, DDT}-GP, we used the square exponential kernel and sparse variational GP inference as in [108] for scalable model estimation. The number of pseudo inputs was 100. For CT-CP, we used 100 knots for the polynomial splines. Except BCTT, all the methods were trained with stochastic mini-batch optimization, with mini-batch size 100. We used ADAM optimization [43]. The learning rate was chosen from $\{10^{-4}, 5\times10^{-4}, 10^{-3}, 5\times10^{-3}, 10^{-2}\}$. We re-scaled all the timestamps to $[0, 10]$ to ensure numerical stability. We examined all the methods with the number of factors $R \in \{3, 5, 7, 9\}$. Following [42, 108], we randomly sampled 80% observed entry values and their time points for training, and then tested on the remaining entries. For discrete-time decomposition methods, we set the number of time steps to 50 (we tested with more steps but did not obtain improvement). We repeated the experiments for five times, and examined the average root mean-square-error (RMSE), average mean-absolute-error (MAE), and their standard deviations.

As shown in Table 5.1, our approach BCTT outperforms the competing methods in all the cases except that in Table 5.1d, 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 [23] 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. 5.4a-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.

5.7 Conclusion

We proposed BCTT, a continuous-time dynamic Tucker decomposition method. Our model maintains the interpretable structure while is flexible enough to capture various temporal dynamics within the factor interactions. Our LTI-SDE based message-passing inference avoids sparse GP approximations and enjoys a linear scalability with the data growth.

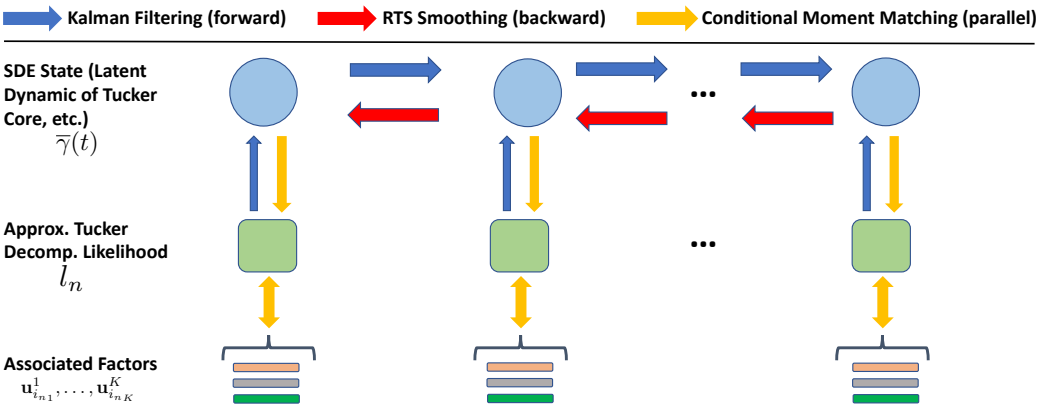
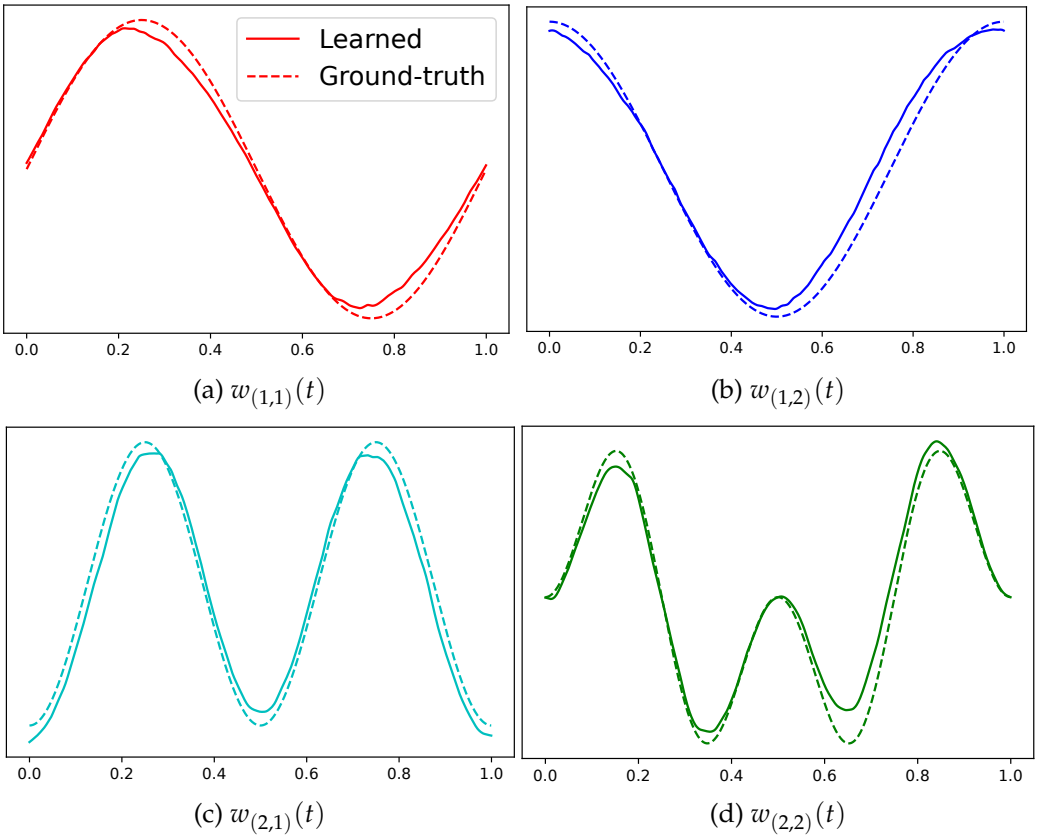


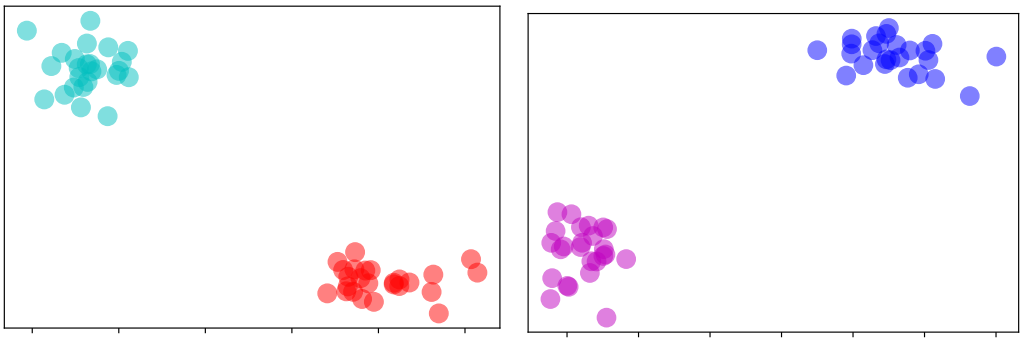
Figure 5.1: Graphical illustration of the message-passing inference algorithm.



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

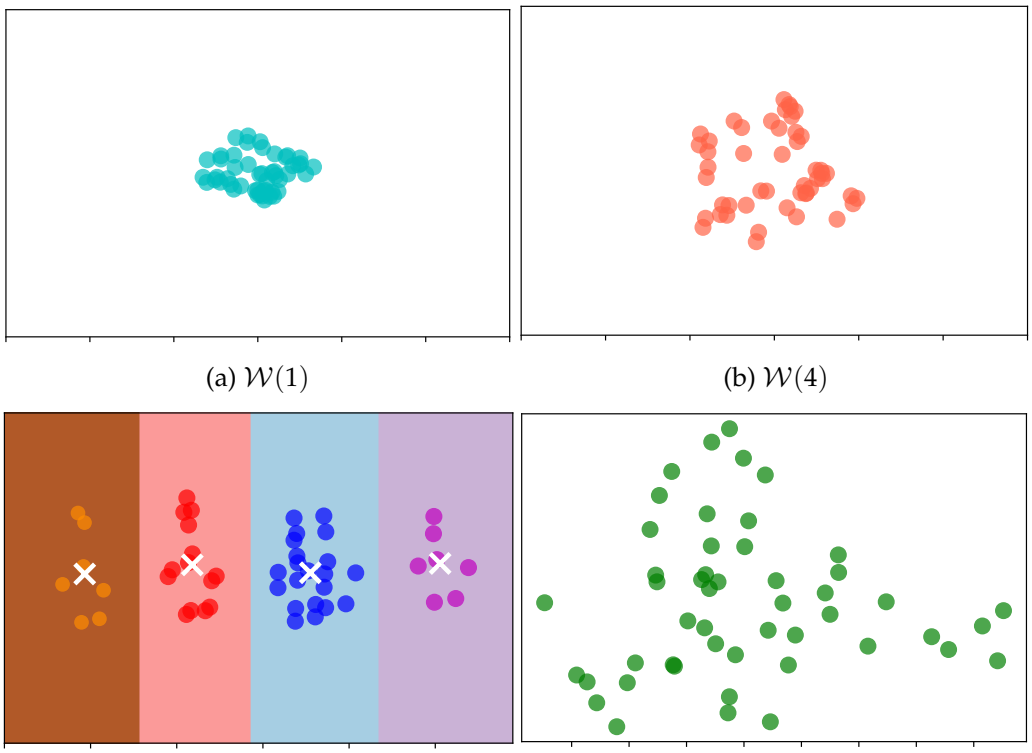
(c) $w_{(2,1)}(t)$ (d) $w_{(2,2)}(t)$

Figure 5.2: Recovered temporal dynamics within factor interactions.



(a) $\mathbf{U}^1$ (b) $\mathbf{U}^2$

Figure 5.3: The estimated latent factors by BCTT.



(a) $\mathcal{W}(1)$ (b) $\mathcal{W}(4)$

(c) $\mathcal{W}(7)$ (d) DDT-TD

Figure 5.4: 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).

Table 5.1: Prediction error and standard deviation averaged over five runs.

RMSE	*MovieLens*	*AdsClicks*	*DBLP*
CT-CP	1.113 ± 0.004	1.337 ± 0.013	0.240 ± 0.007
CT-GP	0.949 ± 0.008	1.422 ± 0.008	0.227 ± 0.009
DT-GP	0.963 ± 0.008	1.436 ± 0.015	0.227 ± 0.007
DDT-GP	0.957 ± 0.008	1.437 ± 0.010	0.225 ± 0.006
DDT-CP	1.022 ± 0.003	1.420 ± 0.020	0.245 ± 0.004
DDT-TD	1.059 ± 0.006	1.401 ± 0.022	0.232 ± 0.09
BCTT	$\mathbf{0.922 \pm 0.002}$	$\mathbf{1.322 \pm 0.012}$	$\mathbf{0.214 \pm 0.009}$
MAE			
CT-CP	0.788 ± 0.004	0.787 ± 0.006	0.105 ± 0.001
CT-GP	0.714 ± 0.004	0.891 ± 0.011	0.092 ± 0.004
DT-GP	0.722 ± 0.008	0.893 ± 0.008	0.084 ± 0.003
DDT-GP	0.720 ± 0.003	0.894 ± 0.009	0.083 ± 0.001
DDT-CP	0.755 ± 0.002	0.901 ± 0.011	0.114 ± 0.002
DDT-TD	0.742 ± 0.006	0.866 ± 0.012	0.101 ± 0.001
BCTT	$\mathbf{0.698 \pm 0.002}$	$\mathbf{0.777 \pm 0.016}$	$\mathbf{0.084 \pm 0.001}$

a $R = 3$

RMSE	*MovieLens*	*AdsClicks*	*DBLP*
CT-CP	1.165 ± 0.008	1.324 ± 0.013	0.263 ± 0.006
CT-GP	0.965 ± 0.019	1.410 ± 0.015	0.227 ± 0.007
DT-GP	0.949 ± 0.007	1.425 ± 0.015	0.225 ± 0.008
DDT-GP	0.948 ± 0.005	1.421 ± 0.012	0.220 ± 0.006
DDT-CP	1.141 ± 0.007	1.623 ± 0.013	0.282 ± 0.011
DDT-TD	0.944 ± 0.003	1.453 ± 0.035	0.312 ± 0.072
BCTT	$\mathbf{0.895 \pm 0.007}$	$\mathbf{1.304 \pm 0.018}$	$\mathbf{0.202 \pm 0.009}$
MAE			
CT-CP	0.835 ± 0.006	0.792 ± 0.007	0.128 ± 0.001
CT-GP	0.717 ± 0.012	0.883 ± 0.016	0.092 ± 0.002
DT-GP	0.714 ± 0.005	0.886 ± 0.012	0.084 ± 0.001
DDT-GP	0.707 ± 0.004	0.882 ± 0.015	0.082 ± 0.003
DDT-CP	0.843 ± 0.003	1.082 ± 0.013	0.141 ± 0.004
DDT-TD	0.712 ± 0.002	0.903 ± 0.024	0.221 ± 0.047
BCTT	$\mathbf{0.679 \pm 0.001}$	$\mathbf{0.785 \pm 0.010}$	$\mathbf{0.080 \pm 0.001}$

b $R = 7$

RMSE	*MovieLens*	*AdsClicks*	*DBLP*
CT-CP	1.026 ± 0.002	1.335 ± 0.012	0.244 ± 0.005
CT-GP	0.970 ± 0.011	1.425 ± 0.011	0.229 ± 0.009
DT-GP	0.952 ± 0.012	1.428 ± 0.015	0.226 ± 0.007
DDT-GP	0.949 ± 0.007	1.417 ± 0.013	0.226 ± 0.007
DDT-CP	1.087 ± 0.012	1.515 ± 0.023	0.257 ± 0.006
DDT-TD	1.050 ± 0.005	1.403 ± 0.053	0.277 ± 0.026
BCTT	$\mathbf{0.901 \pm 0.002}$	$\mathbf{1.317 \pm 0.046}$	$\mathbf{0.204 \pm 0.009}$
MAE			
CT-CP	0.813 ± 0.003	0.796 ± 0.006	0.112 ± 0.001
CT-GP	0.731 ± 0.007	0.890 ± 0.012	0.093 ± 0.002
DT-GP	0.720 ± 0.016	0.888 ± 0.011	0.085 ± 0.001
DDT-GP	0.715 ± 0.003	0.879 ± 0.016	0.085 ± 0.001
DDT-CP	0.807 ± 0.003	0.958 ± 0.012	0.120 ± 0.002
DDT-TD	0.784 ± 0.015	0.831 ± 0.038	0.171 ± 0.043
BCTT	$\mathbf{0.684 \pm 0.001}$	$\mathbf{0.776 \pm 0.013}$	$\mathbf{0.082 \pm 0.001}$

c $R = 5$

RMSE	*MovieLens*	*AdsClicks*	*DBLP*
CT-CP	1.188 ± 0.002	1.335 ± 0.015	0.265 ± 0.004
CT-GP	0.935 ± 0.009	1.406 ± 0.008	0.227 ± 0.008
DT-GP	0.945 ± 0.005	1.410 ± 0.003	0.222 ± 0.008
DDT-GP	0.939 ± 0.003	1.411 ± 0.004	0.217 ± 0.003
DDT-CP	1.117 ± 0.011	1.580 ± 0.022	0.292 ± 0.007
DDT-TD	0.956 ± 0.005	1.473 ± 0.045	0.345 ± 0.096
BCTT	$\mathbf{0.891 \pm 0.003}$	$\mathbf{1.308 \pm 0.026}$	$\mathbf{0.198 \pm 0.006}$
MAE			
CT-CP	0.856 ± 0.003	$\mathbf{0.786 \pm 0.007}$	0.131 ± 0.001
CT-GP	0.703 ± 0.006	0.889 ± 0.009	0.094 ± 0.004
DT-GP	0.713 ± 0.003	0.880 ± 0.003	0.082 ± 0.002
DDT-GP	0.706 ± 0.005	0.874 ± 0.004	0.080 ± 0.001
DDT-CP	0.872 ± 0.006	1.024 ± 0.013	0.155 ± 0.005
DDT-TD	0.718 ± 0.004	0.923 ± 0.034	0.201 ± 0.053
BCTT	$\mathbf{0.678 \pm 0.002}$	0.787 ± 0.008	$\mathbf{0.079 \pm 0.002}$

d $R = 9$

CHAPTER 6

STREAMING FACTOR TRAJECTORY LEARNING FOR TEMPORAL TENSOR DECOMPOSITION

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 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 Rauch-Tung-Striebel smoothing to compute the full posterior of all the trajectories in parallel, without the need for revisiting any previous data. We have shown the advantage of SFTL in both synthetic tasks and real-world applications.

6.1 Introduction

Tensor data is common in real-world applications. For example, one can extract a three-mode tensor *(patient, drug, clinic)* from medical service records and a four-mode tensor *(customer, commodity, seller, web-page)* from the database of an online shopping platform. Tensor decomposition is a fundamental tool for tensor data analysis. It introduces a set of factors to represent the objects in each mode, and estimate these factors by reconstructing the observed entry values. These factors can be viewed as the underlying properties

of the objects. We can use them to search for interesting structures within the objects (*e.g.,* communities and outliers) or as discriminate features for predictive tasks, such as personalized treatment or recommendation.

Real-world tensor data is often accompanied with time information, namely the timestamps at which the objects of different modes interact to produce the entry values. Underlying the timestamps can be rich, valuable temporal patterns. While many temporal decomposition methods are available, most of them estimate a set of static factors for each object — they either introduce a discrete time mode [100, 106] or inject the timestamp into the decomposition model [104, 24, 48]. Hence, these methods cannot learn the temporal variation of the factors. Accordingly, they can miss important evolution of the objects' inner properties, such as health and income. In addition, practical applications produce data streams at a rapid pace [21]. Due to the resource limit, it is often prohibitively expensive to decompose the entire tensor from scratch whenever we receive new data. Many privacy-protecting applications (*e.g.,* SnapChat) even forbid us to preserve or re-access the previous data. Therefore, not only do we need a more powerful decomposition model that can estimate the factor evolution, we also need an effective method to capture such evolution from fast data streams.

To address these issues, we propose SFTL, a Bayesian streaming factor trajectory learning approach for temporal tensor decomposition. Our method can efficiently handle data streams, with which to estimate a posterior distribution of the factor trajectory to uncover the temporal evolution of the objects' representation. Our method never needs to keep or re-visit previous data. The contribution of our work is summarized as follows.

- First, we use a Gaussian process (GP) prior to sample the factor trajectory of each object as a function of time. As a nonparametric function prior, GPs are flexible enough to capture a variety of complex temporal dynamics. The trajectories are combined through a CANDECOMP/PARAFAC (CP) [32] or Tucker decomposition [91] form to sample the tensor entry values at any time point.
- Second, to sidestep the expensive covariance matrix computation in the GP, which further causes challenges in streaming inference, we use spectral analysis to convert the GP into a linear time invariant (LTI) stochastic differential equation (SDE). Then we convert the SDE into an equivalent state-space prior over the factor (trajectory)

states at the observed timestamps. As a result, the posterior inference becomes easier and computationally efficient.

- Third, we take advantage of the chain structure of the state-space prior and use the recent conditional expectation propagation framework [96] to develop an efficient online filtering algorithm. Whenever a collection of entries at a new timestamp arrives, our method can efficiently estimate a decoupled running posterior of the involved factor states, with a Gaussian product form. The decoupled Gaussian estimate enables us to run standard RTS smoothing [74] to compute the full posterior of each factor trajectory independently and in parallel, without revisiting any previous data. Our method at worst has a linear scalability with respect to the number of observed timestamps.

We first evaluated our method in a simulation study. On synthetic datasets, SFTL successfully recovered several nonlinear factor trajectories, and provided reasonable uncertainty estimation. We then tested our method on four real-world temporal tensor datasets for missing value prediction. In both online and final predictive performance, SFTL consistently outperforms the state-of-the-art streaming CP and Tucker decomposition algorithms by a large margin. In most cases, the prediction accuracy of SFTL is even higher than the recent static decomposition methods, which have to pass through the dataset many times. Finally, we investigated the learned factor trajectories from a real-world dataset. The trajectories exhibit interesting temporal evolution.

6.2 Bayesian Temporal Tensor Decomposition with Factor Trajectories

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 $\mathbf{u}_j^m(t) = [u_{j,1}^m(t), \ldots, u_{j,R}^m(t)]^\top$, *i.e.*, $u_{j,r}^m(t) \sim \mathcal{GP}(0, \kappa(t, t'))$ $(1 \leq r \leq R)$. 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

$$p(y_\ell(t)|\mathcal{U}(t)) = \mathcal{N}(y_\ell(t)|\mathbf{1}^\top(\mathbf{u}^1_{\ell_1}(t) \circ \ldots \circ \mathbf{u}^M_{\ell_M}(t)), \tau^{-1}), \tag{6.1}$$

where $\mathcal{U}(t) = \{\mathbf{u}^m_j(t)\}$ includes all the factor trajectories, and τ is the inverse noise variance, for which we assign a Gamma prior, $p(\tau) = \text{Gam}(\tau|\alpha_0, \alpha_1)$. For the Tucker form, we have $p(y_\ell(t)|\mathcal{U}(t), \mathcal{W}) = \mathcal{N}(y_\ell(t)|\text{vec}(\mathcal{W})^\top(\mathbf{u}^1_{\ell_1}(t) \otimes \ldots \otimes \mathbf{u}^M_{\ell_M}(t)), \tau^{-1})$ 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:

$$\mathcal{D} = \{(\ell_1, y_1, t_1), \ldots, (\ell_N, y_N, t_N)\}, \tag{6.2}$$

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 $s^m_{j,1} < \ldots < s^m_{j,c^m_j}$, where c^m_j is the participation count of the object. Note that it is a sub-sequence of $\{t_n\}$. From the GP prior, the values of each $u^m_{j,r}(t)$ at these timestamps follow a multi-variate Gaussian distribution, $p(\mathbf{u}^m_{j,r}) = \mathcal{N}(\mathbf{u}^m_{j,r}|\mathbf{0}, \mathbf{K}^m_j)$ where $\mathbf{u}^m_{j,r} = [u^m_{j,r}(s^m_{j,1}), \ldots, u^m_{j,r}(s^m_{j,c^m_j})]^\top$[1] and $\mathbf{K}^m_j$ is the covariance/kernel matrix computed at these timestamps. The joint probability with the CP form is

$$\begin{aligned} &p(\{\mathbf{u}^m_{j,r}\}, \tau, \mathbf{y}) = \textstyle\prod_{m=1}^M \prod_{j=1}^{d_m} \prod_{r=1}^R \mathcal{N}(\mathbf{u}^m_{j,r}|\mathbf{0}, \mathbf{K}^m_j) \cdot \text{Gam}(\tau|\alpha_0, \alpha_1) \\ &\cdot \textstyle\prod_{n=1}^N \mathcal{N}(y_n|\mathbf{1}^\top(\mathbf{u}^1_{\ell_{n1}}(t_n) \circ \ldots \circ \mathbf{u}^M_{\ell_{nM}}(t_n)), \tau^{-1}). \end{aligned} \tag{6.3}$$

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 (6.3), *i.e.*, $\{\mathcal{N}(\mathbf{u}^m_{j,r}|\mathbf{0}, \mathbf{K}^m_j)\}$. The time and space complexity to compute each $\mathcal{N}(\mathbf{u}^m_{j,r}|\mathbf{0}, \mathbf{K}^m_j)$ is $\mathcal{O}\left(\left(c^m_j\right)^3\right)$ and $\mathcal{O}\left(\left(c^m_j\right)^2\right)$, respectively. With the increase of N, the appearance count c^m_j for many objects can grow as well, making the computation cost very expensive or even

[1] For convenience, we abuse the notation a little bit: we denote by $\mathbf{u}^m_{j,r}(\cdot)$ trajectory function and by $\mathbf{u}^m_{j,r}$ the values of the trajectory function at the observed timestamps.

infeasible. The issue is particularly severe when we handle streaming data — the number of timestamps grows rapidly when new data keeps coming in, so does the size of each covariance matrix.

6.2.1 Equivalent Modeling with State-Space Priors

To sidestep expensive covariance matrix computation and ease the inference with streaming data, we follow [33] to convert the GP prior into an SDE via spectral analysis. We use a Matérn kernel $\kappa_\nu(t,t') = a\frac{\left(\frac{\sqrt{2\nu}}{\rho}\Delta\right)^\nu}{\Gamma(\nu)2^{\nu-1}}K_\nu\left(\frac{\sqrt{2\nu}}{\rho}\Delta\right)$ where $\Gamma(\cdot)$ is the Gamma function, $\Delta = |t-t'|$, $a > 0$, $\rho > 0$, K_ν is the modified Bessel function of the second kind, and $\nu = p + \frac{1}{2}$ ($p \in \{0,1,2,\ldots\}$) as the GP covariance. Via the analysis of the power spectrum of κ_ν, we can show that if $f(t) \sim \mathcal{GP}(0,\kappa_\nu(t,t'))$, it can be characterized by a linear time-invariant (LTI) SDE, with state $\mathbf{z} = (f, f^{(1)}, \ldots, f^{(p)})^\top$ where $f^{(k)} \triangleq \mathrm{d}^k f/\mathrm{d}t^k$,

$$\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = \mathbf{A}\mathbf{z} + \boldsymbol{\eta}\cdot\beta(t), \tag{6.4}$$

where $\beta(t)$ is a white noise process with diffusion σ^2,

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & 0 & 1 \\ -c_0 & \ldots & -c_{p-1} & -c_p \end{pmatrix}, \quad \boldsymbol{\eta} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}. \tag{6.5}$$

Both σ^2 and $\mathbf{A}$ are obtained from the parameters in κ_ν. The LTI-SDE is particularly useful in that its finite set of states follow a Gauss-Markov chain, *i.e.,* the state-space prior. Given arbitrary $t_1 < \ldots < t_L$, we have

$$p(\mathbf{z}(t_1),\ldots,\mathbf{z}(t_L)) = p(\mathbf{z}(t_1))\textstyle\prod_{k=1}^{L-1} p(\mathbf{z}(t_{k+1})|\mathbf{z}(t_k)), \tag{6.6}$$

where $p(\mathbf{z}(t_1)) = \mathcal{N}(\mathbf{z}(t_1)|\mathbf{0},\mathbf{P}_\infty)$, $p(\mathbf{z}(t_{k+1})|\mathbf{z}(t_k)) = \mathcal{N}(\mathbf{z}(t_{k+1})|\mathbf{F}_k\mathbf{z}(t_k),\mathbf{Q}_k)$, $\mathbf{P}_\infty$ is the stationary covariance matrix computed by solving the matrix Riccati equation [46], $\mathbf{F}_n = \exp(\Delta_k\cdot\mathbf{A})$ where $\Delta_k = t_{k+1} - t_k$, and $\mathbf{Q}_k = \mathbf{P}_\infty - \mathbf{A}_k\mathbf{P}_\infty\mathbf{A}_k^\top$. Therefore, we do not need the full covariance matrix as in the standard GP prior, and the computation is much more efficient. The chain structure is also convenient to handle streaming data as we will explain later.

We therefore convert the GP prior over each factor trajectory $u_{j,r}^m(t)$ into an LTI-SDE. We denote the corresponding state by $\mathbf{z}_{j,r}^m(t)$. For example, if we choose $p = 1$, then $\mathbf{z}_{j,r}^m(t) =$

$[u^m_{j,r}(t); \mathrm{d}u^m_{j,r}(t)/\mathrm{d}t]$. For each object j in mode m, we concatenate all its trajectory states into one, $\mathbf{z}^m_j(t) = [\mathbf{z}^m_{j,1}(t); \ldots; \mathbf{z}^m_{j,R}(t)]$. Then on all of its timestamps $s^m_{j,1} < \ldots < s^m_{j,c^m_j}$, we obtain a state-space prior

$$p(\mathbf{h}^m_{j,1}) = \mathcal{N}(\mathbf{h}^m_{j,1}|\mathbf{0}, \overline{\mathbf{P}}_\infty), \;\; p(\mathbf{h}^m_{j,k+1}|\mathbf{h}^m_{j,k}) = \mathcal{N}(\mathbf{h}^m_{j,k+1}|\overline{\mathbf{F}}^m_{j,k}\mathbf{h}^m_{j,k}, \overline{\mathbf{Q}}^m_{j,k}), \tag{6.7}$$

where $\mathbf{h}^m_{j,k} \triangleq \mathbf{z}^m_j(s^m_{j,k})$, $\overline{\mathbf{P}}_\infty = \mathrm{diag}(\mathbf{P}_\infty, \ldots, \mathbf{P}_\infty)$, $\overline{\mathbf{F}}^m_{j,k} = \mathrm{diag}\left(\mathbf{F}^m_{j,k}, \ldots, \mathbf{F}^m_{j,k}\right)$, $\mathbf{F}^m_{j,k} = e^{(s^m_{j,k+1} - s^m_{j,k})\mathbf{A}}$, $\overline{\mathbf{Q}}^m_{j,k} = \mathrm{diag}\left(\mathbf{Q}^m_{j,k}, \ldots, \mathbf{Q}^m_{j,k}\right)$, and $\mathbf{Q}^m_{j,k} = \mathbf{P}_\infty - \mathbf{F}^m_{j,k}\mathbf{P}_\infty\left(\mathbf{F}^m_{j,k}\right)^\top$.

The joint probability of our model with the CP form now becomes

$$\begin{aligned} &p(\{\mathbf{h}^m_{j,k}\}, \tau, \mathbf{y}) = p(\tau)\textstyle\prod_{m=1}^M \prod_{j=1}^{d_m} p(\mathbf{h}^m_{j,1}) \prod_{k=1}^{c^m_j - 1} p(\mathbf{h}^m_{j,k+1}|\mathbf{h}^m_{j,k}) \\ &\cdot \textstyle\prod_{n=1}^N \mathcal{N}(y_n|\mathbf{1}^\top(\mathbf{u}^1_{\ell_{n1}}(t_n) \circ \ldots \circ \mathbf{u}^M_{\ell_{nM}}(t_n)), \tau^{-1}). \end{aligned} \tag{6.8}$$

Note that in the likelihood, each $\mathbf{u}^m_{\ell_{nm}}(t_n)(1 \le j \le M)$ is contained in a corresponding state vector $\mathbf{h}^m_{\ell_{nm},k}$ such that $s^m_{\ell_{nm},k} = t_n$ (by definition, we then have $\mathbf{h}^m_{\ell_{nm},k} = \mathbf{z}^m_{\ell_{nm}}(t_n)$). The joint probability with the Tucker form is similar, which we omit to save the space.

6.3 Trajectory Inference from Streaming Data

In this section, we develop an efficient, scalable algorithm for factor trajectory estimation from streaming data. In general, we assume that we receive a sequence of (small) batches of observed tensor entries, $\{\mathcal{B}_1, \mathcal{B}_2, \ldots\}$, generated at different timestamps, $\{t_1, t_2, \ldots\}$. Each batch $\mathcal{B}_n$ is generated at timestamp t_n and $t_n < t_{n+1}$. Denote by $\mathcal{D}_{t_n}$ all the data up to timestamp t_n, *i.e.,* $\mathcal{D}_{t_n} = \mathcal{B}_1 \cup \ldots \cup \mathcal{B}_n$. Upon receiving $\mathcal{B}_{n+1}$, we intend to update our model without revisiting $\mathcal{D}_{t_n}$ to provide the trajectory posterior estimate, $\{p(\mathbf{u}^m_j(t)|\mathcal{D}_{t_{n+1}})|\forall t \ge 0, 1 \le m \le M, 1 \le j \le d_m\}$, where $\mathcal{D}_{t_{n+1}} = \mathcal{D}_{t_n} \cup \mathcal{B}_{n+1}$.

To this end, we first observe that the standard state-space model with a Gaussian likelihood has already provided a highly efficient streaming inference framework. Denote by $\mathbf{x}_n$ and $\mathbf{y}_n$ the state and observation at each step n, respectively. To handle streaming observations $\{\mathbf{y}_1, \mathbf{y}_2, \ldots\}$, we only need to compute and track the running posterior $p(\mathbf{x}_n|\mathbf{y}_{1:n})$ upon receiving each $\mathbf{y}_n$, where $\mathbf{y}_{1:n}$ denotes the total data up to step n. This is called Kalman filtering [74], which only depends on the running posterior at step $n-1$, *i.e.,* $p(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1})$, and is highly efficient. After all the data is processed (suppose it stops at step N), we can use Rauch-Tung-Striebel (RTS) smoother [74] to efficiently compute the

full posterior of each state, $p(\mathbf{x}_n|\mathbf{y}_{1:N})$, from backward, which does not need to re-access any previous observations.

However, one cannot apply the above framework outright to our model, since the tensor decomposition likelihood of each observed entry couples the states of multiple factor trajectories, see (6.1) — which correspond to the participated objects at different modes. That means, the factor-state chains of different objects are dynamically intertwined through the received data. The multiplicative form of these states in the likelihood render the running posterior of each trajectory intractable to compute, not to mention running RTS smoother. To address this challenge, we take advantage of the chain structure and use the recent conditional Expectation propagation (CEP) framework [96] to develop an efficient online filtering algorithm, which approximates the running posterior of the involved factor states as a product of Gaussian. Thereby, we can decouple the involved factor state chains, and conduct standard RTS smoothing for each chain independently.

Specifically, denote the sequence of timestamps when each object j of mode m has showed up in the data stream up to t_n, by $s^m_{j,1} < s^m_{j,2} < \ldots < s^m_{j,c^m_{j,n}}$ where $c^m_{j,n}$ is the object's appearance count up to t_n. Denote by $\mathcal{I}^m_n$ the indexes of all the objects of mode m appearing in $\mathcal{B}_n$. Hence, for every object $j \in \mathcal{I}^m_n$, we have $s^m_{j,c^m_{j,n}} = t_n$, and $\mathbf{h}^m_{j,c^m_{j,n}} \overset{\Delta}{=} \mathbf{z}^m_j(t_n)$ is the factor state of the object at t_n. Upon receiving each $\mathcal{B}_n$, we intend to approximate the running posterior of all the involved factor states and noise inverse variance τ with the following decoupled form,

$$p(\tau, \{\mathbf{h}^m_{j,c^m_{j,n}}|j \in \mathcal{I}^m_n\}_{1\leq m\leq M}|\mathcal{D}_{t_n}) \approx q(\tau|\mathcal{D}_n)\prod_{m=1}^M \prod_{j\in\mathcal{I}^m_n} q(\mathbf{h}^m_{j,c^m_{j,n}}|\mathcal{D}_{t_n}), \tag{6.9}$$

where $q(\tau|\mathcal{D}_{t_n}) = \text{Gam}(\tau|a_n, b_n)$, and $q(\mathbf{h}^m_{j,c^m_{j,n}}|\mathcal{D}_{t_n}) = \mathcal{N}(\mathbf{h}^m_{j,c^m_{j,n}}|\widehat{\boldsymbol{\mu}}^m_{j,c^m_{j,n}}, \widehat{\mathbf{V}}^m_{j,c^m_{j,n}})$. To this end, let us consider given the approximation at t_n, how to obtain the new approximation at t_{n+1} (*i.e.,* upon receiving $\mathcal{B}_{n+1}$) in the same form of (6.9). To simplify the notation, let us define the preceding states of the involved factors by $\Theta_n = \{\mathbf{h}^m_{j,c^m_{j,n}}|j \in \mathcal{I}^m_{n+1}\}_m$, and the current states by $\Theta_{n+1} = \{\mathbf{h}^m_{j,c^m_{j,n+1}}|j \in \mathcal{I}^m_{n+1}\}_m$. First, due to the chain structure of the prior over each $\{\mathbf{h}^m_{j,c^m_{j,k}}|k = 0, 1, 2, \ldots\}$, we can see that conditioned on $\{\Theta_n, \tau\}$, the current states Θ_{n+1} and the new observations $\mathcal{B}_{n+1}$ are independent of $\mathcal{D}_{t_n}$. This is because in the graphical model representation, $\{\Theta_n, \tau\}$ have blocked all the paths from the old observations $\mathcal{D}_{t_n}$ to the new state and observations [8]; see Fig. 6.1 for an illustration. Then, we can derive that

$$p(\Theta_{n+1}, \Theta_n, \tau|\mathcal{D}_{t_{n+1}}) \propto p(\Theta_{n+1}, \Theta_n, \tau, \mathcal{B}_{n+1}|\mathcal{D}_{t_n}) = p(\Theta_n, \tau|\mathcal{D}_{t_n}) p(\Theta_{n+1}, \mathcal{B}_{n+1}|\Theta_n, \tau, \mathcal{D}_{t_n})$$
$$= p(\Theta_n, \tau|\mathcal{D}_{t_n}) p(\Theta_{n+1}|\Theta_n) p(\mathcal{B}_{n+1}|\Theta_{n+1}, \tau), \tag{6.10}$$

where $p(\Theta_n, \tau|\mathcal{D}_{t_n})$ is the running posterior at t_n,

$$p(\Theta_{n+1}|\Theta_n) = \prod_{m=1}^{M} \prod_{j \in \mathcal{I}_{n+1}^m} p(\mathbf{h}_{j,c_{j,n+1}^m}^m | \mathbf{h}_{j,c_{j,n}^m}^m), \tag{6.11}$$

each $p(\mathbf{h}_{j,c_{j,n+1}^m}^m | \mathbf{h}_{j,c_{j,n}^m}^m)$ is a conditional Gaussian distribution defined in (6.7), and $p(\mathcal{B}_{n+1}|\Theta_{n+1}, \tau) = \prod_{(\ell,y)\in\mathcal{B}_{n+1}} \mathcal{N}\left(y|\mathbf{1}^\top \left(\mathbf{u}_{\ell_1}^1(t_{n+1}) \circ \cdots \circ \mathbf{u}_{\ell_M}^M(t_{n+1})\right), \tau^{-1}\right)$. Since $p(\Theta_n, \tau|\mathcal{D}_{t_n})$ takes the form of (6.9), we can analytically marginalize out each $\mathbf{h}_{j,c_{j,n}^m}^m \in \Theta_n$, and obtain

$$p\left(\Theta_{n+1}, \tau|\mathcal{D}_{t_{n+1}}\right) \propto \mathrm{Gam}(\tau|a_n, b_n) \prod_{m=1}^{M} \prod_{j \in \mathcal{I}_{n+1}^m} \mathcal{N}(\mathbf{h}_{j,c_{j,n+1}^m}^m | \widehat{\boldsymbol{\mu}}_{j,c_{j,n+1}^m}^m, \widehat{\mathbf{V}}_{j,c_{j,n+1}^m}^m) \tag{6.12}$$
$$\cdot \prod_{(\ell,y)\in\mathcal{B}_{n+1}} \mathcal{N}\left(y|\mathbf{1}^\top \left(\mathbf{u}_{\ell_1}^1(t_{n+1}) \circ \cdots \circ \mathbf{u}_{\ell_M}^M(t_{n+1})\right), \tau^{-1}\right). \tag{6.13}$$

If we view the R.H.S of (6.13) as a joint distribution with $\mathcal{B}_{n+1}$, then our task amounts to estimating the posterior distribution, *i.e.,* the L.H.S of (6.13). The product in the CP likelihood (and also Tucker likelihood) renders exact posterior computation infeasible, and we henceforth approximate

$$\mathcal{N}\left(y|\mathbf{1}^\top \left(\mathbf{u}_{\ell_1}^1(t_{n+1}) \circ \cdots \circ \mathbf{u}_{\ell_M}^M(t_{n+1})\right)\right) \overset{\propto}{\approx} \prod_{m=1}^{M} \mathcal{N}\left(\mathbf{u}_{\ell_m}^m(t_{n+1}) | \boldsymbol{\gamma}_{\ell_m}^m, \boldsymbol{\Sigma}_{\ell_m}^m\right) \mathrm{Gam}(\tau|\alpha_\ell, \omega_\ell) \tag{6.14}$$

where $\overset{\propto}{\approx}$ means approximately proportional to. To optimize these approximation terms, we use the recent conditional Expectation propagation (CEP) framework [96] to develop an efficient inference algorithm. It uses conditional moment matching to update each approximation in parallel and conducts fixed point iterations, and hence can converge fast. Once it is done, we substitute the approximation (6.14) into (6.13). Then the R.H.S of (6.13) becomes a product of Gaussian and Gamma terms over each state and τ. We can then immediately obtain a closed-form estimation in the form as (6.9). At the beginning, when estimating $p(\Theta_1, \tau|\mathcal{D}_{t_1})$, since the preceding states $\Theta_0 = \emptyset$, we have $a_n = \alpha_0$, $b_n = \alpha_1$, $\widehat{\boldsymbol{\mu}}_{j,c_{j,n+1}^m}^m = \mathbf{0}$, and $\widehat{\mathbf{V}}_{j,c_{j,n+1}^m}^m = \overline{\mathbf{P}}_\infty$ in (6.13), which is the prior of each $\mathbf{h}_{j,c_{j,1}^m}^m$ and τ (see (6.7)).

In this way, we can continuously filter the incoming batches $\{\mathcal{B}_1, \mathcal{B}_2, \ldots\}$. As a result, for the factor state chain of every object j in every mode m, along with each timestamp

$s^m_{j,k}$, we can online estimate and track a running posterior approximation $\{q(\mathbf{h}^m_{j,k}|\mathcal{D}_{t_{s^m_{j,k}}})|k = 1, 2, \ldots\}$, which is a Gaussian distribution. Hence, we can run the standard RTS smoother, to compute the full posterior of every factor state, with which we can compute the posterior of the trajectory at any time point t [8]. Our method is summarized in Algorithm 4.

The time complexity of our algorithm processing a batch $\mathcal{B}_n$ is $\mathcal{O}(|\mathcal{B}_n|R^3)$, where $|\mathcal{B}_n|$ is the size of the batch. The time complexity of RTS smoother for a particular object j in mode m is $\mathcal{O}(R^3 c^m_{j,N})$, where N is the total number of timestamps. The space complexity of our algorithm is $\mathcal{O}(\sum_{m=1}^{M}\sum_{j=1}^{d_m} c^m_{j,N}R^2)$, which is to track the running posterior of the factor state at each appearing timestamp for every object. Since $c^m_{j,N} \leq N$, the complexity of our algorithm is at worst linear in N.

6.4 Related Work

Many tensor decomposition methods have been developed, such as [102, 69, 107, 108, 106, 86, 64, 23, 25, 88, 87, 24, 105, 63, 94, 97]. For temporal decomposition, most existing methods augment the tensor with a discrete time mode to estimate additional factors for

Algorithm 4 Streaming Factor Trajectory Learning (SFTL)

Input: kernel hyper-parameters $a, \rho, \nu = p + \frac{1}{2}$ $(p \in \{0, 1, 2, \ldots\})$
$n \leftarrow 0$
while Receiving a new batch entreis $\mathcal{B}_{n+1}$ **do**
  if $n = 0$ **then**
    Set $a_n = a_0$, $b_n = b_0$, $\widehat{\boldsymbol{\mu}}^m_{j,c^m_{j,n+1}} = \mathbf{0}$, and $\widehat{\mathbf{V}}^m_{j,c^m_{j,n+1}} = \overline{\mathbf{P}}_\infty$ in (6.13).
    Goto 9.
  end if
  Retrieve the involved preceding factor states $\Theta_n = \{\mathbf{h}^m_{j,c^m_{j,n}}|j \in \mathcal{I}^m_{n+1}\}_m$ and their running posterior, $p(\Theta_n, \tau|\mathcal{D}_{t_n}) \approx \text{Gam}(\tau|a_n, b_n)\prod_{m=1}^{M}\prod_{j\in\mathcal{I}^m_{n+1}}\mathcal{N}(\mathbf{h}^m_{j,c^m_{j,n}}|\widehat{\boldsymbol{\mu}}^m_{j,c^m_{j,n}}, \widehat{\mathbf{V}}^m_{j,c^m_{j,n}})$.
  According to (6.13) and (6.14), use conditional Expectation Propagation to calculate the running posterior of the current factor states, $p(\Theta_{n+1}, \tau|\mathcal{D}_{t_{n+1}}) \approx \text{Gam}(\tau|a_{n+1}, b_{n+1})\prod_{m=1}^{M}\prod_{j\in\mathcal{I}^m_{n+1}}\mathcal{N}(\mathbf{h}^m_{j,c^m_{j,n+1}}|\widehat{\boldsymbol{\mu}}^m_{j,c^m_{j,n+1}}, \widehat{\mathbf{V}}^m_{j,c^m_{j,n+1}})$.
  if Needed **then**
    Run RTS smoothing on any factor state chain $\{\mathbf{h}^m_{j,k}|k = 1, 2, \ldots\}$ of interest.
  end if
  $n \leftarrow n + 1$
end while
Run RTS smoothing for every factor state chain $\{\mathbf{h}^m_{j,k}|k = 1, 2, \ldots\}$.
Return: $\{q(\mathbf{h}^m_{j,k}|\mathcal{D})|k = 1, 2, \ldots\}_{1\leq m\leq M, 1\leq j\leq d_m}$, $q(\tau|\mathcal{D})$, where $\mathcal{D}$ is all the data received.

time steps, *e.g.,* [100, 73, 84, 21, 2]. The most recent works have conducted continuous-time decomposition. Recent work [104] used polynomial splines to model a time function as the CP coefficients. Another work [48] used neuralODE [12] to model the entry value as a function of latent factors and time point. Recent work [24] performed continuous-time Tucker decomposition, and modeled the tensor-core as a time function. To our knowledge, [97] is the first work to estimate factor trajectories. It places a GP prior in the frequency domain, and samples the factor trajectories via inverse Fourier transform. It then uses another GP to sample the entry values. While successful, this method cannot handle streaming data, and the black-box GP decomposition lacks interpretability.

Current Bayesian streaming tensor decomposition methods include [21, 23, 64, 25], which are based on streaming variational Bayes [10] or assumed density filtering (ADF) [9]. ADF can be viewed as an instance of Expectation Propagation (EP) [55] for streaming data. EP approximates complex terms in the probability distribution with exponential-family members, and uses moment matching to iteratively update the approximations, which essentially is a fixed point iteration. To address the challenge of intractable moment matching, [96] proposed conditional EP (CEP), which uses conditional moment matching and Taylor expansion to compute the moments for factorized approximations. The theoretical guarantees and error bound analysis for EP and ADF have been studied for a long time, such as [9, 19, 18]. The most recent work [24] also uses SDEs to represent GPs and CEP framework for inference, but their GP prior is placed on the tensor-core, not for learning factor trajectories, and their method is only for static decomposition, and cannot handle streaming data.

6.5 Experiment

6.5.1 Simulation Study

We first conducted a simulation study to verify the proposed method, for which we simulated a two-mode tensor, with two nodes per mode. Each node is represented by a time-varying factor:

$$u_1^1(t) = -\sin^3(2\pi t),\ u_2^1(t) = \left(1 - \sin^3(\frac{1}{2}\pi t)\right)\sin^3(3\pi t), \tag{6.15}$$

$$u_1^2(t) = \sin(2\pi t),\ u_2^2(t) = -\cos^3(3\pi t)\sin(3\pi t)\sin(2\pi t). \tag{6.16}$$

Given these factors, an entry value at time t is generated via $y_{(i,j)}(t) \sim \mathcal{N}\left(u_i^1(t)u_j^2(t), 0.05\right)$. We randomly sampled 500 (irregular) timestamps from $[0, 1]$. For each timestamp, we randomly picked two entries, and sampled their values accordingly. Overall, we sampled 1,000 observed tensor entry values.

We implemented SFTL with PyTorch [66]. We used $\nu = \frac{3}{2}$ and $a = \rho = 0.3$ for the Matérn kernel. We streamed the sampled entries according to their timestamps, and ran our streaming factor trajectory inference based on the CP form. The estimated trajectories are shown in Fig. 6.2. The shaded region indicates the posterior standard deviation. As we can see, SFTL recovers the ground truth pretty accurately, showing that SFTL has successfully captured the temporal evolution of the factor representation for every node. It is interesting to observe that when t is around 0, 0.5 and 1, the posterior standard deviation (the shaded region) increases significantly. This is reasonable: the ground-truth trajectories overlap at these time points, making it more difficult to differentiate/estimate their values at these time points. Accordingly, the uncertainty of the estimation increases.

6.5.2 Real-World Applications

6.5.2.1 Datasets and Settings

Next, we examined SFTL in four real-world datasets: *FitRecord*, *ServerRoom*, *BeijingAir-2*, and *BeijingAir-3*. We tested 11 competing approaches. We compared with state-of-the-art streaming tensor decomposition methods based on the CP or Tucker model, including (1) POST [21], (2) BASS-Tucker [23] and (3) ADF-CP [96], the state-of-the-art static decomposition algorithms, including (4) P-Tucker [60], (5) CP-ALS and (6) Tucker-ALS [4]. For those methods, we augment the tensor with a time mode, and convert the ordered, unique timestamps into increasing time steps. We also compared with the most recent continuous-time decomposition methods. (7) CT-CP [104], (8) CT-GP, (9) BCTT [24], (10) THIS-ODE [48], and (11) NONFAT [97], nonparametric factor trajectory learning, the only existing work that also estimates factor trajectories for temporal tensor decomposition. Note that the methods 4-11 cannot handle data streams. They have to iteratively access the data to update the model parameters and factor estimates.

For all the competing methods, we used the publicly released implementations of the original authors. The hyper-parameter setting and turning follows the original papers. For

SFTL, we chose ν from $\{\frac{1}{2}, \frac{3}{2}\}$, a from [0.5, 1] and ρ from [0.1, 0.5]. For our online filtering, the maximum number of CEP iterations was set to 50 and the tolerance level to 10^{-4}. For numerical stability, we re-scaled the timestamps to $[0, 1]$. We examined the number of factors (or factor trajectories) $R \in \{2, 3, 5, 7\}$.

6.5.2.2 Final Prediction Accuracy

We first examined the final prediction accuracy with our learned factor trajectories. To this end, we followed [101, 42], and randomly sampled 80% observed entry values and their timestamps for streaming inference and then tested the prediction error on the remaining entries. We also compared with the static decomposition methods, which need to repeatedly access the training entries. We repeated the experiment five times, and computed the average root mean-square-error (RMSE), average mean-absolute-error (MAE), and their standard deviations. We ran our method based on both the CP and Tucker forms, denoted by SFTL-CP and SFTL-Tucker, respectively.

We report the results for $R = 5$ in Table 6.1. As we can see, SFTL outperforms all the streaming approaches by a large margin. SFTL even obtains significantly better prediction accuracy than all the static decomposition approaches, except that on *Server Room*, SFTL is second to THIS-ODE and NONFAT. Note that SFTL only went through the training entries for once. Although NONFAT can also estimate the factor trajectories, it uses GPs to perform black-box nonlinear decomposition and hence loses the interpretability. Note that NONFAT in most case also outperforms the other static decomposition methods that only estimate time-invariant factors. The superior performance of SFTL and NONFAT shows the importance of capturing factor evolution.

6.5.2.3 Online Predictive Performance

N ext, we evaluated the online predictive performance of SFTL. Whenever a batch of entries at a new timestamp has been processed, we examined the prediction accuracy on the test set, with our current estimate of the factor trajectories. We repeated the evaluation for five times, and examine how the average prediction error varies along with the number of processed entries. We show the results for $R = 5$ in Fig. 6.3. It is clear that SFTL in most cases outperforms the competing streaming decomposition algorithms by a large margin

throughout the course of running. Note that the online behavior of BASS-Tucker was quite unstable and so we excluded it in the figures. It confirms the advantage of our streaming trajectory learning approach — even in the streaming scenario, incrementally capturing the time-variation of the factors can perform better than updating fixed, static factors.

6.5.2.4 Investigation of Learning Results

Finally, we investigated our learned factor trajectories. We set $R = 3$ and ran SFTL-CP on *ServerRoom*. In Fig. 6.4, we visualize the trajectory for the 1st air conditioning mode, the 2rd power usage level, and the 3rd location. The shaded region indicates the standard deviation, and the red dashed line the biggest timestamp in the data. First, we can see that the posterior variance of all the trajectories grows quickly when moving to the right of the red line. This is reasonable because it is getting far away from the training region. Second, for each object, the first and second trajectories (1st and 2nd column in Fig. 6.4) exhibit quite different time-varying patterns, *e.g.*, the local periodicity in $u^1_{1,2}(t)$ and $u^2_{2,2}(t)$, the gradual decreasing and increasing trends in $u^3_{3,1}(t)$ and $u^3_{3,2}(t)$, respectively, which imply different inner properties of the object. Third, it is particularly interesting to see that the third trajectory for all the objects appears to be close to zero, with relatively large posterior variance all the time. This might imply that two factor trajectories have been sufficient to represent each object. Requesting for a third trajectory is redundant. More important, our model is empirically able to detect such redundancy and returns a zero-valued trajectory.

6.6 Conclusion

We have presented SFTL, a probabilistic temporal tensor decomposition approach. SFTL can efficiently handle streaming data, and estimate time-varying factor representations. On four real-world applications, SFTL achieves superior online and final prediction accuracy.

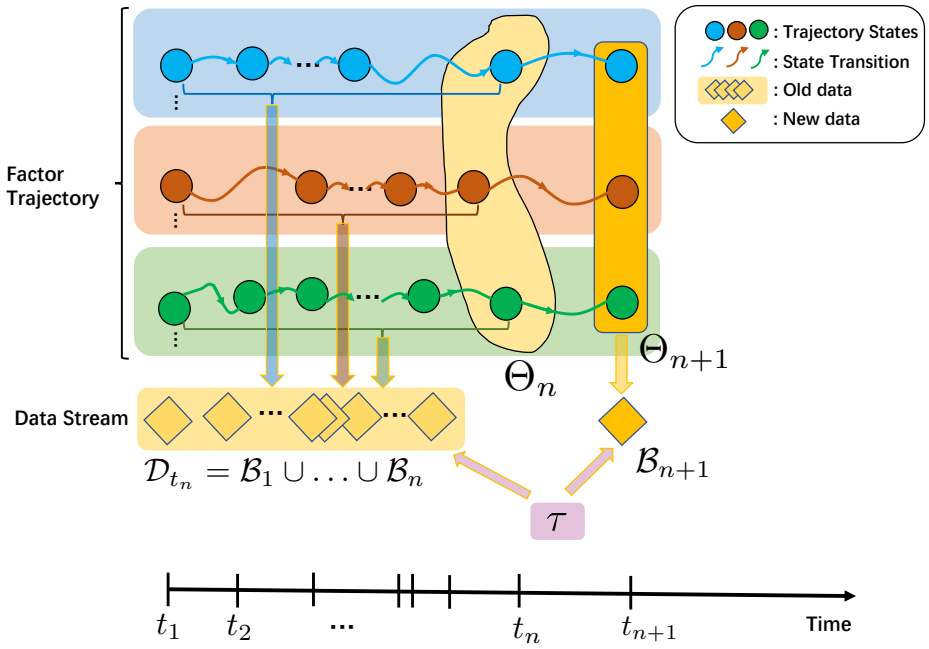
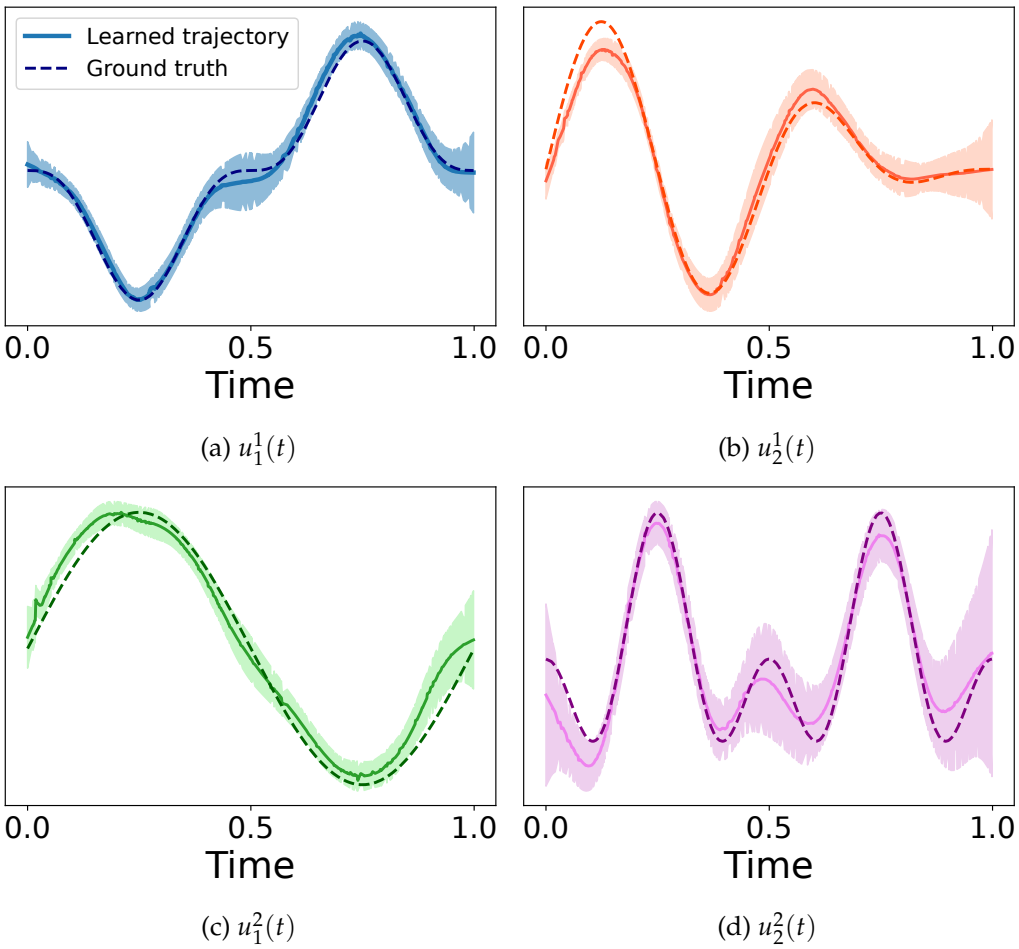


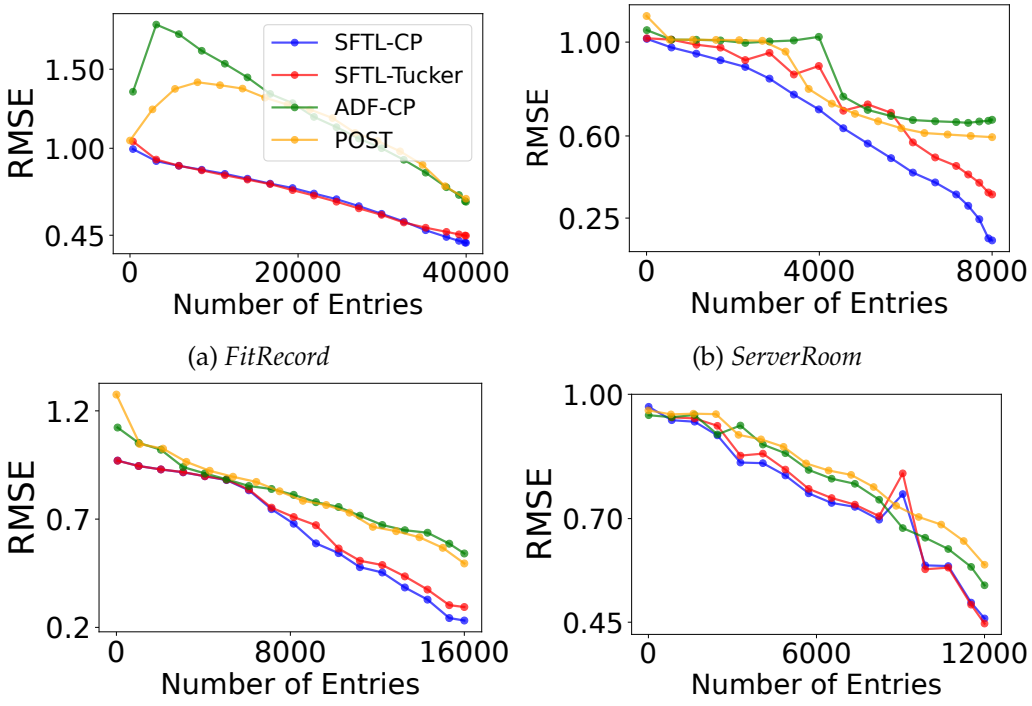
Figure 6.1: A graphical representation of our factor trajectory learning, from which we can see $\{\Theta_{n+1}, \mathcal{B}_{n+1}\}$ are independent to $\mathcal{D}_{t_n}$ conditioned on Θ_n and the noise inverse variance τ, namely, $\Theta_{n+1}, \mathcal{B}_{n+1} \perp \mathcal{D}_{t_n} | \Theta_n, \tau$.



(a) $u_1^1(t)$ (b) $u_2^1(t)$

(c) $u_1^2(t)$ (d) $u_2^2(t)$

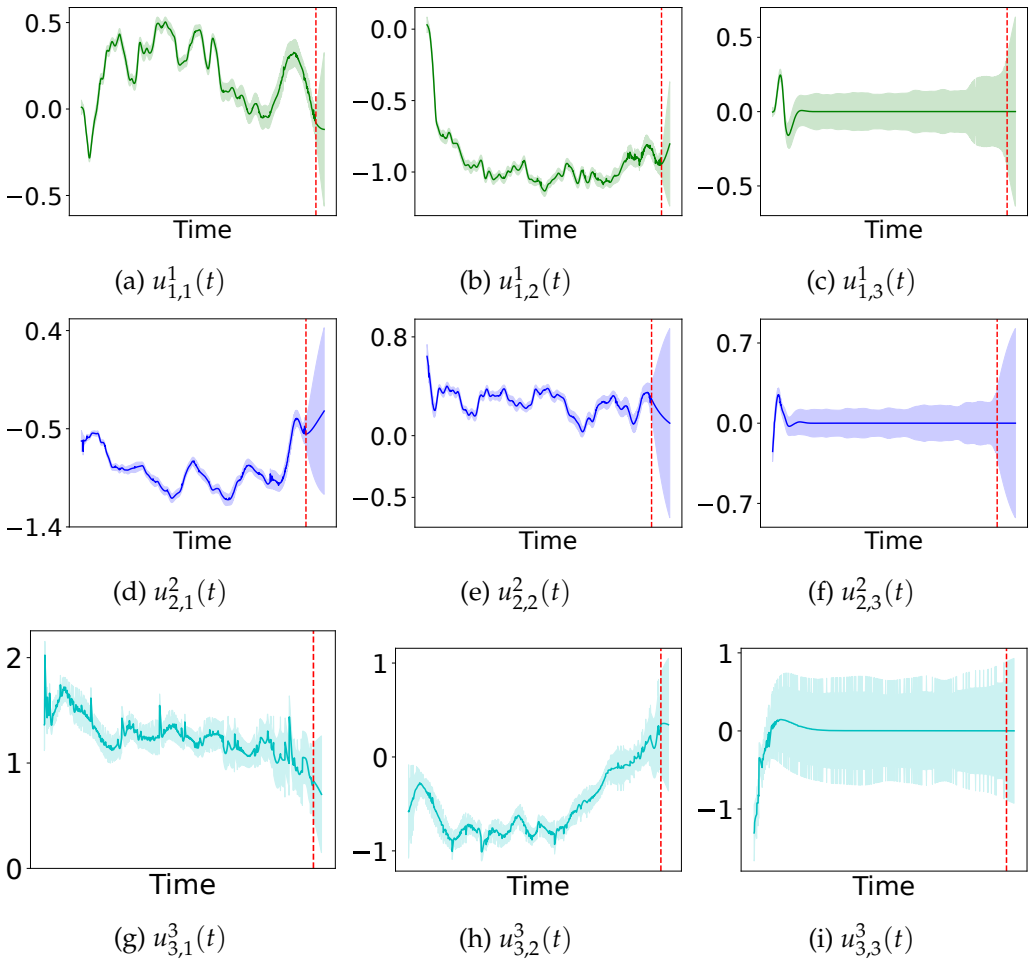
Figure 6.2: The learned factor trajectories from the synthetic data.



(a) *FitRecord* (b) *ServerRoom*

(c) *BeijingAir-2* (d) *BeijingAir-3*

Figure 6.3: Online prediction error with the number of processed entries ($R = 5$).



(a) $u^1_{1,1}(t)$ (b) $u^1_{1,2}(t)$ (c) $u^1_{1,3}(t)$

(d) $u^2_{2,1}(t)$ (e) $u^2_{2,2}(t)$ (f) $u^2_{2,3}(t)$

(g) $u^3_{3,1}(t)$ (h) $u^3_{3,2}(t)$ (i) $u^3_{3,3}(t)$

Figure 6.4: The learned factor trajectories of object 1, 2, 3 in mode 1, 2, 3, respectively, from *ServerRoom* data.

Table 6.1: Final prediction error with $R = 5$ averaged from five runs.

	RMSE	*FitRecord*	*ServerRoom*	*BeijingAir-2*	*BeijingAir-3*
Static	PTucker	0.656 ± 0.147	0.458 ± 0.039	0.401 ± 0.01	0.535 ± 0.062
	Tucker-ALS	0.846 ± 0.005	0.985 ± 0.014	0.559 ± 0.021	0.838 ± 0.026
	CP-ALS	0.882 ± 0.017	0.994 ± 0.015	0.801 ± 0.082	0.875 ± 0.028
	CT-CP	0.664 ± 0.007	0.384 ± 0.009	0.64 ± 0.007	0.815 ± 0.018
	CT-GP	0.604 ± 0.004	0.223 ± 0.035	0.759 ± 0.02	0.892 ± 0.026
	BCTT	0.518 ± 0.007	0.185 ± 0.013	0.396 ± 0.022	0.801 ± 0.02
	NONFAT	0.503 ± 0.002	$\mathbf{0.117 \pm 0.006}$	0.395 ± 0.007	0.882 ± 0.014
	THIS-ODE	0.526 ± 0.004	0.132 ± 0.003	0.54 ± 0.014	0.877 ± 0.026
Stream	POST	0.696 ± 0.019	0.64 ± 0.028	0.516 ± 0.028	0.658 ± 0.103
	ADF-CP	0.648 ± 0.008	0.654 ± 0.008	0.548 ± 0.015	0.551 ± 0.043
	BASS-Tucker	0.976 ± 0.024	1.000 ± 0.016	1.049 ± 0.037	0.991 ± 0.039
	SFTL-CP	$\mathbf{0.424 \pm 0.014}$	0.161 ± 0.014	$\mathbf{0.248 \pm 0.012}$	0.473 ± 0.013
	SFTL-Tucker	0.430 ± 0.010	0.331 ± 0.056	0.303 ± 0.041	$\mathbf{0.439 \pm 0.019}$
	MAE				
Static	PTucker	0.369 ± 0.009	0.259 ± 0.008	0.26 ± 0.006	0.263 ± 0.02
	Tucker-ALS	0.615 ± 0.006	0.739 ± 0.008	0.388 ± 0.008	0.631 ± 0.017
	CP-ALS	0.642 ± 0.012	0.746 ± 0.009	0.586 ± 0.056	0.655 ± 0.018
	CT-CP	0.46 ± 0.004	0.269 ± 0.003	0.489 ± 0.006	0.626 ± 0.01
	CT-GP	0.414 ± 0.001	0.165 ± 0.034	0.55 ± 0.012	0.626 ± 0.011
	BCTT	0.355 ± 0.005	0.141 ± 0.011	0.254 ± 0.007	0.578 ± 0.009
	NONFAT	0.341 ± 0.001	$\mathbf{0.071 \pm 0.004}$	0.256 ± 0.004	0.626 ± 0.007
	THIS-ODE	0.363 ± 0.004	0.083 ± 0.002	0.345 ± 0.004	0.605 ± 0.013
Stream	POST	0.478 ± 0.014	0.476 ± 0.023	0.352 ± 0.022	0.486 ± 0.095
	ADF-CP	0.449 ± 0.006	0.496 ± 0.007	0.385 ± 0.012	0.409 ± 0.029
	BASS	0.772 ± 0.031	0.749 ± 0.01	0.934 ± 0.037	0.731 ± 0.02
	SFTL-CP	$\mathbf{0.242 \pm 0.006}$	0.108 ± 0.008	$\mathbf{0.15 \pm 0.003}$	0.318 ± 0.008
	SFTL-Tucker	0.246 ± 0.001	0.216 ± 0.034	0.185 ± 0.029	$\mathbf{0.278 \pm 0.011}$

CHAPTER 7

FUNCTIONAL BAYESIAN TUCKER DECOMPOSITION

Tucker decomposition is a powerful tensor model to handle multi-aspect data. It demonstrates the low-rank property by decomposing the grid-structured data as interactions between a core tensor and a set of object representations (factors). A fundamental assumption of such decomposition is that there are finite objects in each aspect or mode, corresponding to discrete indexes of data entries. However, real-world data is often not naturally posed in this setting. For example, geographic data is represented as continuous indexes of latitude and longitude coordinates, and cannot fit tensor models directly. To generalize Tucker decomposition to such scenarios, we propose Functional Bayesian Tucker Decomposition (FunBaT). We treat the continuous-indexed data as the interaction between the Tucker core and a group of latent functions. We use Gaussian processes (GP) as functional priors to model the latent functions. Then, we convert each GP into a state-space prior by constructing an equivalent stochastic differential equation (SDE) to reduce computational cost. An efficient inference algorithm is developed for scalable posterior approximation based on advanced message-passing techniques. The advantage of our method is shown in both synthetic data and several real-world applications

7.1 Introduction

Tensor decomposition is widely used to analyze and predict with multi-way or multi-aspect data in real-world applications. For example, medical service records can be summarized by a four-mode tensor (*patients*, *doctor*, *clinics*, *time*) and the entry values can be the visit count or drug usage. Weather data can be abstracted by tensors like (*latitude*, *longitude*, *time*) and the entry values can be the temperature. Tensor decomposition introduces a set of low-rank representations of objects in each mode, known as factors, and estimates these factors to reconstruct the observed entry values. Among numerous proposed ten-

sor decomposition models, such as CANDECOMP/PARAFAC (CP) [32] decomposition and tensor train (TT) [62], Tucker decomposition [91] is widely-used for its fair capacity, flexibility and interpretability.

Despite its success, the standard tensor decomposition framework has a fundamental limitation. That is, it restricts the data format as structured grids, and demands each mode of the tensor must be discrete and finite-dimensional. That means, each mode includes a finite set of objects, indexed by integers, such as user 1, user 2, ..., and each entry value is located by the tuple of discrete indexes. However, in many real-world applications like climate and geographic modeling, the data is represented as continuous coordinates of modes, *e.g., latitude*, *longitude* and *time*. In these cases, each data entry is located by a tuple of continuous indexes, and tensor decomposition cannot be applied directly.

To enable tensor models on such data, one widely-used trick is to discretize the modes, for example, binning the timestamps into time steps (say, by weeks or months), or binning the latitude values into a set of ranges, and number the ranges by 1, 2, 3... While doing this is feasible, the fine-grained information is lost, and it is hard to decide an appropriate discretization strategy in many cases [65]. A more natural solution is to extend tensor models from grid-structure to continuous field. To the best of our knowledge, most existing methods in that direction are limited to tensor-train [31, 7, 5, 14] or the simpler CP format [79], and cannot be applied to Tucker decomposition, an important compact and flexible low-rank representation. What's more, most of the proposed methods are deterministic with polynomial approximations, cannot provide probabilistic inference, and are hard to well handle noisy, incomplete data.

To bridge the gap, we propose FunBaT: Functional Bayesian Tucker decomposition, which generalizes the standard Tucker decomposition to the functional field under the probabilistic framework. We decompose the continuous-indexed tensor data as the interaction between the Tucker core and a group of latent functions, which map the continuous indexes to mode-wise factors. We approximate such functions by Gaussian processes (GPs), and follow [33] to convert them into state-space priors by constructing equivalent stochastic differential equations (SDE) to reduce computational cost. We then develop an efficient inference algorithm based on the conditional expectation propagation (CEP) framework [96] and Bayesian filter to approximate the posterior distribution of the factors. We show

this framework can also be used to extend CP decomposition to continuous-indexed data with a simplified setting. For evaluation, we conducted experiments on both synthetic data and real-world applications of climate modeling. The results show that FunBaT not only outperforms the state-of-the-art methods by large margins in terms of reconstruction error, but also identifies interpretable patterns that align with domain knowledge.

7.2 Function Factorization and State-Space GP

7.2.1 Function Factorization

Function factorization refers to decomposing a complex multivariate function into a set of low-dimensional functions [68, 58]. The most important and widely used tensor method for function factorization is TT [31, 7, 5]. It converts the function approximation into a tensor decomposition problem. A canonical form of applying TT to factorize a multivariate function $f(\mathbf{x})$ with K variables $\mathbf{x} = (x_1, \dots x_K) \in \mathbb{R}^K$ is:

$$f(\mathbf{x}) \approx G_1(x_1) \times G_2(x_2) \times \ldots \times G_K(x_K), \tag{7.1}$$

where each $G_k(\cdot)$ is a univariate matrix-valued function, takes the scaler x_k as the input and output a matrix $G_k(x_k) \in \mathbb{R}^{r_{k-1} \times r_k}$. It is straightforward to see that $G_k(\cdot)$ is a continuous generalization of the factor $\mathbf{u}_j^k$ in standard TT decomposition.

7.2.2 Gaussian Process as State Space Model

Gaussian process (GP) [71] is a powerful non-parametric Bayesian model to approximate functions. Formally, a GP is a prior distribution over function $f(\cdot)$, such that $f(\cdot)$ evaluated at any finite set of points $\mathbf{x} = \{x_1, \dots, x_N\}$ follows a multivariate Gaussian distribution, denoted as $f \sim \mathcal{GP}(0, \kappa(x, x'))$, where $\kappa(x, x')$ is the covariance function, also known as the kernel, measuring the similarity between two points. One of the most widely used kernels is the Matérn kernel:

$$\kappa_{\text{Matérn}} = \sigma^2 \frac{\left(\frac{\sqrt{2\nu}}{\ell}||x - x'||^2\right)^{\nu}}{\Gamma(\nu)2^{\nu-1}} K_\nu \left(\frac{\sqrt{2\nu}}{\ell}||x - x'||^2\right), \tag{7.2}$$

where $\{\sigma^2, \ell, \nu, p\}$ are hyperparameters determining the variance, length-scale, smoothness, and periodicity of the function, K_ν is the modified Bessel function, and $\Gamma(\cdot)$ is the Gamma function.

Despite the great capacity of GP, it suffers from cubic scaling complexity $O(N^3)$ for inference. To overcome this limitation, recent work [33] used spectral analysis to show an equivalence between GPs with stationary kernels and linear time-invariant stochastic differential equations (LTI-SDEs). Specifically, we can formulate a vector-valued latent state $\mathbf{z}(x)$ comprising $f(x)$ and its derivatives up to m-th order. The GP $f(x) \sim \mathcal{GP}(0, \kappa)$ is equivalent to the solution of an LTI-SDE defined as:

$$\mathbf{z}(x) = \left(f(x), \frac{\mathrm{d}f(x)}{\mathrm{d}x}, \ldots, \frac{\mathrm{d}f^m(x)}{\mathrm{d}x} \right)^\top, \ \frac{\mathrm{d}\mathbf{z}(x)}{\mathrm{d}x} = \mathbf{F}\mathbf{z}(x) + \mathbf{L}w(x), \tag{7.3}$$

where $\mathbf{F}$ and $\mathbf{L}$ are time-invariant coefficients, and $w(x)$ is the white noise process with density q_s. At any finite collection of points $x_1 < \ldots < x_N$, the SDE in (7.3) can be further discretized as a Gaussian-Markov chain, also known as the state-space model, defined as:

$$p(\mathbf{z}(x)) = p(\mathbf{z}_1) \prod_{n=1}^{N-1} p(\mathbf{z}_{n+1}|\mathbf{z}_n) = \mathcal{N}(\mathbf{z}(x_1)|\mathbf{0}, \mathbf{P}_\infty) \prod_{n=1}^{N-1} \mathcal{N}(\mathbf{z}(x_{n+1})|\mathbf{A}_n\mathbf{z}(x_n), \mathbf{Q}_n), \tag{7.4}$$

where $\mathbf{A}_n = \exp(\mathbf{F}\Delta_n)$, $\mathbf{Q}_n = \int_{t_n}^{t_{n+1}} \mathbf{A}_n \mathbf{L}\mathbf{L}^\top \mathbf{A}_n^\top q_\mathbf{s} \mathrm{d}t$, $\Delta_n = x_{n+1} - x_n$, and $\mathbf{P}_\infty$ is the steady-state covariance matrix which can be obtained by solving the Lyapunov equation [46]. All the above parameters in (7.3) and (7.4) are fully determined by the kernel κ and the time interval Δ_n. For the Matérn kernel (7.2) with smoothness ν being an integer plus a half, $\mathbf{F}$, $\mathbf{L}$ and $\mathbf{P}_\infty$ possess closed forms [74]. Specifically, when $\nu = 1/2$, we have $\{m = 0, \mathbf{F} = -1/\ell, \mathbf{L} = 1, q_\mathbf{s} = 2\sigma^2/\ell, \mathbf{P}_\infty = \sigma^2\}$; for $\nu = 3/2$, we have $m = 1, \mathbf{F} = (0, 1; -\lambda^2, -2\lambda), \mathbf{L} = (0; 1), q_\mathbf{s} = 4\sigma^2\lambda^3, \mathbf{P}_\infty = (\sigma^2, 0; 0, \lambda^2\sigma^2)$, where $\lambda = \sqrt{3}/\ell$. With the state space prior, efficient $O(n)$ inference can be achieved by using classical Bayesian sequential inference techniques, like Kalman filtering and RTS smoothing [33]. Then the original function $f(x)$ is simply the first element of the inferred latent state $\mathbf{z}(x)$.

7.3 Model

7.3.1 Functional Tucker Decomposition with Gaussian Process

Despite the successes of tensor models, the standard tensor models are unsuitable for continuous-indexed data, such as the climate data with modes *latitude*, *longitude* and *time*. The continuity property encoded in the real-value indexes will be dropped while applying discretization. Additionally, it can be challenging to determine the optimal discretization strategy, and the trained model cannot handle new objects with never-seen

indexes. The function factorization idea with tensor structure is therefore a more natural solution to this problem. However, the early work [79] with the simplest CP model, uses sampling-based inference, which is limited in capacity and not scalable to large data. The TT-based function factorization methods [31, 7, 5, 14] take deterministic approximation like Chebyshev polynomials or splines, which is hard to handle data noises or provide uncertainty quantification.

Compared to CP and TT, Tucker decomposition possesses compact and flexible low-rank representation. The Tucker core can capture more global and interpretable patterns of the data. Thus, we aim to extend Tucker decomposition to continuous-indexed data to fully utilize its advantages as a Bayesian method, and propose FunBaT: Functional Bayesian Tucker decomposition.

Aligned with the setting of the function factorization (7.1), we factorize a K-variate function $f(\mathbf{i})$ in Tucker format (2.2) with preset latent rank:$\{r_1, \dots r_K\}$:

$$f(\mathbf{i}) = f(i_1, \dots i_K) \approx \text{vec}(\mathcal{W})^\top \left(\mathbf{U}^1(i_1) \otimes \dots \otimes \mathbf{U}^K(i_K) \right), \tag{7.5}$$

where $\mathbf{i} = (i_1, \dots i_K) \in \mathbb{R}^K$, and $\mathcal{W} \in \mathbb{R}^{r_1 \times \dots \times r_K}$ is the Tucker core, which is the same as the standard Tucker decomposition. However, $\mathbf{U}^k(\cdot)$ is a r_k-size vector-valued function, mapping the continuous index i_k of mode k to a r_k-dimensional latent factor. We assign independent GP priors over each output dimension of $\mathbf{U}^k(\cdot)$, and model $\mathbf{U}^k(\cdot)$ as the stack of a group of univariate scalar functions. Specifically, we have:

$$\mathbf{U}^k(i_k) = [u_1^k(i_k), \dots, u_{r_k}^k(i_k)]^T;\ u_j^k(i_k) \sim \mathcal{GP}\left(0, \kappa(i_k, i_k')\right), j = 1 \dots r_k \tag{7.6}$$

where $\kappa(i_k, i_k') : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is the covariance (kernel) function of the k-th mode. In this work, we use the popular Matérn kernel (7.2) for GPs over all modes.

7.3.2 State-Space-Prior and Joint Probabilities

Given N observed entries of a K-mode continuous-indexed tensor $\mathcal{Y}$, denoted as $\mathcal{D} = \{(\mathbf{i}^n, y_n)\}_{n=1\dots N}$, where $\mathbf{i}^n = (i_1^n \dots i_K^n)$ is the tuple of continuous indexes, and y_n is the entry values, we can assume all the observations are sampled from the target Tucker-format function $f(\mathbf{i})$ and Gaussian noise τ^{-1}. Specifically, we define the likelihood l_n as:

$$l_n \triangleq p(y_n | \{\mathbf{U}^k\}_{k=1}^K, \mathcal{W}, \tau) = \mathcal{N}\left(y_n \mid \text{vec}(\mathcal{W})^\top \left(\mathbf{U}^1(i_1^n) \otimes \dots \otimes \mathbf{U}^K(i_K^n) \right), \tau^{-1} \right). \tag{7.7}$$

We then handle the functional priors over the $\{\mathbf{U}^k\}_{k=1}^K$. As introduced in Section 7.2.2, the dimension-wise GP with Matérn kernel over $u_j^k(i_k)$ is equivalent to an M-order LTI-SDE (7.3), and then we can discrete it as a state space model (7.4). Thus, for each mode's function $\mathbf{U}^k$, we concatenate the state space representation over all r_k dimensions of $\mathbf{U}^k(i_k)$ to a joint state variable $\mathbf{Z}^k(i_k) = \text{concat}[\mathbf{z}_1^k(i_k), \ldots, \mathbf{z}_{r_k}^k(i_k)]$ for $\mathbf{U}^k$, where $\mathbf{z}_j^k(i_k)$ is the state variable of $u_j^k(i_k)$. We can build an ordered index set $\mathcal{I}_k = \{i_k^1 \ldots i_k^{N_k}\}$, which includes the N_k unique indexes of mode-k's among all observed entries in $\mathcal{D}$. Then, the state space prior over $\mathbf{U}^k$ on $\mathcal{I}_k$ is:

$$p(\mathbf{U}^k) = p(\mathbf{Z}^k) = p(\mathbf{Z}^k(i_k^1), \ldots, \mathbf{Z}^k(i_k^{N_k})) = p(\mathbf{Z}_1^k) \prod_{s=1}^{N_k-1} p(\mathbf{Z}_{s+1}^k | \mathbf{Z}_s^k), \tag{7.8}$$

where

$$p(\mathbf{Z}_1^k) = \mathcal{N}(\mathbf{Z}^k(i_k^1) | \mathbf{0}, \tilde{\mathbf{P}}_\infty^k);\ p(\mathbf{Z}_{s+1}^k | \mathbf{Z}_s^k) = \mathcal{N}(\mathbf{Z}^k(i_k^{s+1}) | \tilde{\mathbf{A}}_s^k \mathbf{Z}^k(i_k^s), \tilde{\mathbf{Q}}_s^k). \tag{7.9}$$

The parameters in (7.9) are block-diagonal concatenates of the corresponding univariate case. Namely, $\tilde{\mathbf{P}}_\infty^k = \text{BlockDiag}(\mathbf{P}_\infty^k \ldots \mathbf{P}_\infty^k)$, $\tilde{\mathbf{A}}_s^k = \text{BlockDiag}(\mathbf{A}_s^k \ldots \mathbf{A}_s^k)$, and $\tilde{\mathbf{Q}}_s^k = \text{BlockDiag}(\mathbf{Q}_s^k \ldots \mathbf{Q}_s^k)$, where $\mathbf{P}_\infty^k$, $\mathbf{A}_s^k$ and $\mathbf{Q}_s^k$ are the corresponding parameters in (7.4). With $\mathbf{Z}^k(i_k)$, we can fetch the value of $\mathbf{U}^k$ by multiplying with a projection matrix $\mathbf{H} = \text{BlockDiag}([1, 0, \ldots], \ldots [1, 0, \ldots])$: $\mathbf{U}^k = \mathbf{H}\mathbf{Z}^k$.

With state space priors of the latent functions, we further assign a Gamma prior over the noise precision τ and a Gaussian prior over the Tucker core. For compact notation, we denote the set of all random variables as $\Theta = \{\mathcal{W}, \tau, \{\mathbf{Z}^k\}_{k=1}^K\}$. Finally, the joint probability of the proposed model is:

$$p(\mathcal{D}, \Theta) = p(\mathcal{D}, \{\mathbf{Z}^k\}_{k=1}^K, \mathcal{W}, \tau) = p(\tau) p(\mathcal{W}) \prod_{k=1}^K [p(\mathbf{Z}_1^k) \prod_{s=1}^{N_k-1} p(\mathbf{Z}_{s+1}^k | \mathbf{Z}_s^k)] \prod_{n=1}^N l_n, \tag{7.10}$$

where $p(\mathcal{W}) = \mathcal{N}(\text{vec}(\mathcal{W}) \mid \mathbf{0}, \mathbf{I})$, $p(\tau) = \text{Gam}(\tau | a_0, b_0)$, a_0 and b_0 are the hyperparameters of the Gamma prior, and l_n is the data likelihood defined in (7.7).

7.4 Algorithm

The inference of the exact posterior $p(\Theta|\mathcal{D})$ with (7.10) is intractable, as there are multiple latent functions $\{\mathbf{U}^k\}_{k=1}^K$ and the Tucker core interleave together in a complex manner in the likelihood term l_n. To address this issue, we propose an efficient approximation

algorithm, which first decouples the likelihood term by a factorized approximation, and then applies the sequential inference of Kalman filter and RTS smoother to infer the latent variables $\{\mathbf{Z}^k\}_{k=1}^K$ at each observed index. Finally, we employ the conditional moment matching technique to update the message factors in parallel. We will introduce the details in the following subsections.

7.4.1 Factorized Approximation with Gaussian and Gamma Distribution

To estimate the intractable posterior $p(\Theta|\mathcal{D})$ with a tractable $q(\Theta)$, we first apply the mean-field assumption and design the approximated posterior as a fully factorized format. Specifically, we approximate the posterior as:

$$p(\Theta|\mathcal{D}) \approx q(\Theta) = q(\tau)q(\mathcal{W})\prod_{k=1}^{K} q(\mathbf{Z}^k), \tag{7.11}$$

where $q(\tau) = \text{Gam}(\tau|a,b)$, $q(\mathcal{W}) = \mathcal{N}(\text{vec}(\mathcal{W}) \mid \boldsymbol{\mu}, \mathbf{S})$ are the approximated posterior of τ and $\mathcal{W}$, respectively. For $q(\mathbf{Z}^k)$, we further decompose it over the observed indexes set $\mathcal{I}_k$ as $q(\mathbf{Z}^k) = \prod_{s=1}^{N_k} q(\mathbf{Z}_s^k)$, where $q(\mathbf{Z}_s^k) = q(\mathbf{Z}^k(i_s^k)) = \mathcal{N}(\mathbf{Z}_s^k \mid \mathbf{m}_s^k, \mathbf{V}_s^k)$. Our goal is to estimate the variational parameters $\{a, b, \boldsymbol{\mu}, \mathbf{S}, \{\mathbf{m}_s^k, \mathbf{V}_s^k\}\}$, and make $q(\Theta)$ close to $p(\Theta|\mathcal{D})$.

To do so, we use Expectation Propagation (EP) [55], to update $q(\Theta)$. However, the standard EP cannot work because the complex Tucker form of the likelihood term l_n makes it intractable to compute the expectation of the likelihood term l_n under $q(\Theta)$. Thus, we use the advanced moment-matching technique, Conditional Expectation Propagation (CEP) [96] to address this issue. With CEP, we employ a factorized approximation to decouple the likelihood term l_n into a group of message factors $\{f_n\}$:

$$\mathcal{N}(y_n \mid \text{vec}(\mathcal{W})^\top \left(\mathbf{U}^1(i_1^n) \otimes \ldots \otimes \mathbf{U}^K(i_K^n)\right), \tau^{-1}) \approx Z_n f_n(\tau) f_n(\mathcal{W}) \prod_{k=1}^{K} f_n(\mathbf{Z}^k(i_k^n)), \tag{7.12}$$

where Z_n is the normalized constant, $f_n(\tau) = \text{Gam}(\tau|a_n, b_n)$, $f_n(\mathcal{W}) = \mathcal{N}(\text{vec}(\mathcal{W}) \mid \boldsymbol{\mu}_n, \mathbf{S}_n)$, $f_n(\mathbf{Z}^k(i_k^n)) = \mathcal{N}(\mathbf{Z}^k(i_k^n)|\mathbf{m}_n^k, \mathbf{V}_n^k)$ are the message factors obtained from the conditional moment-matching of $\mathbf{E}_q[l_n]$. Then we update the posterior $q(\tau)$ and $q(\mathcal{W})$ by simply merging the message factors from likelihood (7.12) and priors:

$$q(\tau) = p(\tau)\prod_{n=1}^{N} f_n(\tau) = \text{Gam}(\tau|a_0, b_0)\prod_{n=1}^{N} \text{Gam}(\tau|a_n, b_n), \tag{7.13}$$

$$q(\mathcal{W}) = p(\mathcal{W})\prod_{n=1}^{N} f_n(\mathcal{W}) = \mathcal{N}(\text{vec}(\mathcal{W}) \mid \mathbf{0}, \mathbf{I})\prod_{n=1}^{N} \mathcal{N}(\text{vec}(\mathcal{W}) \mid \boldsymbol{\mu}_n, \mathbf{S}_n). \tag{7.14}$$

The message approximation in (7.12) and message merging (7.13)(7.14) are based on the conditional moment-matching technique and the property of exponential family presented in [96]. All the involved computation is closed-form and can be conducted in parallel.

7.4.2 Sequential State Inference with Bayesian Filter and Smoother

Handling $q(\mathbf{Z}^k)$ is a bit more challenging. It is because the prior of $\mathbf{Z}^k$ has a chain structure (7.8)(7.9), and we need to handle complicated integration over the whole chain to obtain marginal posterior $q(\mathbf{Z}^k_s)$ in the standard inference. However, with the classical Bayesian filter and smoother method, we can infer the posterior efficiently. Specifically, the state space structure over $\mathbf{Z}^k_s$ is:

$$q(\mathbf{Z}^k_s) = q(\mathbf{Z}^k_{s-1})p(\mathbf{Z}^k_s|\mathbf{Z}^k_{s-1})\prod_{n\in\mathcal{D}^k_s} f_n(\mathbf{Z}^k_s), \tag{7.15}$$

where $p(\mathbf{Z}^k_s|\mathbf{Z}^k_{s-1})$ is the transitions of the state given in (7.9), and $\mathcal{D}^k_s$ is the subset of $\mathcal{D}$, including the observation entries whose k-mode index is i^s_k, namely, $\mathcal{D}^k_s = \{n : i^n_k = i^s_k \mid \mathbf{i}_n \in \mathcal{D}\}$. If we treat $\prod_{n\in\mathcal{D}^k_s} f_n(\mathbf{Z}^k_s)$, a group of Gaussian message factors, as the observation of the state space model, (7.15) is the standard Kalman Filter(KF) [41]. Thus, we can run the KF algorithm and compute $q(\mathbf{Z}^k_s)$ from $s = 1$ to N_k sequentially. After the forward pass over the states, we can run RTS smoothing [72] as a backward pass to compute the global posterior of $q(\mathbf{Z}^k_s)$. This sequential inference is widely used to infer state space GP models [33, 74].

The whole inference algorithm is organized as follows: with the observation $\mathcal{D}$, we initialize the approximated posteriors and message factors for each likelihood. Then for each mode, we firstly approximate the message factors by CEP in parallel, and then merge them to update $q(\tau), q(\mathcal{W})$. We run the KF and RTS smoother to infer $q(\mathbf{Z}^k)$ at each observed index by treating the message factors as observations. We repeat the inference until convergence. We summarize the algorithm in Algorithm 5.

7.4.2.1 Algorithm Complexity

The overall time complexity is $\mathcal{O}(NKR)$, where K is the number of modes, N is the number of observations and R is pre-set mode rank. The space complexity is $\mathcal{O}(NK(R + R^2))$, as we need to store all the message factors. The linear time and space complexity w.r.t

Algorithm 5 FunBaT

Input: Observations $\mathcal{D}$ of a K-mode continuous-indexed tensor , kernel hyperparameters, sorted unique indexes set $\{\mathcal{I}_k\}$ of each mode.
Initialize approx. posterior $q(\tau)$, $q(\mathcal{W})$, $\{q(\mathbf{Z}^k)\}$ and message factors for each likelihood.
repeat
  for $k = 1$ **to** K **do**
    Approximate messages factors $\{f_n(\mathbf{Z}^k(i_k^n)), f_n(\tau), f_n(\mathcal{W})\}_{n=1}^N$ in parallel with CEP (7.12)
    Update the approximated posterior $q(\tau)$, $q(\mathcal{W})$ by merging the message (7.13)(7.14).
    Update $q(\mathbf{Z}^k)$ sequentially by KF and RTS smoother based on (7.15)
  end for
until Convergence
Return: $q(\tau), q(\mathcal{W}), \{q(\mathbf{Z}^k)\}_{k=1}^K$

both data size and tensor mode show the promising scalability of our algorithm.

7.4.2.2 Probabilistic Interpolation at Arbitrary Index

Although the state-space functional prior of $\mathbf{U}^k$ or $\mathbf{Z}^k$ is defined over finite observed index set $\mathcal{I}_k$, we highlight that we can handle the probabilistic interpolation of $\mathbf{Z}^k$ at arbitrary index $i_k^* \notin \mathcal{I}_k$ after model inference. Specifically, with $i_k^{s-1} < i_k^* < i_k^s$ we can infer the $q(\mathbf{Z}^k(i_k^*))$ by integrating the messages from the transitions of its neighbors and will obtain a closed-form solution:

$$q(\mathbf{Z}^k(i_k^*)) = \int q(\mathbf{Z}_{s-1}^k) p(\mathbf{Z}^k(i_k^*)|\mathbf{Z}_{s-1}^k) \mathrm{d}\mathbf{Z}_{s-1}^k \int q(\mathbf{Z}_s^k) p\left(\mathbf{Z}_s^k|\mathbf{Z}^k(i_k^*)\right) \mathrm{d}\mathbf{Z}_s^k = \mathcal{N}(\mathbf{m}^*, \mathbf{V}^*). \tag{7.16}$$

We give the detailed derivation in the following paragraphs. This enables us to build a continuous trajectory for each mode, and predict the tensor value at any indexes, for which standard discrete-mode tensor decomposition cannot do.

To derive the probabilistic imputation equation (7.16), we consider a general state space model, which includes a sequence of states $\mathbf{x}_1, \ldots, \mathbf{x}_M$ and the observed data $\mathcal{D}$. The states are at time $t_1, \ldots, t_M$ respectively. The key of the state space model is that the prior of the states is a Markov chain. The joint probability has the following form,

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_M, \mathcal{D}) = p(\mathbf{x}_1) \prod_{j=1}^{M-1} p(\mathbf{x}_{j+1}|\mathbf{x}_j) \cdot p(\mathcal{D}|\mathbf{x}_1, \ldots, \mathbf{x}_M). \tag{7.17}$$

Note that here we do not assume the data likelihood is factorized over each state, like those

typically used in Kalman filtering. In our point process model, the likelihood often couples multiple states together.

Suppose we have run some posterior inference to obtain the posterior of these states $q(\mathbf{x}_1, \ldots, \mathbf{x}_M)$, and we can easily pick up the marginal posterior of each state and each pair of the states. Now we want to calculate the posterior distribution of the state at time t^* such that $t_m < t^* < t_{m+1}$. Denote the corresponding state by $\mathbf{x}^*$, our goal is to compute $p(\mathbf{x}^*|\mathcal{D})$. To do so, we consider incorporating $\mathbf{x}^*$ in the joint probability (7.17),

$$\begin{aligned} &p(\mathbf{x}_1, \ldots, \mathbf{x}_m, \mathbf{x}^*, \mathbf{x}_{m+1}, \ldots, \mathbf{x}_M, \mathcal{D}) \\ &= p(\mathbf{x}_1) \prod_{j=1}^{m-1} p(\mathbf{x}_{j+1}|\mathbf{x}_j) \cdot p(\mathbf{x}^*|\mathbf{x}_m) p(\mathbf{x}_{m+1}|\mathbf{x}^*) \cdot \prod_{j=m+1}^{M} p(\mathbf{x}_{j+1}|\mathbf{x}_j) \cdot p(\mathcal{D}|\mathbf{x}_1, \ldots, \mathbf{x}_M). \end{aligned} \tag{7.18}$$

Now, we marginalize out $\mathbf{x}_{1:M\setminus\{m,m+1\}} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{m-1}, \mathbf{x}_{m+2}, \ldots, \mathbf{x}_M\}$. Note that since $\mathbf{x}^*$ does not appear in the likelihood, we can take it out from the integral,

$$\begin{aligned} &p(\mathbf{x}_m, \mathbf{x}_{m+1}, \mathbf{x}^*, \mathcal{D}) \\ &= \int p(\mathbf{x}_1) \prod_{j=1}^{m-1} p(\mathbf{x}_{j+1}|\mathbf{x}_j) \prod_{j=m+1}^{M} p(\mathbf{x}_{j+1}|\mathbf{x}_j) \cdot p(\mathcal{D}|\mathbf{x}_1, \ldots, \mathbf{x}_M) \mathrm{d}\mathbf{x}_{1:M\setminus\{m,m+1\}} p(\mathbf{x}^*|\mathbf{x}_m) p(\mathbf{x}_{m+1}|\mathbf{x}^*) \\ &= \frac{p(\mathbf{x}_m, \mathbf{x}_{m+1}, \mathcal{D}) p(\mathbf{x}^*|\mathbf{x}_m) p(\mathbf{x}_{m+1}|\mathbf{x}^*)}{p(\mathbf{x}_{m+1}|\mathbf{x}_m)}. \end{aligned} \tag{7.19}$$

Therefore, we have

$$p(\mathbf{x}_m, \mathbf{x}_{m+1}, \mathbf{x}^*|\mathcal{D}) \propto p(\mathbf{x}_m, \mathbf{x}_{m+1}|\mathcal{D}) p(\mathbf{x}^*|\mathbf{x}_m) p(\mathbf{x}_{m+1}|\mathbf{x}^*). \tag{7.20}$$

Suppose we are able to obtain $p(\mathbf{x}_m, \mathbf{x}_{m+1}|\mathcal{D}) \approx q(\mathbf{x}_m, \mathbf{x}_{m+1})$. We now need to obtain the posterior of $\mathbf{x}^*$. In the LTI SDE model, we know that the state transition is a Gaussian jump. Let us denote

$$p(\mathbf{x}^*|\mathbf{x}_m) = \mathcal{N}(\mathbf{x}^*|\mathbf{A}_1\mathbf{x}_m, \mathbf{Q}_1), \quad p(\mathbf{x}_{m+1}|\mathbf{x}_*) = \mathcal{N}(\mathbf{x}_{m+1}|\mathbf{A}_2\mathbf{x}^*, \mathbf{Q}_2).$$

We can simply merge the natural parameters of the two Gaussian and obtain

$$p(\mathbf{x}_m, \mathbf{x}_{m+1}, \mathbf{x}^*|\mathcal{D}) = p(\mathbf{x}_m, \mathbf{x}_{m+1}|\mathcal{D}) \mathcal{N}(\mathbf{x}^*|\mathbf{m}^*, \mathbf{V}^*), \tag{7.21}$$

where

$$\begin{aligned} (\mathbf{V}^*)^{-1} &= \mathbf{Q}_1^{-1} + \mathbf{A}_2^\top \mathbf{Q}_2^{-1} \mathbf{A}_2, \\ (\mathbf{V}^*)^{-1} \mathbf{m}^* &= \mathbf{Q}_1^{-1} \mathbf{A}_1 \mathbf{x}_m + \mathbf{A}_2^\top \mathbf{Q}_2^{-1} \mathbf{x}_{m+1}. \end{aligned} \tag{7.22}$$

7.4.2.3 Lightweight Alternative: FunBaT-CP

As the CP decomposition is a special and simplified case of Tucker decomposition, it is straightforward to build a functional CP decomposition with the proposed model and algorithm. Specifically, we only need to set the Tucker core $\mathcal{W}$ as all-zero constant tensor except diagonal elements as one, skip the inference step of $\mathcal{W}$, and perform the remaining steps. We then achieve FunBaT-CP, a simple and efficient functional Bayesian CP model. In some experiments, we found FunBaT-CP is more robust than FunBaT, as the dense Tucker core takes more parameters and is easier to get overfitting. We will show it in the experiment section.

7.5 Related Work

The early work to apply tensor format to function factorization is [79]. It takes the wrapped-GP to factorize the function with CP format and infers with Monte Carlo sampling, which is not scalable to large data. Applying tensor-train(TT) to approximate the multivariate function with low-rank structure is a popular topic [31, 7, 5, 13, 14], with typical applications in simultaneous localization and mapping. These methods mainly use Chebyshev polynomials and splines as function basis, and rely on complex optimization methods like alternating updates, cross-interpolation [31, 7] and ANOVA (analysis of variance) representation [5, 14]. Despite the compact form, they are purely deterministic, sensitive to initial values and data noises, and cannot provide probabilistic inference. Similar techniques like state-space GP and CEP in recent work [24, 27], but they are designed to capture temporal information in tensor data.

7.6 Experiment

7.6.1 Synthetic Data

We first evaluated FunBaT on a synthetic task by simulating a rank-1 two-mode tensor with each mode designed as a continuous function:

$$\mathbf{U}^1(i_1) = \exp(-2i_1) \cdot \sin(\frac{3}{2}\pi i_1); \mathbf{U}^2(i_2) = \sin^2(2\pi i_2) \cdot \cos(2\pi i_2). \tag{7.23}$$

The ground truth of the continuous-mode tensor, represented as a surface in Fig. 7.1a, was obtained by: $y_{\mathbf{i}} = \mathbf{U}^1(i_1)\mathbf{U}^1(i_2)$. We randomly sampled 650 indexes entries from $[0,1] \times [0,1]$ and added Gaussian noise $\epsilon \sim \mathcal{N}(0, 0.02)$ as the observations. We used

PyTorch to implement FunBaT, which used the Matérn kernel with $\nu = 3/2$, and set $l = 0.1$ and $\sigma^2 = 1$. We trained the model with $R = 1$ and compared the learned continuous-mode functions with their ground truth. We used the learned factors and the interpolation (7.16) to reconstruct the whole tensor surface, shown in Fig. 7.1b. Our learned trajectories, shown in Fig. 7.1c and d, clearly revealed the real mode functions. The shaded area is the estimated standard deviation.

7.6.2 Real-World Applications

7.6.2.1 Datasets

We evaluated FunBaT on four real-world datasets: *BeijingAir-PM2.5*, *BeijingAir-PM10*, *BeijingAir-SO2* and *US-TEMP*. The first three are extracted from BeijingAir[1], which contain hourly measurements of several air pollutants with weather features in Beijing from 2014 to 2017. We selected three continuous-indexed modes: *(atmospheric-pressure, temperature, time)* and processed it into three tensors by using different pollutants (PM2.5, PM10, SO2). Each dataset contains $17K$ observations across unique indexes of 428 atmospheric pressure measurements, 501 temperature measurements, and 1461 timestamps. We obtain *US-TEMP* from the ClimateChange[2]. The dataset contains temperatures of cities world-wide and geospatial features. We selected temperature data from 248 cities in the United States from 1750 to 2016 with three continuous-indexed modes: *(latitude, longitude, time)*. The tensor contains $56K$ observations across 15 unique latitudes, 95 longitudes, and 267 timestamps.

7.6.2.2 Baselines and Settings

We set three groups of methods as baselines. The first group includes the state-of-art standard Tucker models, including *P-Tucker* [60]: a scalable Tucker algorithm that performs parallel row-wise updates, *Tucker-ALS*: Efficient Tucker decomposition algorithm using alternating least squares(ALS) update [4], and *Tucker-SVI* [38]: Bayesian version of Tucker updating with stochastic variational inference(SVI). The second group includes functional tensor-train(FTT) based methods with different functional basis and optimization strategy:

[1]https://archive.ics.uci.edu/ml/datasets/Beijing+Multi-Site+Air-Quality+Data

[2]https://berkeleyearth.org/data/

FTT-ALS [7], *FTT-ANOVA* [5], and *FTT-cross* [31]. As we can view the prediction task as a regression problem and use the continuous indexe as features, we add *RBF-SVM*: Support Vector Machine [35] with RBF kernels, and *BLR*: Bayesian Linear Regression [54] as competing methods as the third group.

We used the official open-source implementations of most baselines. We use the *TENEVA* library [13, 14] to test the FTT-based methods. We re-scaled all continuous-mode indexes to $[0, 1]$ to ensure numerical robustness. For FunBaT, we varied Matérn kernels $\nu = \{1/2, 3/2\}$ along the kernel parameters for optimal performance for different datasets. We examined all the methods with rank $R \in \{2, 3, 5, 7\}$. We set all modes' ranks to R. Following [86], we randomly sampled 80% observed entry values for training and then tested on the remaining. We repeated the experiments five times, and examined the average root mean-square-error (RMSE), average mean-absolute-error (MAE), and their standard deviations.

To demonstrate the advantages of using continuous indexes rather than index discretization, we set four different discretization granularities by binding the raw continuous indexes into several discrete-indexed bins on *BeijingAir-PM2.5*, *BeijingAir-PM10*, and *BeijingAir-SO2*. We then derived the tensors with four different resolutions: $428 \times 501 \times 1461$ (original resolution), $300 \times 300 \times 1000$, $100 \times 100 \times 300$, and $50 \times 50 \times 150$. We tested the performance of the first group (standard tensor decomposition) baselines on different resolutions. We tested FunBaT, FunBaT-CP, and other baselines with continuous indexes at the original resolution.

7.6.2.3 Prediction Results

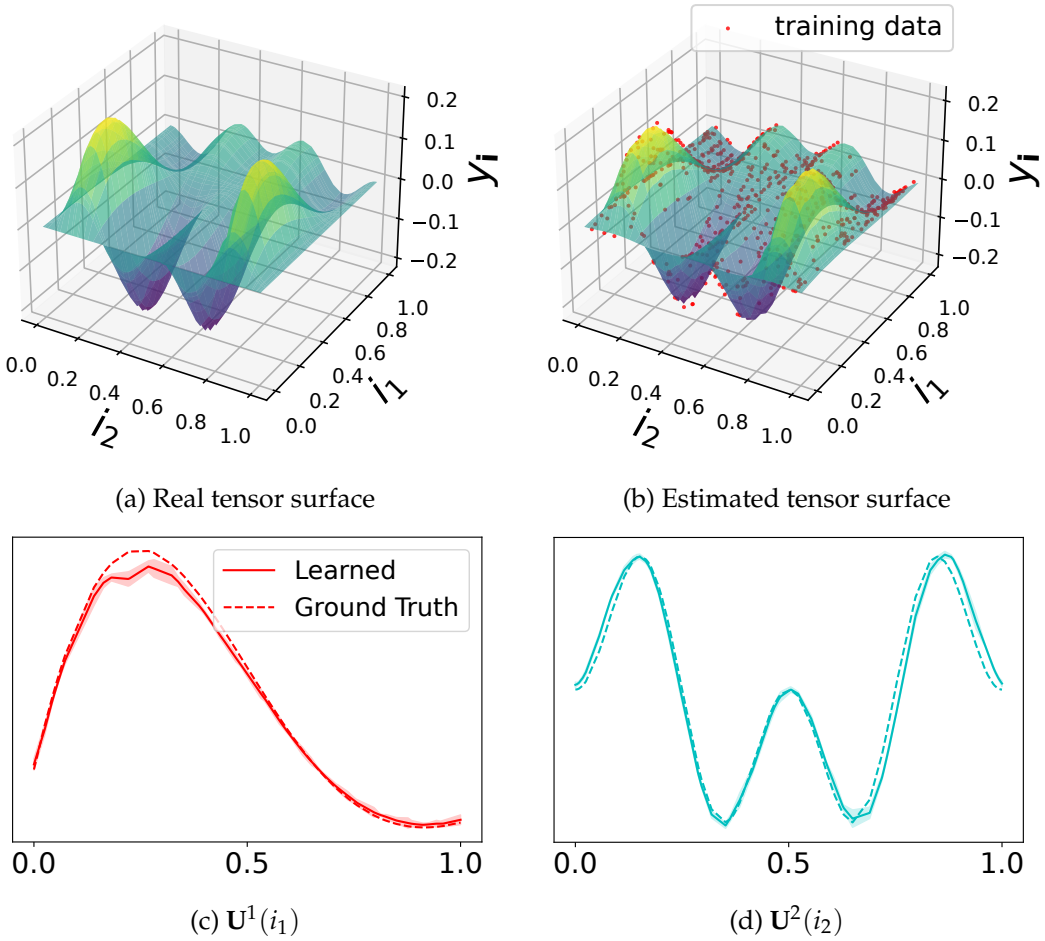
We present the results with $R = 2$ for *BeijingAir-PM2.5*, *BeijingAir-PM10*, and *BeijingAir-SO2* datasets in Table 7.1. The prediction results for *US-TEMP* are listed in Table 7.2. Our approach FunBaT and FunBaT-CP outperform the competing methods by a significant margin in all cases. We found the performance of standard Tucker methods varies a lot with different discrete resolutions, showing the hardness to decide the optimal discretization in real-world applications. We also found that the performance of FunBaT-CP is much better than FunBaT on *US-TEMP*. This might be because the dense Tucker core of FunBaT result in overfitting and is worse for the sparse *US-TEMP* dataset.

7.6.2.4 Investigation of Learned Functions

We explored whether the learned mode functions reveal interpretable patterns that are consistent with domain knowledge. We run FunBaT on *US-TEMP* dataset with $R = 1$, and plotted the learned latent functions of the three modes *(latitude, longitude, time)*, shown in Figs. 7.2, 7.3 and 7.4. As the first two modes correspond to latitude and longitude, respectively, representing real geo-locations, we marked out the city groups (the shaded circles) and some city names in Figs. 7.2 and 7.3. The $\mathbf{U}^1$ of the lantitude mode in Fig. 7.2 shows a clear decreasing trend from southern cities like Miami (in Florida) to northern cities like Anchorage (in Alaska), which is consistent with the fact that temperatures are generally higher in the south and lower in the north. The learned longitude-mode $\mathbf{U}^2$ in Fig. 7.3 exhibits a steady trend from east to west, but with a region (the blue circle) with lower values near the Western longitude 110°. That is the *Rocky Mountain Area* including cities like Denver and Salt Lake City with higher altitudes, and results in lower temperatures. The time-mode $\mathbf{U}^3$ in Fig. 7.4 provides meaningful insights of the climate change in history. It reveals a gradual increase over the past 260 years, with a marked acceleration after 1950. This pattern aligns with the observed trend of rising global warming following the industrialization of the mid-20th century. The function around 1750-1770, characterized by lower and oscillating values, corresponds to the *Little Ice Age*, a well-documented period of global cooling in history.

7.7 Conclusion

We proposed FunBaT, a Bayesian method to generalize Tucker decomposition to the functional field to model continuous-indexed tensor data. We adopt the state-space GPs as functional prior and develop an efficient inference algorithm based on CEP. The results on both synthetic and real-world tasks demonstrate the effectiveness of our method.



(a) Real tensor surface

(b) Estimated tensor surface

(c) $\mathbf{U}^1(i_1)$

(d) $\mathbf{U}^2(i_2)$

Figure 7.1: Results of synthetic data.

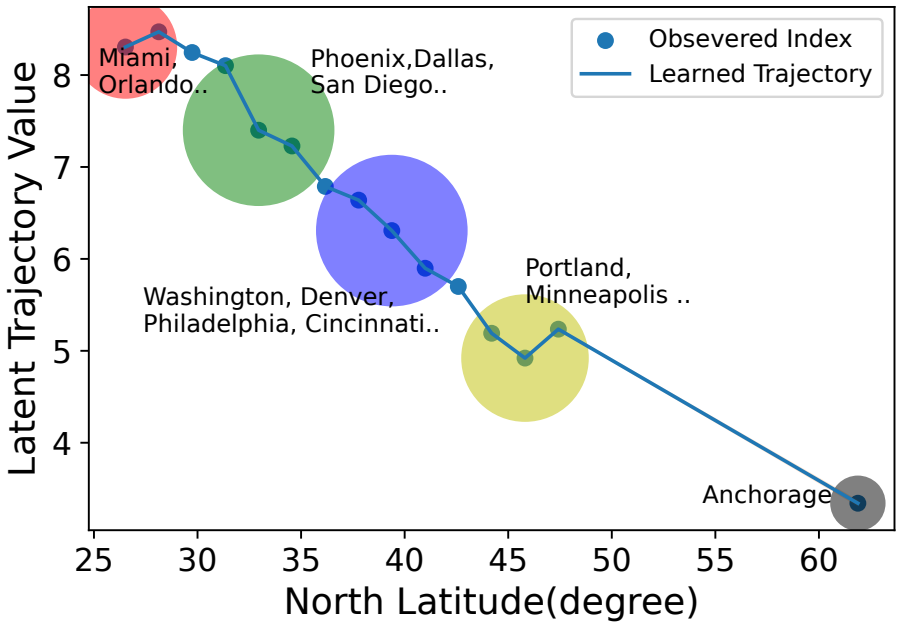


Figure 7.2: $\mathbf{U}^1(i_1)$: Mode of latitude.

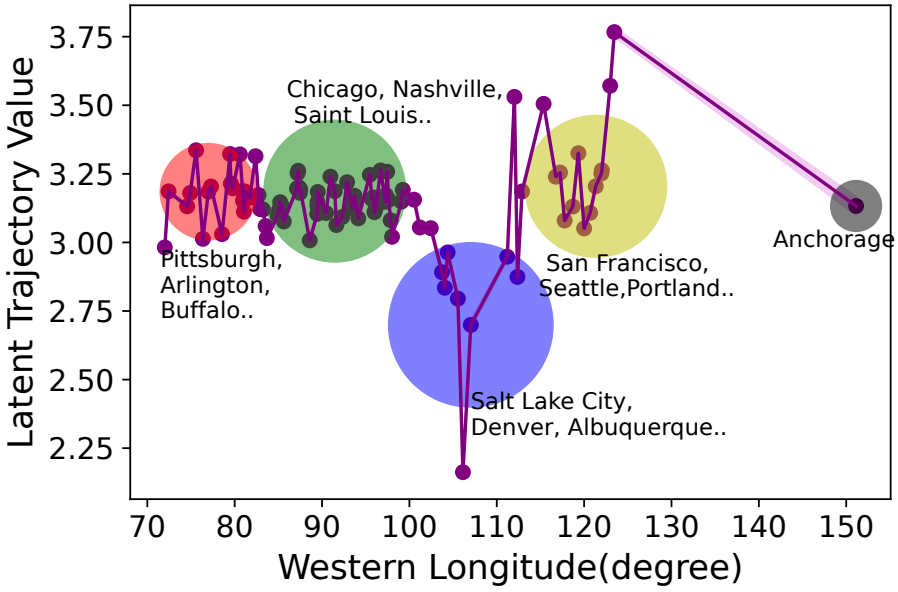


Figure 7.3: $\mathbf{U}^2(i_2)$: Mode of longitude.

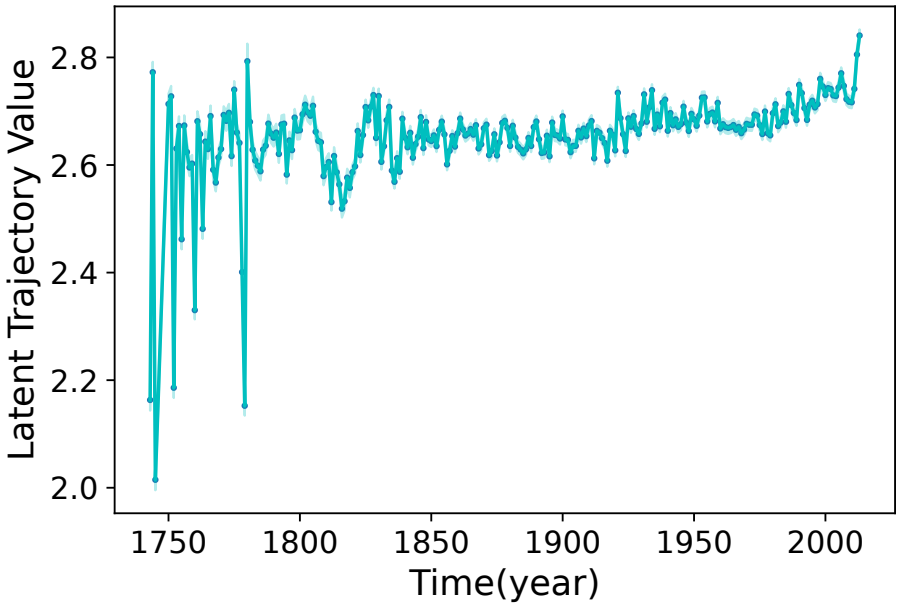


Figure 7.4: $\mathbf{U}^3(i_3)$: Mode of time.

Table 7.1: Prediction error over *BeijingAir-PM2.5*, *BeijingAir-PM10*, and *BeijingAir-SO2* with $R = 2$, which were averaged over five runs.

	RMSE			**MAE**		
Datasets	*PM2.5*	*PM10*	*SO2*	*PM2.5*	*PM10*	*SO2*
Resolution: $50 \times 50 \times 150$						
P-Tucker	0.805 ± 0.017	0.787 ± 0.006	0.686 ± 0.02	0.586 ± 0.003	0.595 ± 0.005	0.436 ± 0.011
Tucker-ALS	1.032 ± 0.049	1.005 ± 0.029	0.969 ± 0.027	0.729 ± 0.016	0.741 ± 0.007	0.654 ± 0.034
Tucker-SVI	0.792 ± 0.01	0.8 ± 0.026	0.701 ± 0.08	0.593 ± 0.01	0.605 ± 0.019	0.423 ± 0.031
Resolution: $100 \times 100 \times 300$						
P-Tucker	0.8 ± 0.101	0.73 ± 0.021	0.644 ± 0.023	0.522 ± 0.011	0.529 ± 0.013	0.402 ± 0.008
Tucker-ALS	1.009 ± 0.027	1.009 ± 0.026	0.965 ± 0.023	0.738 ± 0.01	0.754 ± 0.007	0.68 ± 0.011
Tucker-SVI	0.706 ± 0.011	0.783 ± 0.067	0.69 ± 0.086	0.509 ± 0.008	0.556 ± 0.031	0.423 ± 0.031
Resolution: $300 \times 300 \times 1000$						
P-Tucker	0.914 ± 0.126	1.155 ± 0.001	0.859 ± 0.096	0.401 ± 0.023	0.453 ± 0.002	0.366 ± 0.015
Tucker-ALS	1.025 ± 0.044	1.023 ± 0.038	1.003 ± 0.019	0.742 ± 0.011	0.757 ± 0.011	0.698 ± 0.007
Tucker-SVI	1.735 ± 0.25	1.448 ± 0.176	1.376 ± 0.107	0.76 ± 0.033	0.747 ± 0.028	0.718 ± 0.023
Resolution: $428 \times 501 \times 1461$ (original)						
P-Tucker	1.256 ± 0.084	1.397 ± 0.001	0.963 ± 0.169	0.451 ± 0.017	0.493 ± 0.001	0.377 ± 0.019
Tucker-ALS	1.018 ± 0.034	1.012 ± 0.021	0.997 ± 0.024	0.738 ± 0.005	0.756 ± 0.007	0.698 ± 0.011
Tucker-SVI	1.891 ± 0.231	1.527 ± 0.107	1.613 ± 0.091	0.834 ± 0.032	0.787 ± 0.018	0.756 ± 0.014
Methods using continuous indexes						
FTT-ALS	1.020 ± 0.013	1.001 ± 0.013	1.001 ± 0.026	0.744 ± 0.007	0.755 ± 0.007	0.696 ± 0.011
FTT-ANOVA	2.150 ± 0.033	2.007 ± 0.015	1.987 ± 0.036	1.788 ± 0.031	1.623 ± 0.014	1.499 ± 0.018
FTT-cross	0.942 ± 0.025	0.933 ± 0.012	0.844 ± 0.026	0.566 ± 0.018	0.561 ± 0.011	0.467 ± 0.033
RBF-SVM	0.995 ± 0.015	0.955 ± 0.02	0.794 ± 0.026	0.668 ± 0.008	0.674 ± 0.014	0.486 ± 0.026
BLR	0.998 ± 0.013	0.977 ± 0.014	0.837 ± 0.021	0.736 ± 0.007	0.739 ± 0.008	0.573 ± 0.009
FunBaT-CP	0.296 ± 0.018	0.343 ± 0.028	$\mathbf{0.386 \pm 0.009}$	$\mathbf{0.18 \pm 0.002}$	0.233 ± 0.013	0.242 ± 0.003
FunBaT	$\mathbf{0.288 \pm 0.008}$	$\mathbf{0.328 \pm 0.004}$	$\mathbf{0.386 \pm 0.01}$	0.183 ± 0.006	$\mathbf{0.226 \pm 0.002}$	$\mathbf{0.241 \pm 0.004}$

Table 7.2: Prediction error of *US-TEMP*, which were averaged over five runs.

	RMSE			**MAE**		
Mode-Rank	R=3	R=5	R=7	R=3	R=5	R=7
P-Tucker	1.306 ± 0.02	1.223 ± 0.022	1.172 ± 0.042	0.782 ± 0.011	0.675 ± 0.014	0.611 ± 0.007
Tucker-ALS	> 10	> 10	> 10	> 10	> 10	> 10
Tucker-SVI	1.438 ± 0.025	1.442 ± 0.021	1.39 ± 0.09	0.907 ± 0.005	0.908 ± 0.005	0.875 ± 0.072
FTT-ALS	1.613 ± 0.0478	1.610 ± 0.052	1.609 ± 0.055	0.967 ± 0.009	0.953 ± 0.007	0.942 ± 0.010
FTT-ANOVA	5.486 ± 0.031	4.619 ± 0.054	3.856 ± 0.059	4.768 ± 0.026	4.026 ± 0.100	3.123 ± 0.0464
FTT-cross	1.415 ± 0.0287	1.312 ± 0.023	1.285 ± 0.052	0.886 ± 0.011	0.822 ± 0.006	0.773 ± 0.014
RBF-SVM	2.374 ± 0.047	2.374 ± 0.047	2.374 ± 0.047	1.44 ± 0.015	1.44 ± 0.015	1.44 ± 0.015
BLR	2.959 ± 0.041	2.959 ± 0.041	2.959 ± 0.041	2.029 ± 0.011	2.029 ± 0.011	2.029 ± 0.011
FunBaT-CP	$\mathbf{0.805 \pm 0.06}$	$\mathbf{0.548 \pm 0.03}$	$\mathbf{0.551 \pm 0.048}$	$\mathbf{0.448 \pm 0.06}$	$\mathbf{0.314 \pm 0.005}$	$\mathbf{0.252 \pm 0.008}$
FunBaT	1.255 ± 0.108	1.182 ± 0.117	1.116 ± 0.142	0.736 ± 0.069	0.647 ± 0.05	0.572 ± 0.089

CHAPTER 8

SUMMARY AND FUTURE WORK

This dissertation has advanced the field of tensor decomposition by developing novel probabilistic tensor models tailored to dynamic scenarios, where traditional tensor decomposition methods falter. The primary objective was to address the inherent limitations of classical tensor decomposition approaches, particularly their inadequacy in handling streaming, temporal, and continuously indexed tensor data. This work introduced a series of innovative models that not only enhance the flexibility and applicability of tensor decomposition but also improve the interpretability and robustness of these models in dynamic scenarios.

8.1 Contributions

Through systematic methodology and rigorous testing, we have demonstrated that the new models provide substantial improvements over traditional methods, especially in terms of adaptability to changes in data patterns, efficiency in computational performance, and effectiveness in handling large-scale tensor data. The contributions of this dissertation are significant, setting new benchmarks for future research in tensor decomposition and its application to dynamic data systems across various fields such as social media analysis, sensor data monitoring, and real-time recommendation systems.

- **Bayesian Streaming Sparse Tucker Decomposition (BASS):** Motivated by the need for efficient and interpretable tensor decompositions in streaming data scenarios, the BASS model introduces a spike-and-slab prior to selectively retain significant tensor interactions while discarding the rest. This approach facilitates sparse representations and prevents overfitting, ideal for high-dimensional streaming data. Innovative one-shot incremental updates enable the model to adapt quickly to new data without retraining from scratch, preserving computational resources while ensuring timely updates.

- **Streaming Bayesian Deep Tensor Factorization (SBDT):** SBDT combines deep learning with Bayesian inference to tackle the complexity of streaming tensor data. By integrating Bayesian neural networks, this model dynamically adapts to data streams, leveraging deep architectures to capture non-linear interactions within the data. The use of spike-and-slab priors on neural network weights promotes model sparsity and interpretability. The development of an efficient streaming posterior inference algorithm underpins real-time learning and decision-making.
- **Bayesian Continuous-Time Tucker Decomposition (BCTT):** Addressing the challenge of capturing intrinsic temporal dynamics in tensor data, BCTT models the tensor core as a time-varying function. Gaussian process priors enable the model to accommodate a wide range of temporal behaviors, providing the flexibility to model complex temporal relationships. This method stands out by integrating stochastic differential equations to represent temporal Gaussian processes, enhancing both the accuracy and efficiency of temporal data analysis.
- **Streaming Factor Trajectory Learning for Temporal Tensor Decomposition (SFTL):** SFTL is designed to trace the evolving trajectories of tensor factors over time, using Gaussian processes modeled via stochastic differential equations. This method allows for the continuous update of factor states, enabling the model to adjust to changes in data trends efficiently. The use of state-space representations for Gaussian processes significantly reduces computational complexity, making this approach particularly suited for large-scale streaming data.
- **Functional Bayesian Tucker Decomposition for Continuous-Indexed Tensor (FunBaT):** FunBaT extends tensor decomposition techniques to handle continuous-indexed data, such as spatial or spectral data that cannot be neatly discretized without loss of information. By treating the continuous indices with Gaussian process-based latent functions and converting these into state-space models, FunBaT achieves a high level of efficiency and scalability. This approach not only preserves the detailed structure of the data but also opens up new possibilities for tensor analysis in fields where continuous data is prevalent.

Through the development and implementation of these five advanced methods, this dissertation has systematically and comprehensively addressed the challenges associated

with dynamic tensor data. Each method has been meticulously designed to cater to specific aspects of tensor dynamics—ranging from streaming updates and continuous-time modeling to handling continuous-indexed data. Collectively, these innovations significantly extend the capabilities of Bayesian tensor learning, providing robust, efficient, and scalable solutions that can adapt to the evolving nature of real-world data. This work not only enhances the flexibility and applicability of tensor decomposition but also establishes a new benchmark for future research in the field, paving the way for more sophisticated data analysis techniques in various dynamic settings.

8.2 Future Work

The methodologies and insights garnered from this work may lay a foundation for future advancements in machine learning, particularly in how high-order data structures are understood and utilized across various domains. Below, we discuss the future directions that will extend the scope of Bayesian tensor learning to a wider array of machine learning applications, leveraging the versatility and depth of the approaches developed.

- **Unified Bayesian Frameworks for Integrating Diverse High-Order Data**

  High-order structural data comes in a kaleidoscope of forms: sparse tensors, graphs, hypergraphs, and spatiotemporal sequences, complex systems among others. The prevailing approach in machine learning has treated these as distinct entities, with tailored tools crafted for each – tensor decomposition for sparse tensors with low-rank factors, Graph Neural Networks (GNNs) for graphs with structural smoothness, and LSTM/transformers for sequence data with long-term dependency modeling. However, there remains a largely untapped potential in recognizing and exploiting the intrinsic relationships between these data types. For example, tensors are closely related to hypergraphs , while climate spatial-temporal sequences can be conceptualized as continuous-indexed temporal tensors, and physical dynamical systems may be viewed as flows of information and energy through structural spaces.

  This compartmentalization has led to a siloed research landscape, with experts in one domain often oblivious to parallel advances in another. Therefore, a promising direction for bridging these divides is to forge connections across these types of high-

order structure data, crafting unified Bayesian frameworks that meld the insights and methodologies from diverse fields. Our work Dynamic Tensor Decomposition via Neural Diffusion-Reaction Processes [95] represents an initial foray into this realm – it is a framework that marries graph learning with dynamic tensor decomposition, modeling temporal tensors as dynamic stochastic processes on graphs.

Envisioning beyond this, we can develop a meta-framework that serves as a translation layer between these data types. Such a framework can utilize neural representation theories to map one high-order data form to another, thus enabling cross-pollination of insights and techniques. For instance, we can represent graphs as multi-dimensional arrays and apply tensor decomposition methods to uncover latent relationships, or reframe spatiotemporal patterns within the graph-theoretical domain to harness the power of GNNs for forecasting and anomaly detection in complex systems. This unified approach could yield a step change in how we perceive and interact with high-order data, leading to breakthroughs in interpretability, prediction accuracy, and, ultimately, our understanding of complex phenomena.

- **Adaptive Representation Learning for Large-Scale Dynamic Systems**

  Recently, large language models (LLMs) like ChatGPT have heralded a revolution not only in AI but in the broader spectrum of human technology. The essence of their success lies in the model's in-context learning capabilities and the transferable, task-agnostic representation learning in NLP, known as the pretrained word embeddings. However, beyond NLP, the creation of embeddings that are adaptive and context-aware for large-scale dynamic systems remains a formidable challenge. Current methodologies often produce representations that are task-specific, static, and not generalizable. For instance, latent representations learned in recommender systems are often tailored to a particular platform's data and no longer applicable to other domains. Moreover, these representations are often static and do not adapt to the evolving data, leading to suboptimal performance in real-world applications.

  Drawing inspiration from the success in LLM, our ambition is to craft "embeddings for everything"— multitask, broadly applicable, and context-aware representations with domain-specific constraints. This involves developing scalable, efficient, and dynamic representation learning for real-world large-scale dynamic

systems. The latent embeddings that we aim to learn are not mere static vectors but continuous, adaptive latent functions that can be updated online and are suitable for multiple tasks. Our current work on streaming analysis and latent dynamics modeling is pivotal in achieving this goal.

The benefits of such universal representation learning are manifold. Take climate modeling as an instance. By learning the latent dynamics behind temperature changes, we can significantly improve predictions not just for temperature, but for related areas like ocean currents patterns and energy usage. Additionally, a promising avenue lies in compressing LLMs themselves for deployment on edge devices. Our tensorized approach to compressing neural network weights—a direction that has garnered much attention—can be particularly effective for LLMs that are constantly updated and are too large for existing tensor decomposition methods to handle. Our work on streaming tensor analysis provides a solution, paving the way for more efficient and adaptive LLMs.

- **Encoding *First Principles* for Scientific Discovery and Applications**

  Harnessing the power of AI to enhance scientific discovery, a field often referred to as **AI4science**, is one of the most promising and impactful directions in machine learning. A pivotal challenge in AI4science is to ensure that ML models are deeply attuned to the fundamental mechanisms—*First principle* — governing various scientific domains, such as differential equations in physical systems, conservation laws in chemistry, symmetry in materials grids, many-body interactions in quantum mechanics, and scaling laws in urban science. To yield more accurate, trustworthy, and interpretable outcomes, ML models must recognize and leverage these universal natural laws to help scientific discovery.

  Some current ML initiatives have acknowledged this necessity, developing models like physics-informed networks (PINNs) [70] that encode differential equations, and Fourier operator learning (FNO) [49] that captures wave-like data transformations. Equivariant GNNs [76] are being used to encode symmetry. My research into Bayesian models represents a powerful and versatile framework for integrating these universal natural laws into the models' priors. Our existing works, including GP-PDE-Solver [22], BayOTIDE [26], and BASS-Tucker, have begun this journey, en-

coding domain mechanisms like differential equations, periodic patterns, and sparse structures into the priors of the model.

Moving forward, we aim to deepen this exploration, infusing Bayesian models with nature laws in broader scientific domains. What's more, we will try to better utilize the Bayesian methods' unique strength in uncertainty quantification—a key feature for scientific discovery. By further integrating tools such as Bayesian Optimization (BO), we can enhance data-driven scientific discovery while conforming to domain-specific first principles. A prime example is in drug discovery, where these models can be employed to predict complex interactions between molecules and biological systems. The Bayesian approach allows us to navigate the inherent uncertainties in predicting drug outcomes, identify optimal therapeutic targets, and streamline the development of new treatments.

In summary, we believe the research conducted in this dissertation not only addresses current challenges but also opens up exciting avenues for future work. The next steps will delve deeper into the intersection of Bayesian modeling and high-order structure data, setting the stage for groundbreaking developments in machine learning and scientific discovery.

REFERENCES

[1] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Morup, *Scalable tensor factorizations for incomplete data*, Chemometr. Intell. Lab. Syst., 106 (2011), pp. 41–56.

[2] D. Ahn, J.-G. Jang, and U. Kang, *Time-aware tensor decomposition for sparse tensors*, Mach. Learn., 111 (2021), pp. 1409–1430.

[3] D. Bacciu and D. P. Mandic, *Tensor decompositions in deep learning*. Preprint, arXiv:2002.11835 [cs.LG], 2020.

[4] B. W. Bader and T. G. Kolda, *Efficient MATLAB computations with sparse and factored tensors*, SIAM J. Sci. Comput., 30 (2008), pp. 205–231.

[5] R. Ballester-Ripoll, E. G. Paredes, and R. Pajarola, *Sobol tensor trains for global sensitivity analysis*, Reliab. Eng. Syst. Saf., 183 (2019), pp. 311–322.

[6] P. J. Bickel and K. A. Doksum, *Mathematical Statistics: Basic Ideas and Selected Topics*, vol. 1, CRC Press, Boca Raton, FL, 2015.

[7] D. Bigoni, A. P. Engsig-Karup, and Y. M. Marzouk, *Spectral tensor-train decomposition*, SIAM J. Sci. Comput., 38 (2016), pp. A2405–A2439.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, NY, 2007.

[9] X. Boyen and D. Koller, *Tractable inference for complex stochastic processes*, in Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, ACM, 1998, pp. 33–42.

[10] T. Broderick, N. Boyd, A. Wibisono, A. C. Wilson, and M. I. Jordan, *Streaming variational Bayes*, in Advances in Neural Information Processing Systems, NeurIPS, 2013, pp. 1–9.

[11] T. D. Bui, C. Nguyen, and R. E. Turner, *Streaming sparse Gaussian process approximations*, in Advances in Neural Information Processing Systems, NeurIPS, 2017, pp. 21–29.

[12] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, *Neural ordinary differential equations*, in Advances in Neural Information Processing Systems, NeurIPS, 2018, pp. 1–9.

[13] A. Chertkov, G. Ryzhakov, G. Novikov, and I. Oseledets, *Optimization of functions given in the tensor train format*. Preprint, arXiv:2209.14808 [math.NA], 2022.

[14] A. Chertkov, G. Ryzhakov, and I. Oseledets, *Black box approximation in the tensor train format initialized by ANOVA decomposition*, SIAM J. Sci. Comput., 45 (2023), pp. A2101–A2118.

[15] J. H. Choi and S. Vishwanathan, *DFacTo: Distributed factorization of tensors*, in Advances in Neural Information Processing Systems, NeurIPS, 2014, pp. 1296–1304.

[16] W. Chu and Z. Ghahramani, *Probabilistic models for incomplete multi-dimensional arrays*, in Artificial Intelligence and Statistics, PMLR, 2009, pp. 89–96.

[17] L. De Lathauwer, B. De Moor, and J. Vandewalle, *On the best rank-1 and rank-(r1, r2, ..., rn) approximation of higher-order tensors*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1324–1342.

[18] G. Dehaene and S. Barthelmé, *Expectation propagation in the large data limit*, J. R. Stat. Soc. Ser. B Stat. Methodol., 80 (2018), pp. 199–217.

[19] G. P. Dehaene and S. Barthelmé, *Bounding errors of expectation-propagation*, in Advances in Neural Information Processing Systems, NeurIPS, 2015, pp. 1–9.

[20] V. L. Deringer, A. P. Bartók, N. Bernstein, D. M. Wilkins, M. Ceriotti, and G. Csányi, *Gaussian process regression for materials and molecules*, Chem. Rev., 121 (2021), pp. 10073–10141.

[21] Y. Du, Y. Zheng, K.-c. Lee, and S. Zhe, *Probabilistic streaming tensor decomposition*, in Proceedings of the 2018 IEEE International Conference on Data Mining, IEEE, 2018, pp. 99–108.

[22] S. Fang, M. Cooley, D. Long, S. Li, R. Kirby, and S. Zhe, *Solving high frequency and multi-scale PDEs with Gaussian processes*. Preprint, arXiv:2311.04465 [cs.LG], 2024.

[23] S. Fang, R. M. Kirby, and S. Zhe, *Bayesian streaming sparse Tucker decomposition*, in Uncertainty in Artificial Intelligence, PMLR, 2021, pp. 558–567.

[24] S. Fang, A. Narayan, R. Kirby, and S. Zhe, *Bayesian continuous-time Tucker decomposition*, in International Conference on Machine Learning, PMLR, 2022, pp. 6235–6245.

[25] S. Fang, Z. Wang, Z. Pan, J. Liu, and S. Zhe, *Streaming Bayesian deep tensor factorization*, in International Conference on Machine Learning, PMLR, 2021, pp. 3133–3142.

[26] S. Fang, Q. Wen, Y. Luo, S. Zhe, and L. Sun, *BayOTIDE: Bayesian online multivariate time series imputation with functional decomposition*, in International Conference on Machine Learning, PMLR, 2024, pp. 1–17.

[27] S. Fang, X. Yu, S. Li, Z. Wang, R. Kirby, and S. Zhe, *Streaming factor trajectory learning for temporal tensor decomposition*, in Advances in Neural Information Processing Systems, NeurIPS, 2023, pp. 25965–25978.

[28] S. Fang, X. Yu, Z. Wang, S. Li, M. Kirby, and S. Zhe, *Functional Bayesian Tucker decomposition for continuous-indexed tensor data*, in International Conference on Learning Representation, PMLR, 2024, pp. 1–18.

[29] S. Fang, S. Zhe, K.-c. Lee, K. Zhang, and J. Neville, *Online Bayesian sparse learning with spike and slab priors*, in 2020 IEEE International Conference on Data Mining, IEEE, 2020, pp. 142–151.

[30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, 2016.

[31] A. A. Gorodetsky, S. Karaman, and Y. M. Marzouk, *Function-train: A continuous analogue of the tensor-train decomposition*. Preprint, arXiv:1510.09088 [cs.NA], 2015.

[32] R. A. Harshman, *Foundations of the PARAFAC procedure: Model and conditions for an "explanatory" multi-mode factor analysis*, UCLA Work. Pap. Phon., 16 (1970), pp. 1–84.

[33] J. Hartikainen and S. Särkkä, *Kalman filtering and smoothing solutions to temporal Gaussian process regression models*, in 2010 IEEE International Workshop on Machine Learning for Signal Processing, IEEE, 2010, pp. 379–384.

[34] A. G. Hawkes, *Spectra of some self-exciting and mutually exciting point processes*, Biometrika, 58 (1971), pp. 83–90.

[35] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Schölkopf, *Support vector machines*, IEEE Intell. Syst. Appl., 13 (1998), pp. 18–28.

[36] J. M. Hernández-Lobato and R. Adams, *Probabilistic backpropagation for scalable learning of Bayesian neural networks*, in International Conference on Machine Learning, PMLR, 2015, pp. 1861–1869.

[37] P. Hoff, *Hierarchical multilinear models for multiway data*, Comput. Stat. Data Anal., 55 (2011), pp. 530–543.

[38] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, *Stochastic variational inference*, J. Mach. Learn. Res., 14 (2013), pp. 1303–1347.

[39] C. Hu, P. Rai, and L. Carin, *Zero-truncated poisson tensor factorization for massive binary tensors*. Preprint, arXiv:1508.04210 [cs.LG], 2015.

[40] H. Ishwaran and J. S. Rao, *Spike and slab variable selection: Frequentist and Bayesian strategies*, Ann. Stat., 33 (2005), pp. 730–773.

[41] R. E. Kalman, *A new approach to linear filtering and prediction problems*, Trans. ASME D, 82 (1960), pp. 35–44.

[42] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, *GigaTensor: Scaling tensor analysis up by 100 times-algorithms and discoveries*, in Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2012, pp. 316–324.

[43] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*. Preprint, arXiv:1412.6980 [cs.LG], 2014.

[44] D. P. Kingma and M. Welling, *Auto-encoding variational Bayes*. Preprint, arXiv:1312.6114 [cs.LG], 2013.

[45] T. G. Kolda, *Multilinear Operators for Higher-Order Decompositions*, vol. 2, United States. Department of Energy, 2006.

[46] P. Lancaster and L. Rodman, *Algebraic Riccati Equations*, Clarendon Press, Oxford, UK, 1995.

[47] D. D. Lee and H. S. Seung, *Learning the parts of objects by non-negative matrix factorization*, Nature, 401 (1999), pp. 788–791.

[48] S. Li, R. Kirby, and S. Zhe, *Decomposing temporal high-order interactions via latent odes*, in International Conference on Machine Learning, PMLR, 2022, pp. 12797–12812.

[49] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, *Fourier neural operator for parametric partial differential equations*. Preprint, arXiv:2010.08895 [cs.LG], 2020.

[50] B. Liu, L. He, Y. Li, S. Zhe, and Z. Xu, *NeuralCP: Bayesian multiway data analysis with neural tensor decomposition*, Cogn. Comput., 10 (2018), pp. 1051–1061.

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

[52] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, *When Gaussian process meets big data: A review of scalable GPs*, IEEE Trans. Neural Netw. Learn. Syst., 31 (2020), pp. 4405–4423.

[53] D. J. C. MacKay, *The evidence framework applied to classification networks*, Neural Comput., 4 (1992), pp. 720–736.

[54] T. Minka, *Bayesian Linear Regression*, Technical report, Massachusetts Institute of Technology, 2000.

[55] T. P. Minka, *Expectation propagation for approximate Bayesian inference*, in Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence, PMLR, 2001, pp. 362–369.

[56] T. P. Minka, *A Family of Algorithms for Approximate Bayesian Inference*, PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2001.

[57] K. P. Murphy, Y. Weiss, and M. I. Jordan, *Loopy belief propagation for approximate inference: An empirical study*, in Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence, Morgan Kaufmann Publishers Inc., 1999, pp. 467–475.

[58] A. Nouy, *Low-rank tensor methods for model order reduction*. Preprint, arXiv:1511.01555 [math.NA], 2015.

[59] G. W. Oehlert, *A note on the delta method*, Am. Stat., 46 (1992), pp. 27–29.

[60] S. Oh, N. Park, S. Lee, and U. Kang, *Scalable Tucker factorization for sparse tensors-algorithms and discoveries*, in IEEE 34th International Conference on Data Engineering, IEEE, 2018, pp. 1120–1131.

[61] B. Oksendal, *Stochastic Differential Equations: An Introduction with Applications*, Springer Science & Business Media, Berlin, Germany, 2013.

[62] I. V. Oseledets, *Tensor-train decomposition*, SIAM J. Sci. Comput., 33 (2011), pp. 2295–2317.

[63] Z. Pan, Z. Wang, and S. Zhe, *Scalable nonparametric factorization for high-order interaction events*, in International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 4325–4335.

[64] ——, *Streaming nonlinear Bayesian tensor decomposition*, in Conference on Uncertainty in Artificial Intelligence, PMLR, 2020, pp. 490–499.

[65] R. S. Pasricha, E. Gujral, and E. E. Papalexakis, *Adaptive granularity in tensors: A quest for interpretable structure*, Front. Big Data, 5 (2022), 929511.

[66] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., *PyTorch: An imperative style, high-performance deep learning library*, in Advances in Neural Information Processing Systems, NeurIPS, 2019, pp. 8026–8037.

[67] J. Quiñonero-Candela and C. E. Rasmussen, *A unifying view of sparse approximate Gaussian process regression*, J. Mach. Learn. Res., 6 (2005), pp. 1939–1959.

[68] P. Rai, *Sparse Low Rank Approximation of Multivariate Functions–Applications in Uncertainty Quantification*, Ph.D. dissertation, Ecole Central de Nantes, Nantes, France, 2014.

[69] P. Rai, Y. Wang, S. Guo, G. Chen, D. Dunson, and L. Carin, *Scalable Bayesian low-rank decomposition of incomplete multiway tensors*, in Proceedings of the 31th International Conference on Machine Learning, PMLR, 2014, pp. 1–9.

[70] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, J. Comput. Phys., 378 (2019), pp. 686–707.

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

[72] H. E. Rauch, F. Tung, and C. T. Striebel, *Maximum likelihood estimates of linear dynamic systems*, AIAA J., 3 (1965), pp. 1445–1450.

[73] M. Rogers, L. Li, and S. J. Russell, *Multilinear dynamical systems for tensor time series*, Advances in Neural Information Processing Systems, 26 (2013), pp. 2634–2642.

[74] S. Särkkä, *Bayesian Filtering and Smoothing*, Cambridge University Press, Cambridge, UK, 2013.

[75] ——, *Recursive Bayesian Inference on Stochastic Differential Equations*, Ph.D. dissertation, Helsinki University of Technology, Espoo, Finland, 2006.

[76] V. G. Satorras, E. Hoogeboom, and M. Welling, *E(n) equivariant graph neural networks*, in International Conference on Machine Learning, PMLR, 2021, pp. 9323–9332.

[77] A. Schein, J. Paisley, D. M. Blei, and H. Wallach, *Bayesian Poisson tensor factorization for inferring multilateral relations from sparse dyadic event counts*, in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2015, pp. 1045–1054.

[78] A. Schein, M. Zhou, D. M. Blei, and H. Wallach, *Bayesian Poisson Tucker decomposition for learning the structure of international relations*, in Proceedings of the 33rd International Conference on International Conference on Machine Learning, PMLR, 2016, pp. 2810–2819.

[79] M. N. Schmidt, *Function factorization using warped Gaussian processes*, in Proceedings of the 26th Annual International Conference on Machine Learning, PMLR, 2009, pp. 921–928.

[80] E. Schulz, M. Speekenbrink, and A. Krause, *A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions*, J. Math. Psychol., 85 (2018), pp. 1–16.

[81] A. Shafieloo, A. G. Kim, and E. V. Linder, *Gaussian process cosmography*, Phys. Rev. D, 85 (2012), 123530.

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

[83] E. Snelson and Z. Ghahramani, *Sparse Gaussian processes using pseudo-inputs*, in Advances in Neural Information Processing Systems, NeurIPS, 2005, pp. 1257–1264.

[84] Q. Song, X. Huang, H. Ge, J. Caverlee, and X. Hu, *Multi-aspect streaming tensor completion*, in Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2017, pp. 435–443.

[85] I. Sutskever, J. B. Tenenbaum, and R. R. Salakhutdinov, *Modelling relational data using Bayesian clustered tensor factorization*, in Advances in Neural Information Processing Systems, NeurIPS, 2009, pp. 1821–1828.

[86] C. Tillinghast, S. Fang, K. Zhang, and S. Zhe, *Probabilistic neural-kernel tensor decomposition*, in 2020 IEEE International Conference on Data Mining, IEEE, 2020, pp. 531–540.

[87] C. Tillinghast, Z. Wang, and S. Zhe, *Nonparametric sparse tensor factorization with hierarchical Gamma processes*, in International Conference on Machine Learning, PMLR, 2022, pp. 21432–21448.

[88] C. Tillinghast and S. Zhe, *Nonparametric decomposition of sparse tensors*, in International Conference on Machine Learning, PMLR, 2021, pp. 10301–10311.

[89] M. Titsias and M. Lázaro-Gredilla, *Spike and slab variational inference for multi-task and multiple kernel learning*, in Advances in Neural Information Processing Systems, NeurIPS, 2011, pp. 23–31.

[90] M. K. Titsias, *Variational learning of inducing variables in sparse Gaussian processes*, in International Conference on Artificial Intelligence and Statistics, PMLR, 2009, pp. 567–574.

[91] L. Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311.

[92] M. J. Wainwright and M. I. Jordan, *Graphical Models, Exponential Families, and Variational Inference*, Now Publishers Inc, Hanover, MA, 2008.

[93] C. Wang and D. M. Blei, *Variational inference in nonconjugate models*, J. Mach. Learn. Res., 14 (2013), pp. 1005–1031.

[94] Z. Wang, X. Chu, and S. Zhe, *Self-modulating nonparametric event-tensor factorization*, in International Conference on Machine Learning, PMLR, 2020, pp. 9857–9867.

[95] Z. Wang, S. Fang, S. Li, and S. Zhe, *Dynamic tensor decomposition via neural diffusion-reaction processes*, in Advances in Neural Information Processing Systems, NeurIPS, 2024, pp. 1–11.

[96] Z. Wang and S. Zhe, *Conditional expectation propagation*, in Uncertainty in Artificial Intelligence, PMLR, 2019, pp. 28–37.

[97] ——, *Nonparametric factor trajectory learning for dynamic tensor decomposition*, in International Conference on Machine Learning, PMLR, 2022, pp. 23459–23469.

[98] K. Wolter, *Introduction to Variance Estimation*, Springer Science & Business Media, New York, NY, 2007.

[99] X. Wu, B. Shi, Y. Dong, C. Huang, and N. V. Chawla, *Neural tensor factorization for temporal interaction learning*, in Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, ACM, 2019, pp. 537–545.

[100] L. Xiong, X. Chen, T.-K. Huang, J. Schneider, and J. G. Carbonell, *Temporal collaborative filtering with Bayesian probabilistic tensor factorization*, in Proceedings of the 2010 SIAM International Conference on Data Mining, SIAM, 2010, pp. 211–222.

[101] Z. Xu, F. Yan, and Y. Qi, *Infinite Tucker decomposition: Nonparametric Bayesian models for multiway data analysis*, in Proceedings of the 29th International Conference on Machine Learning, PMLR, 2012, pp. 1675–1682.

[102] Y. Yang and D. B. Dunson, *Bayesian conditional tensor factorizations for high-dimensional classification*, J. R. Stat. Soc. Ser. B Stat. Methodol., 75 (2013), pp. 569–589.

[103] J. Ye, L. Wang, G. Li, D. Chen, S. Zhe, X. Chu, and Z. Xu, *Learning compact recurrent neural networks with block-term tensor decomposition*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2018, pp. 9378–9387.

[104] Y. Zhang, X. Bi, N. Tang, and A. Qu, *Dynamic tensor recommender systems*, J. Mach. Learn. Res., 22 (2021), pp. 3032–3066.

[105] S. Zhe and Y. Du, *Stochastic nonparametric event-tensor decomposition*, in Proceedings of the 32nd International Conference on Neural Information Processing Systems, NeurIPS, 2018, pp. 6857–6867.

[106] S. Zhe, Y. Qi, Y. Park, Z. Xu, I. Molloy, and S. Chari, *DinTucker: Scaling up Gaussian process models on large multidimensional arrays*, in Thirtieth AAAI Conference on Artificial Intelligence, AAAI, 2016, pp. 1479–1485.

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

[108] S. Zhe, K. Zhang, P. Wang, K.-c. Lee, Z. Xu, Y. Qi, and Z. Ghahramani, *Distributed flexible nonlinear tensor factorization*, in Advances in Neural Information Processing Systems, NeurIPS, 2016, pp. 928–936.