Decomposing Temporal High-Order Interactions via Latent ODEs

Shibo Li [1] **Robert M. Kirby** [1 2] **Shandian Zhe** [1]

Abstract

High-order interactions between multiple objects are common in real-world applications. Although tensor decomposition is a popular framework for high-order interaction analysis and prediction, most methods cannot well exploit the valuable timestamp information in data. The existent methods either discard the timestamps or convert them into discrete steps or use over-simplistic decomposition models. As a result, these methods might not be capable enough of capturing complex, fine-grained temporal dynamics or making accurate predictions for long-term interaction results. To overcome these limitations, we propose a novel Temporal High-order Interaction decompoSition model based on Ordinary Differential Equations (THIS-ODE). We model the time-varying interaction result with a latent ODE. To capture the complex temporal dynamics, we use a neural network (NN) to learn the time derivative of the ODE state. We use the representation of the interaction objects to model the initial value of the ODE and to constitute a part of the NN input to compute the state. In this way, the temporal relationships of the participant objects can be estimated and encoded into their representations. For tractable and scalable inference, we use forward sensitivity analysis to efficiently compute the gradient of ODE state, based on which we use integral transform to develop a stochastic mini-batch learning algorithm. We demonstrate the advantage of our approach in simulation and four real-world applications.

1. Introduction

Many applications involve interactions between multiple objects. For example, *customers* purchase *items* at different *grocery stores*, and *people* take *outdoor exercises* at various *places*. From the results of those interactions, *e.g.,* purchase amount and heart rates, an important task is to learn a representation of the participant objects, *i.e.,* embeddings, from which we can discover hidden structures of the objects, such as communities and outliers, and build predictive tools or pipelines for downstream applications, such as recommendation and health counseling.

In practice, interaction data often includes valuable time information, namely the timestamp at which each interaction occurred. This information implies rich, complex temporal dynamics within the observed interactions. While tensor decomposition (Tucker, 1966; Harshman, 1970; Chu and Ghahramani, 2009; Choi and Vishwanathan, 2014; Zhe et al., 2016b) is a popular framework for representation learning and prediction of high-order interactions, current methods has yet fully exploited the fine-grained temporal information. Most methods either simply drop the timestamps (Pan et al., 2020b; Fang et al., 2021a) or truncate the timestamps to coarse steps, *e.g.,* weeks or months, and learn a representation of the time steps (Xiong et al., 2010; Rogers et al., 2013; Zhe et al., 2016a; Du et al., 2018). The temporal dependencies within each step are therefore ignored. While the most recent work (Zhang et al., 2021) introduces time-varying coefficients to support continuous time, the multilinear decomposition structure and polynomial spline modeling of the coefficients might be oversimplistic, and restrict the approach from capturing highly-nonlinear temporal relationships in data. As a result, the existent methods might not be capable enough of learning complex, fine-grained temporal dynamics underlying the data and accurately predicting long-term interaction results. Although recent works (Zhe and Du, 2018; Pan et al., 2020a; Wang et al., 2020) also use point processes to estimate the temporal dependencies between the interaction events (*e.g.,* excitation and inhibition), they focus on event modeling and ignore the interaction results. These methods cannot make use of the observed or predict future interaction results.

To address these issues, we propose THIS-ODE, a novel decomposition model of temporal high-order interactions. Our model can fully leverage continuous timestamps, flexibly capture all kinds of complex temporal dynamics within the interactions, and encode nonlinear temporal relationships of the participant objects into their representations. Specifically, we model the result of each interaction with a latent

[1]School of Computing, University of Utah [2]Scientific Computing and Imaging (SCI) Institute, University of Utah. Correspondence to: Shandian Zhe <zhe@cs.utah.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

Ordinary Differential Equation (ODE). In order to capture the complex temporal dynamics, we use a neural network (NN) to model the time derivative of the ODE state. The representations of the participant objects are then used to model the initial value of the ODE and serve as a part of the input to the NN to calculate the state. In this way, the ODE is governed by the participants' representations and the nonlinear temporal relationship of the participants can thereby be captured and absorbed into these representations. For tractable inference, we use sensitivity analysis to solve an augmented ODE system that simultaneously evolve each state and its partial derivatives w.r.t. the initialization and model parameters. In doing so, we can efficiently compute the total derivative of the state at any time point. We then embed this computation procedure into a scalable stochastic optimization framework to maximize the log joint probability of the model. To further improve the efficiency, we use integral transform to align the ending time and jointly solve all the ODEs in the mini-batch at a single timestamp.

For evaluation, we examined THIS-ODE in both simulation and real-world applications. The simulation experiments show that THIS-ODE can accurately capture the underlying complex dynamics, and the learned representations further reflected the hidden structures of the objects. We then examined THIS-ODE in four real-world applications. In terms of prediction accuracy, THIS-ODE nearly always outperforms the competing methods by a large margin, in predicting long-term interaction results where the test time frame do not overlap with the training time frame.

2. Notations and Background

Suppose our data consists of interaction results between K types of objects (*e.g., customers*, *products*, and *stores*). For each type k, there are d_k objects. We use integers to index these objects, and so a particular interaction is indexed by a tuple $\mathbf{i} = (i_1, \cdots, i_K)$ where $1 \leq i_k \leq d_k$ for each k. We denote the interaction result (*e.g.,* purchase amount) by $m_{\mathbf{i}} \in \mathbb{R}$. In practice, the timestamp information is also associated with the observed interaction results. Hence, we denote by $m_{\mathbf{i}}(t)$ the result of interaction $\mathbf{i}$ at time t. Assume we have observed N interactions results and their timestamps. We denote the dataset by $\mathcal{D} = \{(\mathbf{i}_1, t_1, y_1), \ldots, (\mathbf{i}_N, t_N, y_N)\}$, where each y_n is a (noisy) observation of $m_{\mathbf{i}_n}(t_n)$. Note that the indices of the interactions $\{\mathbf{i}_1, \ldots \mathbf{i}_N\}$ can have duplications. In other words, a particular interaction might have a sequence of results at different timestamps. For example, when one is taking exercise (*e.g.,* yoga or running), their heart rate can vary along with time. Given the data $\mathcal{D}$, we want to learn a representation for each participant object, and use the representations to reconstruct the observed and to predict future interaction results at different time points, *i.e.,* decomposition. We denote by $\mathbf{u}_j^k \in \mathbb{R}^{r_k}$ the representation of the j-th object of type k, where r_k is the dimension of the representation.

To decompose high-order interactions, a commonly used approach is tensor decomposition. We can view each object type k as one tensor mode and construct a K-mode tensor $\mathcal{M} \in \mathbb{R}^{d_1 \times \ldots \times d_K}$, where each entry corresponds to a particular interaction. The classical Tucker decomposition (Tucker, 1966) models that $\mathcal{M} = \mathcal{W} \times_1 \mathbf{U}^1 \times_2 \ldots \times_K \mathbf{U}^K$ where $\mathcal{W} \in \mathbb{R}^{r_1 \times \ldots \times r_K}$ is parametric core tensor, each $\mathbf{U}^k$ consists of the representations of all the objects in mode k (of size $d_k \times r_k$)), and $\times_k$ is the tensor-matrix multiplication at mode k (Kolda, 2006). If we set $r_1 = \ldots = r_K = r$ and $\mathcal{W}$ to be diagonal, Tucker decomposition is reduced to the popular CANDECOMP/PARAFAC (CP) decomposition (Harshman, 1970),

$$\mathcal{M} = \sum_{j=1}^{r} \lambda_j \cdot \mathbf{U}^1[:, j] \circ \ldots \circ \mathbf{U}^K[:, j], \tag{1}$$

where $\{\lambda_1, \ldots, \lambda_r\}$ correspond to the diagonal elements of $\mathcal{W}$, $\mathbf{U}^k[:, j]$ is the j-the column of $\mathbf{U}^k$, and $\circ$ is the vector outer-product. The representations can be estimated by minimizing a loss (*e.g.,* square loss) on the observed entry values. When the tensor is not fully observed (which is often the case), we can use the element-wise decomposition form. In spite of their popularity, CP and Tucker decomposition only estimate a multilinear relationship between the objects in terms of their representations. To flexibly estimate all kinds of relationships, including highly nonlinear ones, (Xu et al., 2012; Zhe et al., 2015; 2016a) model the value of each entry $m_{\mathbf{i}}$ as an unknown function of the representations of corresponding objects in each mode and assign a Gaussian process prior (Rasmussen and Williams, 2006),

$$m_{\mathbf{i}} = g(\mathbf{u}_{i_1}^1, \ldots, \mathbf{u}_{i_K}^K) \sim \mathcal{GP}\left(0, \kappa(\cdot, \cdot)\right) \tag{2}$$

where $\kappa(\cdot, \cdot)$ is a kernel function.

To incorporate temporal information, current methods usually bin the timestamps into L steps according to a specified interval, *e.g.,* one day or one week, and then augment the tensor with a time mode (Xiong et al., 2010; Rogers et al., 2013; Zhe et al., 2015; 2016a; Du et al., 2018), $\overline{\mathcal{M}} \in \mathbb{R}^{d_1 \times \ldots \times d_K \times L}$, where a representation $\mathbf{s}_j$ is introduced for each time step j ($1 \leq j \leq L$). Then we can apply an arbitrary tensor decomposition approach to estimate all the representations. To better capture the temporal dependencies, we can further model the dynamics between the time-step representations, *e.g.,* via a conditional prior over consecutive steps, $p(\mathbf{t}_{j+1}|\mathbf{t}_j) = \mathcal{N}(\mathbf{t}_{j+1}|\mathbf{t}_j, v\mathbf{I})$ (Xiong et al., 2010) and/or recurrent neural networks.

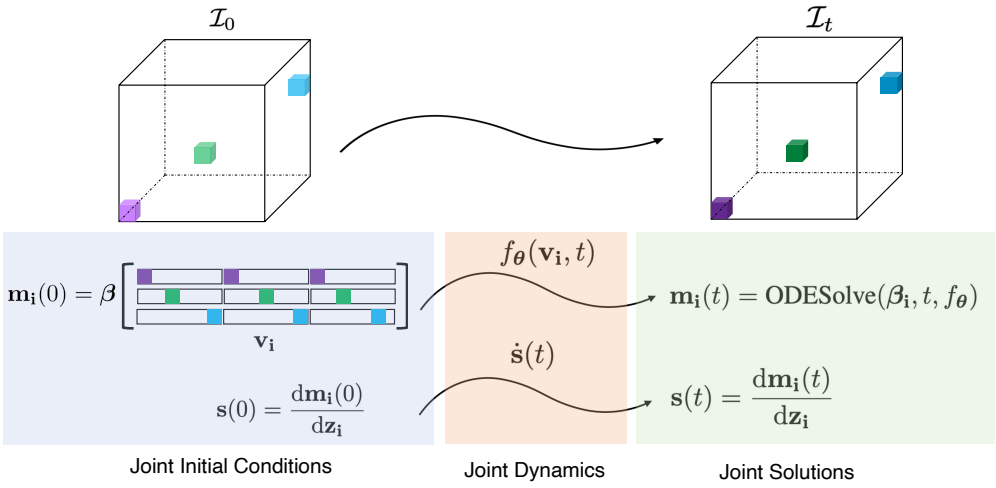


Figure 1. A graphical representation of THIS-ODE.

3. Model

Despite the success of existing methods, they are still not capable enough of capturing complex, fine-grained temporal dynamics and relationships from data. The discrete time steps actually lose the accurate time point information and ignore the temporal dynamics within each step, which can hurt prediction accuracy of the interaction results (especially for long-term results). Although the most recent work (Zhang et al., 2021) introduces a continuous-time coefficients in the CP framework (*i.e.,* $\{\lambda_j\}$ in (1)), its multilinear form and polynomial temporal/trend modeling might be inadequate to estimate complex, highly nonlinear dynamics and relationships between the interaction objects. To address these issues, we propose THIS-ODE, a continuous-time decomposition model based on ordinary differential equations, presented as follows. A graphical illustration is given by Fig. 1.

Specifically, for each particular interaction $\mathbf{i}$, we model the evolution of the interaction result $m_{\mathbf{i}}(t)$ as the trajectory of an unknown latent ODE system, which is governed by the representations of the participant objects $\mathbf{v}_i = \{\mathbf{u}^1_{i_1}, \ldots, \mathbf{u}^K_{i_K}\}$,

$$\begin{cases} \frac{\mathrm{d}m_{\mathbf{i}}(t)}{\mathrm{d}t} = f\left(m_{\mathbf{i}}(t), \mathbf{v_i}, t\right) \\ m_{\mathbf{i}}(0) = \beta(\mathbf{v}_i) \end{cases} \tag{3}$$

where $f(\cdot)$ is the time derivative of the state, and $m_{\mathbf{i}}(0)$ is the initial state. Here $\beta(\cdot)$ is a function that can be the element-wise form of any static decomposition model, *e.g.,* the CP model in (1). Clearly, the ODE system deals with continuous time information in a natural way. In order to flexibly learn all kinds of complex dynamics within the observed sequence, we model $f(\cdot)$ as a neural network (NN) parameterized by $\boldsymbol{\theta}$. As we can see, the initial state and the time derivative are both determined by $\mathbf{v_i}$ — the representations of the participant objects. For the latter, $\mathbf{v_i}$ is a part of the input to the derivative function $f(\cdot)$ and hence influences the evolution of the state. In this way, both the static and nonlinear temporal relationships can be captured and encoded into the learned representations.

Given the system, the state at an arbitrary time point t is computed by $m_{\mathbf{i}}(t) = m_{\mathbf{i}}(0) + \int_0^t f_{\boldsymbol{\theta}}(m_{\mathbf{i}}(s), \mathbf{v_i}, s)\mathrm{d}s$. Although in general this integration is analytically intractable (note $m_{\mathbf{i}}$ is also an input to f), we can use numerical ODE solvers to solve it. There are many general-purpose solvers, such as those based on Runge-Kutta methods (Dormand and Prince, 1980). Being developed for decades by the numerical computation community, these solvers are mature and reliable, allowing a flexible trade-off between the efficiency and numerical precision.

Given the dataset $\mathcal{D} = \{(\mathbf{i}_1, t_1, y_1), \ldots, (\mathbf{i}_N, t_N, y_N)\}$, we use the Gaussian likelihood for each observed interaction result y_n $(1 \le n \le N)$,

$$p(y_n | m_{\mathbf{i}_n}(t_n)) = \mathcal{N}\left(y_n | m_{\mathbf{i}_n}(t_n), \nu^{-1}\right) \tag{4}$$

where ν is the inverse variance of the Gaussian noise. We further assign a Gamma prior distribution over ν and a standard Gaussian prior over the representations. The joint probability of our model is then given by

$$p(\mathcal{U}, \nu, \mathcal{D} | \boldsymbol{\theta}) = \prod_{k=1}^{K} \prod_{j=1}^{d_k} \mathcal{N}(\mathbf{u}^k_j | \mathbf{0}, \mathbf{I}) \cdot \mathrm{Gam}(\nu | a_0, b_0) \cdot \prod_{n=1}^{N} \mathcal{N}(y_n | m_{\mathbf{i}_n}(t_n), \nu^{-1}) \tag{5}$$

where $\mathcal{U} = \{\mathbf{u}^k_j\}_{1 \le k \le K, 1 \le j \le d_k}$ is the collection of all the representations, and a_0 and b_0 are the shape and rate parameters of the Gamma prior. We can set a_0 and b_0 to small

values to make the Gamma prior weakly informative or uninformative (*e.g.,* $a_0 = b_0 = 10^{-3}$).

4. Algorithm

Now we present the model estimation algorithm. Given the joint probability (5), we conduct maximum-a-posterior (MAP) estimation to learn the representations $\mathcal{U}$, NN parameters $\boldsymbol{\theta}$, and the other parameters (*e.g.,* those for $\beta(\cdot)$). However, a critical challenge here is to compute the gradient of the log-likelihood for each data point y_n,

$$J(m_{\mathbf{i}_n}(t_n)) = \log p(y_n | m_{\mathbf{i}_n}(t_n)). \tag{6}$$

Since the state $m_{\mathbf{i}_n}(t_n)$ is obtained from an integration of the nonlinear dynamics, its gradient w.r.t the related model parameters, *e.g.,* $\mathbf{v_i}$ and $\boldsymbol{\theta}$, is analytically intractable. To tackle this challenge, we use forward sensitivity analysis to evolve an augmented ODE system that includes the partial derivative of the state w.r.t to the model parameters and the initialization, based on which, we can efficiently calculate the total derivative of the state at any time point. We then develop an efficient stochastic min-batch learning algorithm.

4.1. Forward Sensitivity Analysis

Specifically, let us denote by $\boldsymbol{\eta}$ all the model parameters we want to estimate. For notation convenience, we drop the subscript n in (6) and consider a general case — for interaction $\mathbf{i}$ we observed y at time t and have the likelihood,

$$J(m_{\mathbf{i}}(t), \boldsymbol{\eta}) = \log p(y_{\mathbf{i}}(t) | m_{\mathbf{i}}(t)) = \log \mathcal{N}(y | m_{\mathbf{i}}(t), \nu^{-1}).$$

According to the chain rule, we have

$$\frac{\mathrm{d}J}{\mathrm{d}\boldsymbol{\eta}} = \frac{\partial J}{\partial \boldsymbol{\eta}} + \frac{\partial J}{\partial m_{\mathbf{i}}(t)} \cdot \frac{\mathrm{d}m_{\mathbf{i}}(t)}{\mathrm{d}\boldsymbol{\eta}}. \tag{7}$$

The partial gradients $\frac{\partial J}{\partial \boldsymbol{\eta}}$ and $\frac{\partial J}{\partial m_{\mathbf{i}}(t)}$ are easy to compute while the bottleneck is the state gradient, $\frac{\mathrm{d}m_{\mathbf{i}}(t)}{\mathrm{d}\boldsymbol{\eta}}$. According to (3), the state at any time t is determined by both the initial state and the parameters that control the dynamics, *i.e.,* $m_{\mathbf{i}}(t) = m_{\mathbf{i}}(m_{\mathbf{i}}^0, \boldsymbol{\eta}, t)$ where $m_{\mathbf{i}}^0 \triangleq m_{\mathbf{i}}(0)$. Note that only a part of parameters in $\boldsymbol{\eta}$, namely, $\mathbf{v}_i$ and $\boldsymbol{\theta}$, actually control the dynamics f. For presentation convenience, we still draw the dependency to the entire $\boldsymbol{\eta}$. Since the initial state is also computed from $\boldsymbol{\eta}$ (see (3)), the total derivative of the state w.r.t. the model parameters is given by

$$\frac{\mathrm{d}m_{\mathbf{i}}(t)}{\mathrm{d}\boldsymbol{\eta}} = \frac{\partial m_{\mathbf{i}}(t)}{\partial \boldsymbol{\eta}} + \frac{\partial m_{\mathbf{i}}(t)}{\partial m_{\mathbf{i}}^0} \cdot \frac{\mathrm{d}m_{\mathbf{i}}^0}{\mathrm{d}\boldsymbol{\eta}}. \tag{8}$$

While $\frac{\mathrm{d}m_{\mathbf{i}}^0}{\mathrm{d}\boldsymbol{\eta}} = \frac{\mathrm{d}\beta}{\mathrm{d}\boldsymbol{\eta}}$ can be computed analytically, the partial derivatives $[\frac{\partial m_{\mathbf{i}}(t)}{\partial m_{\mathbf{i}}^0}; \frac{\partial m_{\mathbf{i}}(t)}{\partial \boldsymbol{\eta}}]$ do not have closed forms. These partial derivatives are called the *sensitivity* of the system, which we denote by $\mathbf{s_i}(t)$. Let us define $\mathbf{z_i} = [m_{\mathbf{i}}^0; \boldsymbol{\eta}]$, and then $\mathbf{s_i}(t) = \frac{\partial m_{\mathbf{i}}(t)}{\partial \mathbf{z_i}}$. To compute the sensitivity, based on (3), we observe that

$$\underbrace{\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial m_{\mathbf{i}}(t)}{\partial \mathbf{z_i}}}_{\dot{\mathbf{s}}_{\mathbf{i}}(t)} = \frac{\partial}{\partial \mathbf{z_i}}\frac{\mathrm{d}m_{\mathbf{i}}(t)}{\mathrm{d}t} = \frac{\partial f}{\partial m_{\mathbf{i}}(t)}\underbrace{\frac{\partial m_{\mathbf{i}}(t)}{\partial \mathbf{z_i}}}_{\mathbf{s}_{\mathbf{i}}(t)} + \frac{\partial f}{\partial \mathbf{z_i}}.$$

Note that since $\frac{\partial f}{\partial m_{\mathbf{i}}^0} = 0$, we have $\frac{\partial f}{\partial \mathbf{z_i}} = [0; \frac{\partial f}{\partial \boldsymbol{\eta}}]$. Incorporating the initial condition, we can construct another ODE system that characterizes how the sensitivity evolves,

$$\begin{cases} \frac{\mathrm{d}\mathbf{s_i}(t)}{\mathrm{d}t} = \frac{\partial f}{\partial m_{\mathbf{i}}(t)}\mathbf{s_i}(t) + \frac{\partial f}{\partial \mathbf{z_i}} \\ \mathbf{s_i}(0) = \frac{\mathrm{d}m_{\mathbf{i}}^0}{\mathrm{d}\mathbf{z_i}}, \end{cases} \tag{9}$$

where $\frac{\mathrm{d}m_{\mathbf{i}}^0}{\mathrm{d}\mathbf{z_i}} = \left[1; \frac{\mathrm{d}\beta}{\mathrm{d}\boldsymbol{\eta}}\right]$. Therefore, we can merge (9) and (3) to obtain an augmented ODE system,

$$\begin{cases} \frac{\mathrm{d}\mathbf{h_i}(t)}{\mathrm{d}t} = \boldsymbol{\alpha}(\mathbf{h_i}, t) \\ \mathbf{h_i}(0) = \mathbf{h_i^0}, \end{cases} \tag{10}$$

where the state $\mathbf{h_i}(t) = [m_{\mathbf{i}}(t); \mathbf{s_i}(t)]$, $\boldsymbol{\alpha}(\mathbf{h_i}, t) = [\frac{\mathrm{d}m_{\mathbf{i}}}{\mathrm{d}t}; \frac{\mathrm{d}s_{\mathbf{i}}}{\mathrm{d}t}]$ and $\mathbf{h_i^0} = [m_{\mathbf{i}}^0; \mathbf{s_i}(0)]$.

To compute the log-likelihood J and its gradient, we just solve (10) forward, read out $\mathbf{h_i}(t)$, and then apply (8) in (7).

4.2. Time Alignment for Efficient Stochastic Mini-Batch Optimization

Next, to scale to a large number of observations, we develop a stochastic mini-batch optimization algorithm based on the sensitivity analysis. Each step, we randomly sample a mini-batch of B observed interactions $\mathcal{B} = \{(\mathbf{i}_{n_1}, t_{n_1}, y_{n_1}), \ldots, (\mathbf{i}_{n_B}, t_{n_B}, y_{n_B})\}$ and update the parameters with a stochastic gradient, namely the gradient of an unbiased stochastic estimate of the log joint probability,

$$\widehat{\mathcal{L}} = \log(\text{Prior}) + \frac{N}{B}\sum\nolimits_{j=1}^{B} J(m_{\mathbf{i}_{n_j}}(t_{n_j})). \tag{11}$$

To compute $\frac{\mathrm{d}\widehat{\mathcal{L}}}{\mathrm{d}\boldsymbol{\eta}}$, we need to solve B augmented ODE systems in the form of (10) to compute each state $\mathbf{h}_{\mathbf{i}_{n_j}}$ at the corresponding timestamp t_{n_j}. While we can sequentially solve these systems, it can be very inefficient because we cannot fully leverage highly-optimized numerical linear algebra libraries, say for parallel computation on GPUs. Therefore, we consider merging the B systems into one ODE system, where the state is $\mathbf{h}(t) = [\mathbf{h}_{\mathbf{i}_{n_1}}(t); \ldots; \mathbf{h}_{\mathbf{i}_{n_B}}(t)]$. Then we solve one ODE, which is equivalent to jointly solving the original B systems.

However, this is still inefficient in that it involves too much additional state computation. Specifically, for the original B ODE systems, we are interested in their states at different

timestamps, namely $\{\mathbf{h}_{\mathbf{i}_{n_j}}(t_{n_j})|1 \leq j \leq B\}$ where t_{n_j} varies across j. When we merge these the states into one joint state $\mathbf{h}$, we have to solve $\mathbf{h}$ at all the B timestamps, *i.e.,* $\{\mathbf{h}(t_{n_j})|1 \leq j \leq B\}$ and then read out the corresponding sub-states at the designated timestamps. This is equivalent to solving every original ODE at all the B timestamps, which gives B^2 states in total, $\{\mathbf{h}_{\mathbf{i}_{n_j}}(t_{n_k})|1 \leq k, j \leq B\}$. However, we only need B states. Therefore, it can waste too much computational expense. For a concrete example, consider a commonly used mini-batch size $B = 100$. With this method, we have to solve 10^4 states although we only need 1% of them.

To tackle this problem, we use an integral transform to align the ending time for solving the ODEs in the mini-batch so as to avoid computing unnecessary states. Specifically, we specify a unified ending time t_e. For each interaction $\mathbf{i}_l$ in the mini-batch ($l \in \{n_1, \ldots, n_B\}$), we modify the ODE system for $\mathbf{h}_{\mathbf{i}_l}$ such that, the state of the new system, denoted by $\widetilde{\mathbf{h}}_{\mathbf{i}_l}$, satisfies $\widetilde{\mathbf{h}}_{\mathbf{i}_l}(t_e) = \mathbf{h}_{\mathbf{i}_l}(t_l)$ (note that t_l is the target timestamp and varies across l). To do so, we observe that, according to (10),

$$\begin{aligned}\mathbf{h}_{\mathbf{i}_l}(t_l) &= \mathbf{h}_{\mathbf{i}_l}(0) + \int_0^{t_l} \boldsymbol{\alpha}\left(\mathbf{h}_{\mathbf{i}_l}(\tau), \tau\right) \mathrm{d}\tau \\ &= \mathbf{h}_{\mathbf{i}_l}(0) + \int_0^{t_e} \frac{t_l}{t_e} \boldsymbol{\alpha}\left(\mathbf{h}_{\mathbf{i}_l}(\frac{t_l}{t_e}s), \frac{t_l}{t_e}s\right) \mathrm{d}s, \end{aligned} \tag{12}$$

where the second line is obtained via a transform of the integration variable, $s = \frac{t_e}{t_l}\tau$. Accordingly, we can define $\widetilde{\mathbf{h}}_{\mathbf{i}_l}(s) = \mathbf{h}_{\mathbf{i}_l}(\frac{t_l}{t_e}s)$, and the ODE system for $\widetilde{\mathbf{h}}_{\mathbf{i}_l}$ is given by

$$\begin{cases} \frac{\mathrm{d}\widetilde{\mathbf{h}}_{\mathbf{i}_l}(s)}{\mathrm{d}s} = \frac{t_l}{t_e}\boldsymbol{\alpha}(\widetilde{\mathbf{h}}_{\mathbf{i}_l}, \frac{t_l}{t_e}s) \\ \widetilde{\mathbf{h}}_{\mathbf{i}_l}(0) = \mathbf{h}_{\mathbf{i}_l}(0). \end{cases} \tag{13}$$

Obviously, we have $\widetilde{\mathbf{h}}_{\mathbf{i}_l}(t_e) = \mathbf{h}_{\mathbf{i}_l}(t_l)$. We then merge the modified ODE system (13) for every interaction in the mini-batch into one ODE system, where the state $\mathbf{h}$ is the concatenation of $\{\widetilde{\mathbf{h}}_{\mathbf{i}_l}|l \in \{n_1, \ldots, n_B\}\}$. In this way, we only need to solve $\mathbf{h}$ for one timestamp t_e, which is equivalent to solving B target states $\{\mathbf{h}_{\mathbf{i}_l}(t_l)\}$ simultaneously. We do not need to solve $\mathbf{h}$ for any additional timestamps. Therefore, the cost can be largely reduced. While the choice of the ending time t_e can be arbitrary, for convenience we simply set $t_e = 1$ in our implementation.

During the model estimation, except the sensitivity, all the other partial derivatives, such as $\frac{\partial f}{\partial \boldsymbol{\eta}}$ and $\frac{\partial f}{\partial m_{\mathbf{i}}(t)}$ can be conveniently and efficiently computed by automatic differential libraries, such as PyTorch (Paszke et al., 2019). Moreover, each data point only associates with a part of the model parameters and so many elements of the partial derivatives and sensitivities are zero. For example, the data point y_l is a noisy observation of the interaction $\mathbf{i}_l$'s result at timestamp t_l, namely $m_{\mathbf{i}_l}(t)$. This interaction result is calculated based on $\mathbf{v}_{\mathbf{i}_l} = \{\mathbf{u}^1_{i_{l_1}}, \ldots, \mathbf{u}^K_{i_{l_K}}\}$ — the representations of the participant objects in $\mathbf{i}_l$, and is irrelevant to the representations of other objects, *i.e.,* $\mathcal{U} \setminus \mathbf{v}_{\mathbf{i}_l}$; see (3). For the latter, the corresponding elements in the partial derivatives and sensitivity related to $m_{\mathbf{i}_l}$ are therefore zero. To further save the memory usage and computational cost, we use sparse matrices during the ODE solving and optimization. The model estimation is summarized in Algorithm 1.

Algorithm 1 THIS-ODE

Input: $\mathcal{D} = \{(\mathbf{i}_1, t_1, y_1), \cdots, (\mathbf{i}_N, t_N, y_N)\}$
Initialize: $\mathcal{U}$, $\boldsymbol{\theta}$ and other parameters
repeat
  Randomly sample a mini-batch $\mathcal{B}$ from $\mathcal{D}$.
  For each interaction $\mathbf{i}_l \in \mathcal{B}$, construct an augmented ODE system (13) that includes both the state and sensitivity.
  Merge all the ODEs into one and solve the joint state $\mathbf{h}$ to ending time t_e.
  Read out the sub-states $\{\mathbf{h}_{\mathbf{i}_l}(t_l) = \widetilde{\mathbf{h}}_{\mathbf{i}_l}(t_e)|\mathbf{i}_l \in \mathcal{B}\}$ from $\mathbf{h}(t_e)$ and compute the stochastic gradient $\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{\eta}}$ according to (11), (7), and (8).
  $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} + \gamma \cdot \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{\eta}}$
until the maximum number of epcohes has been finished or other stopping criteria are met

4.3. Algorithm Complexity

Our algorithm conducts stochastic mini-batch optimization to learn the representations of interaction objects $\mathcal{U}$, ODE parameters $\boldsymbol{\theta}$, and the other parameters. To do so, at each step, our algorithm samples a mini-batch of B observed interactions, solve a joint ODE system to time t_e, with which to compute the stochastic gradient and to update the model parameters. The joint ODE is obtained by merging B (augmented) ODE systems in the form of (13), each of which is for a particular interaction in the mini-batch and the dimension of state is $p + 1$, where p is the number of parameters. Hence, the state dimension of the joint system is $B(p+1)$. The time complexity is thereby $\mathcal{O}((p+1)BT)$, where T is the number of integration steps in the ODE solver. It is linear in the mini-batch size. The space complexity is $\mathcal{O}(B(p+1))$, which is to store the state at each step when solving the joint ODE.

5. Related Work

A variety of tensor decomposition methods have been proposed, such as (Tucker, 1966; Harshman, 1970; Chu and Ghahramani, 2009; Sutskever et al., 2009; Hoff, 2011; Kang et al., 2012; Yang and Dunson, 2013; Choi and Vishwanathan, 2014; Du et al., 2018; Fang et al., 2021a). Most

of these methods are based on a multilinear decomposition model, such as Tucker (Tucker, 1966) and CP (Harshman, 1970). To grasp complex, nonlinear relationships from data, recent works have developed nonlinear decomposition models based on Gaussian processes (GPs) or neural networks, such as (Xu et al., 2012; Zhe et al., 2016a;b; Liu et al., 2018; Pan et al., 2020b; Liu et al., 2019; Tillinghast and Zhe, 2021; Fang et al., 2021b; Tillinghast et al., 2022). To incorporate time information, existent methods usually use discrete time steps and augment the tensor with a time mode. *e.g.,* (Xiong et al., 2010; Rogers et al., 2013; Zhe et al., 2015; 2016b; Wu et al., 2019; Ahn et al., 2021; Du et al., 2018). To better capture the temporal dependencies, a dynamic transition model on the time step representations is often incorporated during the tensor decomposition. For example, Xiong et al. (2010) used a conditional Gaussian prior over the successive steps, Wu et al. (2019) used recurrent neural networks, and Ahn et al. (2021) used kernel smoothing and regularization. To support continuous-time decomposition, Zhang et al. (2021) modeled the coefficients in the CP decomposition ($\{\lambda_j\}_{j=1}^r$ in (1)) as a time-trend function, which are parameterized by polynomial splines.

Another line of research uses point processes to model interaction events (Schein et al., 2016; Zhe and Du, 2018; Pan et al., 2020a; Wang et al., 2020). The temporal dependencies are learned through the modeling of the event rate function. For example, Zhe and Du (2018) used Hawkes processes to estimate the local excitation effects between the events and Wang et al. (2020) constructed a self-adaptable point process to estimate both the excitation and inhibition effects. While valuable, these methods only care about the events. They do not leverage the observed interaction results nor can predict their values.

Our method is related to the recent neural ODE work (Chen et al., 2018), which also uses a neural network to model the ODE state derivative. However, our work differs in several aspects. First, about the motivation, our method aims to learn from temporal interaction sequences, while neural ODE is motivated to construct continuous-depth neural networks (to simplify NN architecture design, enhance the expressivity, *etc.*). Second, our method models many ODE systems altogether, where each ODE is for the sequence of one interaction and governed by the representations of the interaction objects. In this way, our method decomposes these sequences while neural ODE usually uses one ODE system to model the data and does not conduct decomposition. Third, our method uses forward sensitivity analysis for gradient computation, while neural ODE uses the adjoint state method (Pontryagin, 1987), which needs to construct a backward companion ODE and sequentially solve two systems — first solving the original ODE forward and then solving the adjoint ODE backward. Note that the adjoint method is efficient for *coupled*, multi-dimensional states, where evolving the sensitivity, *i.e.,* state Jacobian, takes the time complexity $\mathcal{O}(d^2p)$ where d is the state dimension and p is the number of parameters, while the adjoint method only takes $\mathcal{O}(dp)$ to compute the gradient. However, since the ODE states in our method are only scalar interaction results, *i.e.,* $d = 1$, and these states are non-coupled, the forward sensitivity enjoys the same complexity as the adjoint method, yet is more convenient to implement. The actual computation is even more efficient, because we do not need to sequentially solve two systems. The recent work (Heinonen et al., 2018) uses GPs and sensitivity analysis to learn ODEs from data; but it does not perform decomposition or estimate representations.

6. Experiment

6.1. Ablation Study: Spiral Interactions

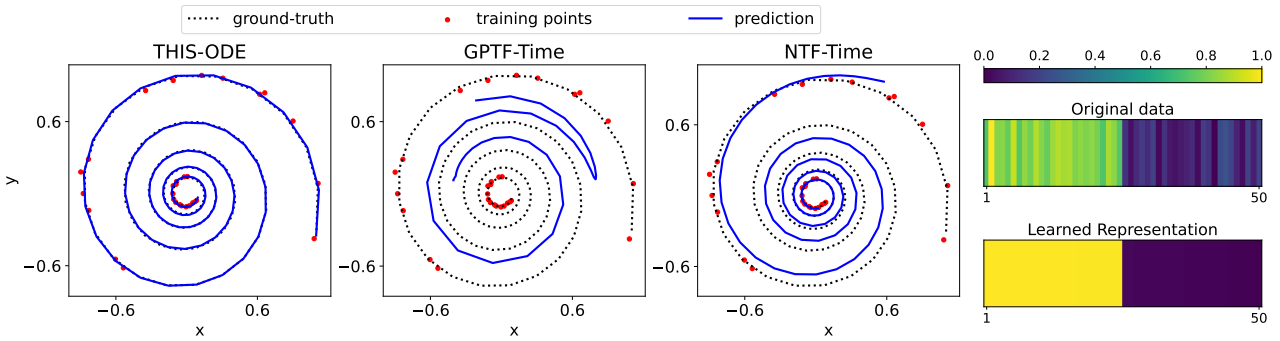
We first evaluated THIS-ODE on a synthetic task — decomposing interactions that form spiral curves. A spiral curve is controlled by two dynamic parameters: the radius $r(t)$ and angle $\theta(t)$, which were simulated via different interactions. Specifically, we considered interactions among two types of objects. Each type consists of 50 objects. We generated a scalar representation for each object. For the first half of the objects in type I, their representations $\{u_j^1 | 1 \le j \le 25\}$ were sampled from Unifrom$(0.45, 0.65)$ and the second half from Uniform$(1.0, 1.2)$. The representations of the first half of the objects in type II, $\{u_j^2 | 1 \le j \le 25\}$, were sampled from $\mathcal{N}(0.5, 0.1)$ and the remaining from $\mathcal{N}(-0.5, 0.1)$. The trajectory for a specific interaction $\mathbf{i} = (i_1, i_2)$ is generated by

$$m_{\mathbf{i}}(t) = \left(u_{i_1}^1 \exp(-0.5t)\right)^{\mathbb{1}(i_1+i_2 \mod 2=0)} \cdot \left(u_{i_2}^2 + 2\pi t\right)^{\mathbb{1}(i_1+i_2 \mod 2=1)}, \quad (14)$$

where $\mathbb{1}(\cdot)$ is the indicator function. When $i_1 + i_2$ is even, the interaction simulates a radius $r(t)$, which is an exponential decaying function, while when $i_1 + i_2$ is odd, the interaction simulates the angle $\theta(t)$, which is a linear function. The radius and angle are determined by the corresponding representations, $u_{i_1}^1$ or $u_{i_2}^2$. We can combine any radius and angle pair to form a spiral $(x(t) = r(t)\cos(\theta(t)), y(t) = r(t)\sin(\theta(t)))$, which are parameterized by different representations. For each interaction $\mathbf{i}$, we generated the interaction result $m_{\mathbf{i}}(t)$ at 50 timestamps uniformly sampled from $[0, 5]$. We obtained $125K$ data points in total.

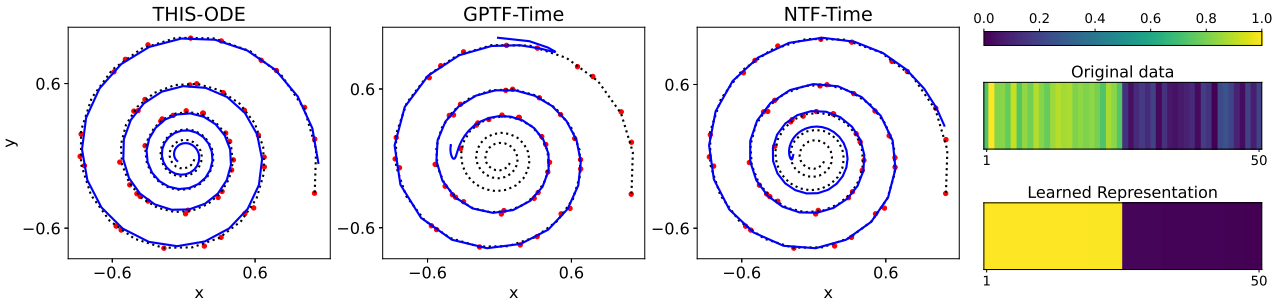
We then examined THIS-ODE in two settings. One is *interpolation*, where we used the first $1/3$ and last $1/3$ timestamps for training, and predicted on the $1/3$ timestamps in the middle. The other is *extrapolation*, where we only used the first $1/2$ timestamps for training, and predicted on the remaining $1/2$ timestamps.

We compared with (1) GPTF-time (Zhe et al., 2016b), a Gaussian process tensor factorization approach adjusted



(a) Recovered Spiral in the *interpolation* experiment. Radius: $m_{(2,2)}(t)$; Angle: $m_{(1,2)}(t)$.

(b) Structures in *interpolation*.



(c) Recovered Spiral in the *extrapolation* experiment. Radius: $m_{(2,2)}(t)$; Angle: $m_{(1,2)}(t)$.

(d) Structures in *extrapolation*.

Figure 2. Examples of recovered spirals (a, c) and the learned representations for type II objects (b, d) where colors indicate values.

to continuous time, where the time t is plugged into the kernel to sample the entry value as a function of both the representations and time, *i.e.,* $m_{\mathbf{i}} = g(\mathbf{u}^1_{i_1}, \ldots, \mathbf{u}^K_{i_K}, t)$ and $g \sim \mathcal{GP}(0, \kappa(\cdot,\cdot))$. (2) NTF-time, a neural tensor factorization model (Liu et al., 2019) with the time t as an additional input, and a neural network is used to model g. All these methods were implemented with PyTorch (Paszke et al., 2019). For our method THIS-ODE, we used the Torchdiffeq library[1] to solve the ODEs, with the explicit Runge-Kutta method of order 5 and a fixed step-size 10^{-4}. For the initial state, we simply used the CP form for β (see (3)). Following (Zhe et al., 2016b), we used the Square-Exponential (SE) kernel for GPTF-time and sparse variational GP approximation with 50 pseudo inputs for efficient inference. For both NTF-time and THIS-ODE, we used one layer neural network with 50 neurons and `tanh` activation. We ran all the methods with ADAM optimization (Kingma and Ba, 2014) with learning rate 10^{-3}. We ran 500 epochs, which is sufficient for convergence.

As shown in Table 1, the prediction error of THIS-ODE is much smaller than GPTF-time[2] and NTF-time in both the interpolation and extrapolation settings, showing that our method can much better capture the hidden dynamics in data. To showcase the prediction result, we examined the recovered spirals by each method, where the angle is determined by interaction $(1, 2)$ while the radius interaction $(2, 2)$. As shown in Fig. 2 a and c, the predicted spiral by THIS-ODE almost overlaps with the ground-truth, no matter if the training points are absent in the middle (interpolation) or in the inner long tail (extrapolation). In the interpolation

	Interpolation	Extrapolation
GPTF-Time	0.5557	0.9032
NTF-Time	0.1004	0.3656
THIS-ODE	**0.0148**	**0.0746**

Table 1. Root Mean Square Error (RMSE) of predicting dynamic interactions that form spiral curves. The time range is $[0, 5]$.

	State	Sensitivity
RK5(w/o TA)	$1.2 \pm 0.07 \times 10^{-4}$	$4.1 \pm 0.3 \times 10^{-5}$
RK5-ADP(w/o TA)	$1.0 \pm 0.1 \times 10^{-4}$	$4.0 \pm 0.9 \times 10^{-5}$
RK5(w/ TA)	$1.1 \pm 0.06 \times 10^{-4}$	$4.2 \pm 0.3 \times 10^{-5}$
RK5-ADP(w/ TA)	$1.1 \pm 0.1 \times 10^{-4}$	$4.3 \pm 0.6 \times 10^{-5}$

Table 2. Relative error of the state and sensitivity. "w/o TA" means without time alignment and "w/ TA" means with time alignment. The original time range is $[0, 5]$, and the range after time alignment is $[0, 1]$. The results were averaged over ten runs.

[1] https://github.com/rtqichen/torchdiffeq

[2] We found that GPTF-time failed when jointly learning the representations and function g. This might be due to the disrupt changes of the interaction types across their indices; see (14). To obtain a reasonable result, we fixed the representations as their ground-truth values for GPTF-time (but not for the other methods).

setting, the predictions of GPTF-time and NTF-time deviate from the ground-truth spiral quite much, exhibiting inferior accuracy. Although in the extrapolation setting, both GPTF-time and NTF-time can well predict the spiral in the region that covers the training points, their predictions do not extend to more distant regions to recover the inner long tail of the ground-truth, showing a failure to extrapolate. To give more details, we show the prediction of each interaction separately (*i.e.,* radius and angle) in the Appendix (Figure 3, 4). We then examined the representation learning results. In Fig. 2 b and d, we show the learned representations by THIS-ODE for the type II objects, *i.e.,* $\{u_j^2|1 \leq j \leq 50\}$, and the original data. For a better contrast, both the learned representations and the original data were normalized to $[0, 1]$. As we can see, these representations accurately reflect the hidden clusters of the objects in the original data: object 1-25 are in the first cluster and 26-50 in the second cluster. It demonstrates that our method can discover the structural knowledge underlying the data and encode this knowledge into the learned representations.

We then verified the accuracy of the proposed time alignment method (see Sec. 4.2). We randomly set all the model parameters, and then used the explicit RK method of order 5 with a fixed step-size 10^{-4} (RK5), and with the adaptive step-size (RK5-ADP). We ran each method with the time alignment and without time alignment, to compute the state and sensitivity (*i.e.,* state gradient) at the end time. We then used the explicit, adaptive step-size RK method of order 8, without the time alignment, to compute the result as the ground-truth. We measured the relative error of the states and sensitivity. We repeated the test for ten times, and report the average error and standard deviation in Table 2. We can see that for both solvers, applying the time alignment method gives almost the same relative error as not applying that method. This confirms our time alignment method almost has no effect on the accuracy of the solvers. But our method saves much computation (see Sec. 4.2).

6.2. Real-World Applications

Datasets. Next, we examined THIS-ODE in four real-world applications. (1) *Fit Record*[3], workout logs of EndoMondo users in outdoor exercises. We extracted three-way (user, sport-type, altitude-level) interactions among 500 users, 20 sport types, and 50 attitudes. The interaction result is the user' heart rate during the exercise, which was measured for 4000 to 6000 seconds. In total, we collected 50K heart rates and their timestamps. (2) *Beijing Air*[4], a two-way interaction dataset that records the concentration of 6 pollutants across 12 districts in Beijing between year 2013 and 2017. The concentration was measures hourly. The original dataset includes 2.45M measurements. We randomly sampled 15K continuous measurements and their timestamps for training and testing. (3) *Server Room*[5], temperature data of Poznan Supercomputing and Networking Center, under different air-conditional modes (24, 27, and 30 Celsius degrees), server power usage levels (50%, 75%, and 100%) at 34 locations. We thereby extracted three way interactions (air conditional mode, power usage level, location), and interaction result is the temperature along with time. We collected 25K temperature records and their timestamps. (4) *Indoor Condition*[6], house conditions data monitored by wireless sensor networks. We extracted two-way interactions (room, ambient condition). There are 9 rooms, such as living room and kitchen area, and 2 ambient conditions: humidity and temperature. The sensors measured the value of each interaction pair every 10 minutes. We collected 25K interaction results and their timestamps.

Methods. We compared with following typical and/or state-of-the-art temporal decomposition approaches. (1) GPTF-Time and (2) NTF-time as mentioned in Sec. 6.1, (3) CP-Time (Zhang et al., 2021) that uses polynomial splines to estimate time-varying coefficients $\boldsymbol{\lambda}$ in the CP decomposition (see (1)), (4) GPTF-DTL, GPTF with discrete time steps and linear dynamics between the time steps. Specifically, the observed interaction results were organized by a tensor plus one time mode. For decomposition, we placed a conditional Gaussian prior over time-step representations, $p(\mathbf{t}_{j+1}|\mathbf{t}_j) = \mathcal{N}(\mathbf{t}_{j+1}|\mathbf{A}\mathbf{t}_j + \mathbf{b}, v\mathbf{I})$. This is similar to (Xiong et al., 2010), but more general because their prior corresponds to $\mathbf{A} = \mathbf{I}$ and $\mathbf{b} = \mathbf{0}$. (5) GPTF-DTN, GPTF with discrete time steps and nonlinear dynamics. It is similar to GPTF-DTL, except that conditional Gaussian prior is $p(\mathbf{t}_{j+1}|\mathbf{t}_j) = \mathcal{N}(\mathbf{t}_{j+1}|\sigma(\mathbf{A}\mathbf{t}_j + \mathbf{b}), v\mathbf{I})$ where $\sigma(\cdot)$ is a nonlinear activation function. Hence it fulfills an RNN-like dynamic model. (6) NTF-DTL and (7) NTF-DTN, which are similar to GPTF-DTL and GPTF-DTN, respectively, except the decomposition model switches to DTN. (8) CP-DTL and (9) CP-DTN, similar to GPTF-DTL and GPTF-DTN, except the decomposition model is CP. (10) PTucker (Oh et al., 2018), an efficient parallel Tucker decomposition on the tensor augmented with a discrete time mode.

Settings. All the methods were implemented with PyTorch. For GP based approaches, *i.e.,* GPTF-{Time, DTL, DTN}, we again followed (Zhe et al., 2016b) to use SE kernel and sparse variational approximations to address the computation hurdle caused by large kernel matrices. For CP-Time, the number of knots for the polynomial splines was

[3] https://cseweb.ucsd.edu/~jmcauley/datasets.html#endomondo

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

[5] https://zenodo.org/record/3610078#.YeEHmljMLAx

[6] https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction

Interpolation	*Beijing Air*	*Indoor Condition*	*Server Room*	*Fit Record*
CP-Time	0.897 ± 0.012	0.780 ± 0.012	0.998 ± 0.006	1.021 ± 0.030
CP-DTL	0.898 ± 0.015	0.842 ± 0.003	0.754 ± 0.003	0.721 ± 0.076
CP-DTN	0.833 ± 0.003	0.889 ± 0.005	0.685 ± 0.007	0.688 ± 0.044
GPTF-Time	0.711 ± 0.011	0.849 ± 0.005	0.505 ± 0.116	0.794 ± 0.013
GPTF-DTL	0.686 ± 0.045	0.852 ± 0.004	0.592 ± 0.119	0.640 ± 0.035
GPTF-DTN	0.670 ± 0.062	0.713 ± 0.104	0.373 ± 0.025	0.642 ± 0.043
NTF-Time	0.745 ± 0.095	0.800 ± 0.009	0.739 ± 0.007	0.705 ± 0.014
NTF-DTL	0.757 ± 0.006	0.777 ± 0.018	0.733 ± 0.004	0.725 ± 0.048
NTF-DTN	0.686 ± 0.011	0.665 ± 0.004	0.346 ± 0.049	0.685 ± 0.037
PTucker	0.959 ± 0.015	0.806 ± 0.027	0.916 ± 0.008	0.727 ± 0.037
THIS-ODE	$\mathbf{0.624 \pm 0.008}$	$\mathbf{0.618 \pm 0.007}$	$\mathbf{0.246 \pm 0.009}$	$\mathbf{0.615 \pm 0.024}$
Extrapolation				
CP-Time	0.863 ± 0.022	0.867 ± 0.010	1.082 ± 0.005	0.958 ± 0.061
CP-DTL	0.553 ± 0.005	0.527 ± 0.006	0.545 ± 0.030	0.595 ± 0.026
CP-DTN	0.557 ± 0.004	0.584 ± 0.009	0.340 ± 0.003	0.637 ± 0.050
GPTF-Time	0.527 ± 0.018	0.489 ± 0.011	0.280 ± 0.007	0.634 ± 0.089
GPTF-DTL	0.577 ± 0.035	0.506 ± 0.013	0.234 ± 0.010	0.576 ± 0.024
GPTF-DTN	0.511 ± 0.002	0.489 ± 0.003	0.218 ± 0.002	0.572 ± 0.025
NTF-Time	0.537 ± 0.002	0.510 ± 0.027	0.275 ± 0.022	0.621 ± 0.026
NTF-DTL	0.512 ± 0.009	0.593 ± 0.079	0.269 ± 0.005	0.659 ± 0.031
NTF-DTN	0.513 ± 0.003	0.484 ± 0.011	0.247 ± 0.010	0.573 ± 0.030
PTucker	0.522 ± 0.022	0.749 ± 0.006	0.600 ± 0.002	0.722 ± 0.030
THIS-ODE	$\mathbf{0.498 \pm 0.013}$	$\mathbf{0.460 \pm 0.004}$	$\mathbf{0.215 \pm 0.002}$	$\mathbf{0.568 \pm 0.029}$

Table 3. Root Mean Square error (RMSE) with $r = 3$. The results were averaged over five runs.

100. For nonlinear dynamic models ({GPTF, NTF, CP}-DTN), we used `tanh` activation. For all the discrete time methods, we partitioned the total time span into 50 (equal-length) steps. Like in Sec. 6.1, we used one-layer NN for THIS-ODE and NTF based methods, with 50 neurons and `tanh` activation. We used ADAM to run stochastic mini-batch optimization for all the methods, where the mini-batch size was 100 and the learning rate was selected from $\{10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}\}$. For numerical stability, we re-scaled the timestamps to $[0, 10]$ for all the datasets. We ran every method for 500 epochs, which guarantees the convergence. We varied the dimension of the representations r from $\{1, 2, 3, 5, 7\}$. Similar to Sec. 6.1, we evaluated all the methods in two settings, *Interpolation* and *Extrapolation*. For *interpolation*, We randomly sampled 80% interactions and used their first $1/3$ and last $1/3$ interaction results for training, and then tested on the remaining interaction results in the middle. For extrapolation, we used the first $1/2$ interaction results for training, and tested on remaining half. We repeated the experiments for five times, and computed the average root mean-square error (RMSE) of each method.

Results. Due to the space limit, here we only list the prediction accuracy of each method for $r = 3$, as in Table 3. We list the other results in Appendix (see Table 4-7). In all the cases, THIS-ODE always outperforms the competing approaches by a large margin (except when $r = 2$, GPTF-time is slightly better than THIS-ODE: 0.521 *vs.* 0.524, for *interpolation* on *Beijing Air* dataset; see Table 5 in Appendix). The improvements obtained by THIS-ODE are large and significant ($p < 0.05$). The results together have demonstrated the advantage of THIS-ODE in predicting long-term interaction results, which can be important for temporal data analysis and predictive tasks.

7. Conclusion

We have presented THIS-ODE, an ODE based approach to decompose high-order interactions sequences. THIS-ODE is robust and expressive to capture complex unknown dynamics from data for better representation learning. Compared with the existing methods, it shows significant improvement in predicting long-term interaction results, in both interpolation and extrapolation cases.

Acknowledgments

This work has been supported by MURI AFOSR grant FA9550-20-1-0358, NSF IIS-1910983 and NSF CAREER Award IIS-2046295.

References

Ahn, D., Jang, J.-G., and Kang, U. (2021). Time-aware tensor decomposition for sparse tensors. Machine Learning, pages 1–22.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2018). Neural ordinary differential equations. arXiv preprint arXiv:1806.07366.

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

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

Dormand, J. R. and Prince, P. J. (1980). A family of embedded runge-kutta formulae. Journal of computational and applied mathematics, 6(1):19–26.

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

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

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

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

Heinonen, M., Yildiz, C., Mannerström, H., Intosalmi, J., and Lähdesmäki, H. (2018). Learning unknown ODE models with Gaussian processes. In International Conference on Machine Learning, pages 1959–1968. PMLR.

Hoff, P. (2011). Hierarchical multilinear models for multiway data. Computational Statistics & Data Analysis, 55:530–543.

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

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

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

Liu, B., He, L., Li, Y., Zhe, S., and Xu, Z. (2018). Neuralcp: Bayesian multiway data analysis with neural tensor decomposition. Cognitive Computation, 10(6):1051–1061.

Liu, H., Li, Y., Tsang, M., and Liu, Y. (2019). CoSTCo: A Neural Tensor Completion Model for Sparse Tensors, page 324–334. Association for Computing Machinery, New York, NY, USA.

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

Pan, Z., Wang, Z., and Zhe, S. (2020a). Scalable nonparametric factorization for high-order interaction events. In International Conference on Artificial Intelligence and Statistics, pages 4325–4335. PMLR.

Pan, Z., Wang, Z., and Zhe, S. (2020b). Streaming nonlinear bayesian tensor decomposition. In Conference on Uncertainty in Artificial Intelligence, pages 490–499. PMLR.

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

Pontryagin, L. S. (1987). Mathematical theory of optimal processes. CRC press.

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

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

Schein, A., Zhou, M., Blei, D. M., and Wallach, H. (2016). Bayesian poisson tucker decomposition for learning the structure of international relations. In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML'16, pages 2810–2819. JMLR.org.

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

Tillinghast, C., Wang, Z., and Zhe, S. (2022). Nonparametric sparse tensor factorization with hierarchical Gamma processes. In International Conference on Machine Learning. PMLR.

Tillinghast, C. and Zhe, S. (2021). Nonparametric decomposition of sparse tensors. In International Conference on Machine Learning, pages 10301–10311. PMLR.

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

Wang, Z., Chu, X., and Zhe, S. (2020). Self-modulating nonparametric event-tensor factorization. In International Conference on Machine Learning, pages 9857–9867. PMLR.

Wu, X., Shi, B., Dong, Y., Huang, C., and Chawla, N. V. (2019). Neural tensor factorization for temporal interaction learning. In Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, pages 537–545.

Xiong, L., Chen, X., Huang, T.-K., Schneider, J., and Carbonell, J. G. (2010). Temporal collaborative filtering with bayesian probabilistic tensor factorization. In Proceedings of the 2010 SIAM International Conference on Data Mining, pages 211–222. SIAM.

Xu, Z., Yan, F., and Qi, Y. A. (2012). Infinite tucker decomposition: Nonparametric bayesian models for multiway data analysis. In ICML.

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

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

Zhe, S. and Du, Y. (2018). Stochastic nonparametric event-tensor decomposition. In Advances in Neural Information Processing Systems, pages 6856–6866.

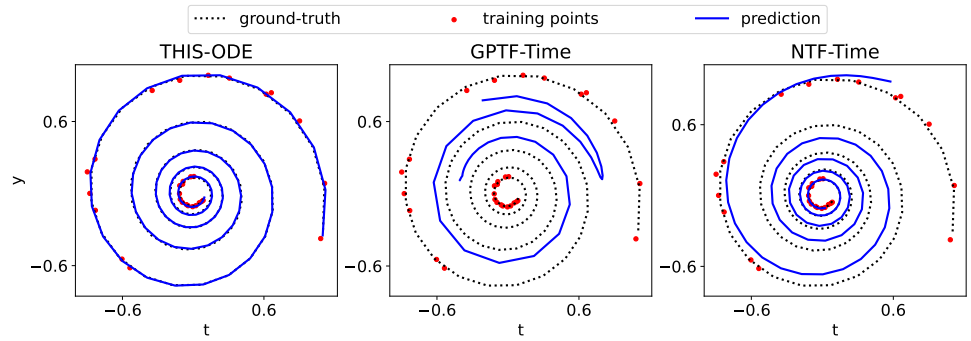
Zhe, S., Qi, Y., Park, Y., Xu, Z., Molloy, I., and Chari, S. (2016a). Dintucker: Scaling up gaussian process models on large multidimensional arrays. In Thirtieth AAAI conference on artificial intelligence.

Zhe, S., Xu, Z., Chu, X., Qi, Y., and Park, Y. (2015). Scalable nonparametric multiway data analysis. In Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, pages 1125–1134.

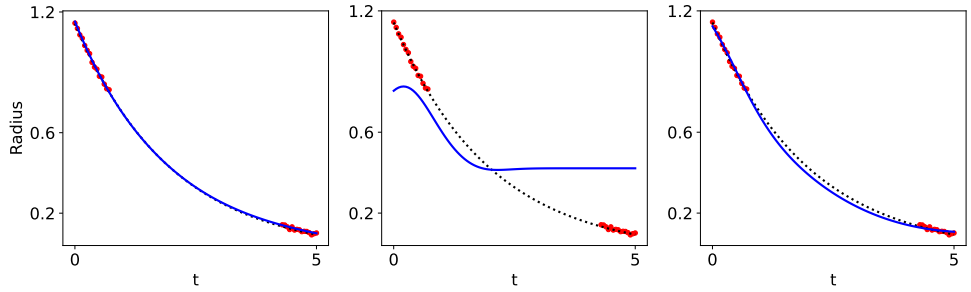
Zhe, S., Zhang, K., Wang, P., Lee, K.-c., Xu, Z., Qi, Y., and Ghahramani, Z. (2016b). Distributed flexible nonlinear tensor factorization. In Advances in Neural Information Processing Systems, pages 928–936.

A. More Plots for Ablation Study

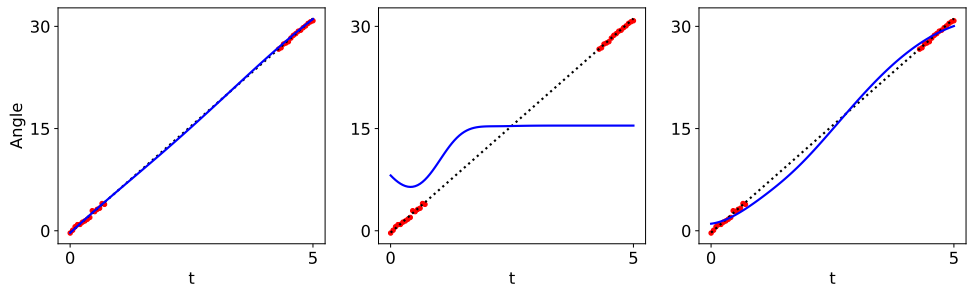
B. Prediction Accuracy for Real-World Applications with $r = 1, 2, 5, 7$



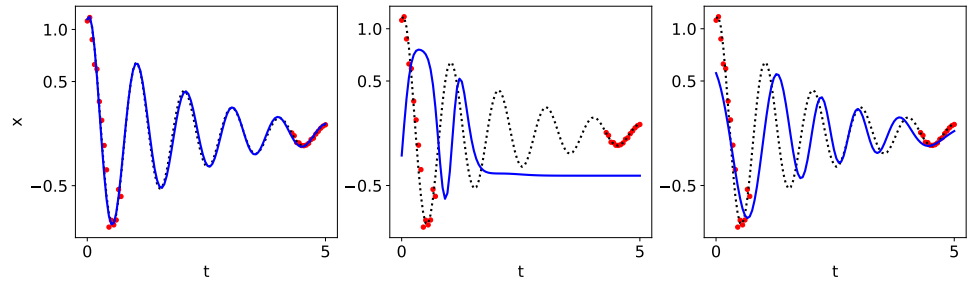
(a) Recovered Spiral in the *interpolation* experiment. Radius: $m_{(2,2)}(t)$; Angle: $m_{(1,2)}(t)$.



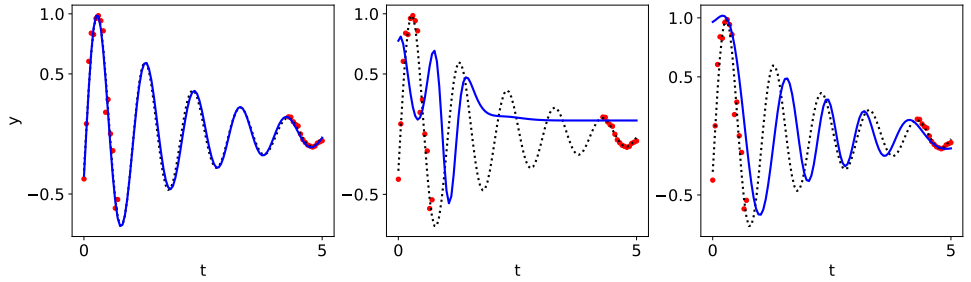
(b) Recovered Radius: $m_{(2,2)}(t)$



(c) Angle: $m_{(1,2)}(t)$.

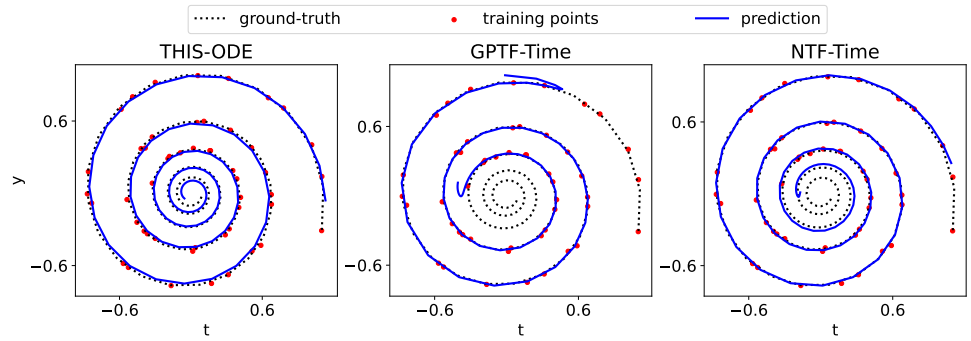


(d) $x(t)$ projected from the recovered spiral

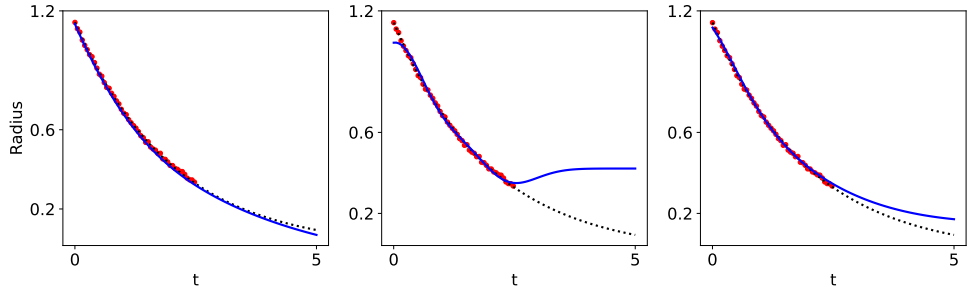


(e) $y(t)$ projected from the recovered spiral

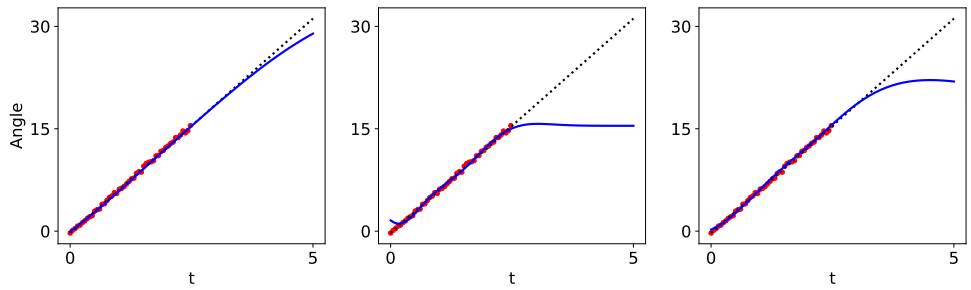
Figure 3. Examples of recovered spirals in the *interpolation* experiment.



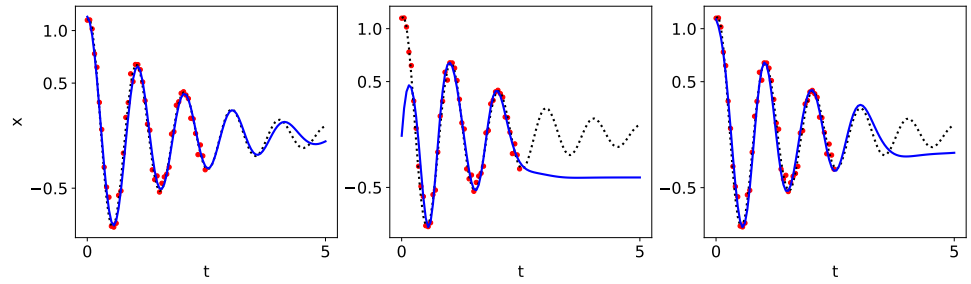
(a) Recovered Spiral in the *Extrapolation* experiment. Radius: $m_{(2,2)}(t)$; Angle: $m_{(1,2)}(t)$.



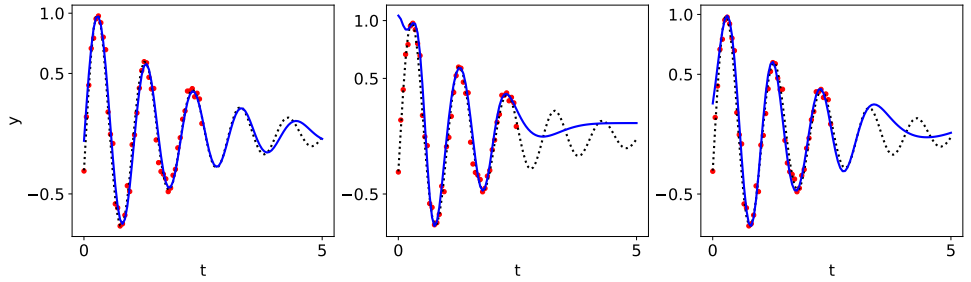
(b) Recovered Radius: $m_{(2,2)}(t)$



(c) Angle: $m_{(1,2)}(t)$.



(d) $x(t)$ projected from the recovered spiral



(e) $y(t)$ projected from the recovered spiral

Figure 4. Examples of recovered spirals in the *extrapolation* experiment.

Interpolation	*Beijing Air*	*Indoor Condition*	*Server Room*	*Fit Record*
CP-Time	0.863 ± 0.021	0.862 ± 0.008	1.082 ± 0.005	0.958 ± 0.062
CP-DTL	0.861 ± 0.021	0.591 ± 0.001	0.775 ± 0.005	0.655 ± 0.074
CP-DTN	0.847 ± 0.026	0.595 ± 0.001	0.771 ± 0.002	0.655 ± 0.069
GPTF-Time	0.620 ± 0.006	0.806 ± 0.012	0.623 ± 0.012	0.886 ± 0.058
GPTF-DTL	0.616 ± 0.001	0.815 ± 0.006	0.612 ± 0.012	0.610 ± 0.047
GPTF-DTN	0.571 ± 0.049	0.770 ± 0.019	0.306 ± 0.005	0.601 ± 0.018
NTF-Time	0.615 ± 0.013	0.659 ± 0.003	0.610 ± 0.011	0.583 ± 0.017
NTF-DTL	0.624 ± 0.012	0.662 ± 0.003	0.625 ± 0.009	0.588 ± 0.018
NTF-DTN	0.610 ± 0.013	0.512 ± 0.006	0.226 ± 0.004	0.588 ± 0.016
PTucker	0.656 ± 0.008	0.643 ± 0.003	0.859 ± 0.002	0.769 ± 0.068
THIS-ODE	$\mathbf{0.563 \pm 0.005}$	$\mathbf{0.487 \pm 0.004}$	$\mathbf{0.222 \pm 0.006}$	$\mathbf{0.574 \pm 0.027}$
Extrapolation				
CP-Time	0.896 ± 0.012	0.890 ± 0.005	1.006 ± 0.004	1.021 ± 0.030
CP-DTL	0.888 ± 0.012	0.886 ± 0.005	0.821 ± 0.002	0.682 ± 0.048
CP-DTN	0.887 ± 0.013	0.890 ± 0.004	0.795 ± 0.003	0.682 ± 0.049
GPTF-Time	0.894 ± 0.011	0.890 ± 0.004	0.764 ± 0.004	0.799 ± 0.031
GPTF-DTL	0.888 ± 0.012	0.889 ± 0.003	0.765 ± 0.010	0.668 ± 0.022
GPTF-DTN	0.872 ± 0.004	0.757 ± 0.007	0.504 ± 0.009	0.669 ± 0.020
NTF-Time	0.770 ± 0.013	0.723 ± 0.007	0.757 ± 0.005	0.669 ± 0.029
NTF-DTL	0.746 ± 0.006	0.905 ± 0.016	0.722 ± 0.083	0.663 ± 0.021
NTF-DTN	0.737 ± 0.005	0.694 ± 0.002	0.542 ± 0.003	0.695 ± 0.021
PTucker	1.211 ± 0.023	0.889 ± 0.005	0.862 ± 0.001	0.747 ± 0.031
THIS-ODE	$\mathbf{0.710 \pm 0.017}$	$\mathbf{0.655 \pm 0.021}$	$\mathbf{0.407 \pm 0.029}$	$\mathbf{0.644 \pm 0.019}$

Table 4. Root Mean Square error (RMSE) with $r = 1$. The results were averaged over five runs.

Interpolation	*Beijing Air*	*Indoor Condition*	*Server Room*	*Fit Record*
CP-Time	0.860 ± 0.018	0.867 ± 0.010	1.082 ± 0.005	0.989 ± 0.054
CP-DTL	0.739 ± 0.005	0.527 ± 0.005	0.357 ± 0.002	0.618 ± 0.037
CP-DTN	0.661 ± 0.007	0.578 ± 0.015	0.352 ± 0.006	0.688 ± 0.020
GPTF-Time	0.597 ± 0.010	0.548 ± 0.009	0.300 ± 0.071	0.876 ± 0.046
GPTF-DTL	0.572 ± 0.024	0.557 ± 0.016	0.288 ± 0.005	0.577 ± 0.043
GPTF-DTN	$\mathbf{0.521 \pm 0.025}$	0.780 ± 0.014	0.242 ± 0.005	0.579 ± 0.045
NTF-Time	0.569 ± 0.008	0.481 ± 0.009	0.266 ± 0.007	0.632 ± 0.018
NTF-DTL	0.540 ± 0.008	0.486 ± 0.005	0.256 ± 0.005	0.655 ± 0.011
NTF-DTN	0.531 ± 0.013	0.485 ± 0.003	0.212 ± 0.003	0.597 ± 0.034
PTucker	0.566 ± 0.018	0.658 ± 0.005	0.586 ± 0.012	0.691 ± 0.041
THIS-ODE	0.524 ± 0.006	$\mathbf{0.467 \pm 0.004}$	$\mathbf{0.194 \pm 0.001}$	$\mathbf{0.564 \pm 0.036}$
Extrapolation				
CP-Time	0.899 ± 0.014	0.802 ± 0.003	0.998 ± 0.006	1.021 ± 0.030
CP-DTL	0.827 ± 0.011	0.872 ± 0.007	0.726 ± 0.005	0.680 ± 0.017
CP-DTN	0.853 ± 0.008	0.890 ± 0.005	0.777 ± 0.007	0.667 ± 0.025
GPTF-Time	0.754 ± 0.019	0.888 ± 0.004	0.497 ± 0.217	0.655 ± 0.029
GPTF-DTL	0.795 ± 0.055	0.889 ± 0.016	0.666 ± 0.065	0.653 ± 0.034
GPTF-DTN	0.723 ± 0.041	0.778 ± 0.036	0.564 ± 0.007	0.762 ± 0.042
NTF-Time	0.904 ± 0.016	0.696 ± 0.030	0.607 ± 0.011	0.648 ± 0.036
NTF-DTL	0.880 ± 0.011	0.727 ± 0.016	0.782 ± 0.014	0.707 ± 0.032
NTF-DTN	0.697 ± 0.003	0.684 ± 0.004	0.567 ± 0.025	0.688 ± 0.060
PTucker	0.895 ± 0.012	1.016 ± 0.000	0.917 ± 0.023	0.741 ± 0.032
THIS-ODE	$\mathbf{0.675 \pm 0.010}$	$\mathbf{0.679 \pm 0.002}$	$\mathbf{0.402 \pm 0.013}$	$\mathbf{0.629 \pm 0.003}$

Table 5. Root Mean Square error (RMSE) with $r = 2$. The results were averaged over five runs.

Interpolation	*Beijing Air*	*Indoor Condition*	*Server Room*	*Fit Record*
CP-Time	0.864 ± 0.022	0.867 ± 0.010	1.083 ± 0.005	0.957 ± 0.061
CP-DTL	0.525 ± 0.014	0.489 ± 0.003	0.540 ± 0.019	0.602 ± 0.022
CP-DTN	0.559 ± 0.004	0.638 ± 0.001	0.259 ± 0.002	0.687 ± 0.057
GPTF-Time	0.526 ± 0.009	0.497 ± 0.005	0.279 ± 0.004	0.726 ± 0.099
GPTF-DTL	0.536 ± 0.012	0.488 ± 0.003	0.245 ± 0.011	0.587 ± 0.023
GPTF-DTN	0.515 ± 0.007	0.485 ± 0.004	0.228 ± 0.005	0.579 ± 0.024
NTF-Time	0.514 ± 0.011	0.473 ± 0.007	0.264 ± 0.043	0.631 ± 0.013
NTF-DTL	0.505 ± 0.011	0.506 ± 0.008	0.258 ± 0.004	0.641 ± 0.039
NTF-DTN	0.498 ± 0.026	0.486 ± 0.003	0.214 ± 0.006	0.576 ± 0.013
PTucker	0.549 ± 0.001	0.654 ± 0.005	0.703 ± 0.004	0.838 ± 0.062
THIS-ODE	$\mathbf{0.497 \pm 0.010}$	$\mathbf{0.461 \pm 0.004}$	$\mathbf{0.195 \pm 0.001}$	$\mathbf{0.538 \pm 0.021}$
Extrapolation				
CP-Time	0.897 ± 0.011	0.679 ± 0.022	0.998 ± 0.006	1.020 ± 0.030
CP-DTL	0.899 ± 0.013	0.780 ± 0.097	0.794 ± 0.012	0.729 ± 0.053
CP-DTN	0.800 ± 0.011	0.889 ± 0.005	0.701 ± 0.148	0.807 ± 0.030
GPTF-Time	0.729 ± 0.052	0.820 ± 0.000	0.469 ± 0.073	0.752 ± 0.053
GPTF-DTL	0.696 ± 0.026	0.732 ± 0.015	0.406 ± 0.074	0.640 ± 0.031
GPTF-DTN	0.665 ± 0.010	0.633 ± 0.007	0.393 ± 0.016	0.648 ± 0.028
NTF-Time	0.765 ± 0.001	0.719 ± 0.068	0.805 ± 0.012	0.678 ± 0.048
NTF-DTL	0.799 ± 0.029	0.901 ± 0.004	0.746 ± 0.022	0.700 ± 0.036
NTF-DTN	0.666 ± 0.013	0.642 ± 0.005	0.399 ± 0.040	0.693 ± 0.013
PTucker	0.922 ± 0.004	0.902 ± 0.002	1.357 ± 0.021	0.812 ± 0.099
THIS-ODE	$\mathbf{0.644 \pm 0.015}$	$\mathbf{0.618 \pm 0.002}$	$\mathbf{0.297 \pm 0.052}$	$\mathbf{0.616 \pm 0.032}$

Table 6. Root Mean Square error (RMSE) with $r = 5$. The results were averaged over five runs.

Interpolation	*Beijing Air*	*Indoor Condition*	*Server Room*	*Fit Record*
CP-Time	0.862 ± 0.021	0.866 ± 0.010	1.083 ± 0.005	0.959 ± 0.062
CP-DTL	0.519 ± 0.012	0.528 ± 0.009	0.566 ± 0.110	0.598 ± 0.021
CP-DTN	0.536 ± 0.006	0.619 ± 0.002	0.279 ± 0.010	0.708 ± 0.033
GPTF-Time	0.523 ± 0.001	0.508 ± 0.005	0.238 ± 0.004	0.735 ± 0.094
GPTF-DTL	0.524 ± 0.010	0.493 ± 0.016	0.242 ± 0.013	0.577 ± 0.016
GPTF-DTN	0.519 ± 0.015	0.483 ± 0.003	0.283 ± 0.004	0.589 ± 0.011
NTF-Time	0.512 ± 0.009	0.479 ± 0.003	0.274 ± 0.015	0.608 ± 0.034
NTF-DTL	0.504 ± 0.016	0.622 ± 0.111	0.334 ± 0.008	0.634 ± 0.027
NTF-DTN	0.510 ± 0.011	0.466 ± 0.004	0.209 ± 0.000	0.578 ± 0.019
PTucker	0.539 ± 0.008	0.693 ± 0.005	0.908 ± 0.003	1.044 ± 0.118
THIS-ODE	$\mathbf{0.496 \pm 0.014}$	$\mathbf{0.457 \pm 0.003}$	$\mathbf{0.191 \pm 0.002}$	$\mathbf{0.525 \pm 0.026}$
Extrapolation				
CP-Time	0.896 ± 0.011	0.799 ± 0.000	0.998 ± 0.006	1.021 ± 0.030
CP-DTL	0.898 ± 0.013	0.665 ± 0.037	0.774 ± 0.017	0.696 ± 0.044
CP-DTN	0.770 ± 0.049	0.890 ± 0.005	0.399 ± 0.027	0.805 ± 0.006
GPTF-Time	0.730 ± 0.034	0.812 ± 0.000	0.650 ± 0.259	0.759 ± 0.050
GPTF-DTL	0.739 ± 0.005	0.703 ± 0.046	0.536 ± 0.074	0.635 ± 0.034
GPTF-DTN	0.656 ± 0.013	0.689 ± 0.019	0.352 ± 0.049	0.638 ± 0.033
NTF-Time	0.743 ± 0.055	0.848 ± 0.025	0.788 ± 0.016	0.693 ± 0.032
NTF-DTL	0.801 ± 0.013	0.765 ± 0.013	0.783 ± 0.006	0.695 ± 0.076
NTF-DTN	0.658 ± 0.011	0.639 ± 0.005	0.378 ± 0.026	0.691 ± 0.036
PTucker	1.192 ± 0.020	1.008 ± 0.018	1.822 ± 0.014	1.053 ± 0.112
THIS-ODE	$\mathbf{0.620 \pm 0.011}$	$\mathbf{0.609 \pm 0.015}$	$\mathbf{0.240 \pm 0.008}$	$\mathbf{0.605 \pm 0.039}$

Table 7. Root Mean Square error (RMSE) with $r = 7$. The results were averaged over five runs.