Machine Learning (2022) 111:1409–1430
https://doi.org/10.1007/s10994-021-06059-7

Time-aware tensor decomposition for sparse tensors

Dawon Ahn[1] · **Jun-Gi Jang**[1] · **U Kang**[1]

Received: 30 September 2020 / Revised: 8 July 2021 / Accepted: 6 August 2021 /
Published online: 27 September 2021

Abstract
Given a sparse time-evolving tensor, how can we effectively factorize it to accurately discover latent patterns? Tensor decomposition has been extensively utilized for analyzing various multi-dimensional real-world data. However, existing tensor decomposition models have disregarded the temporal property for tensor decomposition while most real-world data are closely related to time. Moreover, they do not address accuracy degradation due to the sparsity of time slices. The essential problems of how to exploit the temporal property for tensor decomposition and consider the sparsity of time slices remain unresolved. In this paper, we propose time-aware tensor decomposition (TATD), an accurate tensor decomposition method for sparse temporal tensors. TATD is designed to exploit time dependency and time-varying sparsity of real-world temporal tensors. We propose a new smoothing regularization with Gaussian kernel for modeling time dependency. Moreover, we improve the performance of TATD by considering time-varying sparsity. We design an alternating optimization scheme suitable for temporal tensor decomposition with our smoothing regularization. Extensive experiments show that TATD provides the state-of-the-art accuracy for decomposing temporal tensors.

Keywords Temporal tensor · Time-aware tensor decomposition · Time dependency · Kernel smoothing regularization · Time-varying sparsity

1 Introduction

Given a sparse temporal tensor where one mode denotes time, how can we discover its latent factors? A tensor, or multi-dimensional array, has been widely used to model multi-faceted relationships for time-evolving data. For example, air quality data (Zhang et al.,

Editors: João Gama, Alípio Jorge, Salvador García.

✉ U Kang
ukang@snu.ac.kr

Dawon Ahn
dawon@snu.ac.kr

Jun-Gi Jang
elnino4@snu.ac.kr

[1] Seoul National University, Seoul, South Korea

2017) containing measurements of contaminants collected from sensors at every time step are modeled as a 3-mode temporal tensor (time, site, contaminant).

Tensor decomposition is a fundamental building block for effectively analyzing tensors by revealing latent factors between entities (Kolda & Sun, 2008; Kang et al., 2012; Oh et al., 2018; Park et al., 2017), and it has been extensively utilized in various real-world applications across diverse domains including recommender systems (Symeonidis 2016), clustering (Sun et al., 2015), anomaly detection (Kolda & Sun, 2008), correlation analysis (Sun et al., 2006), network forensic (Maruhashi et al., 2011), and latent concept discovery (Kolda et al., 2005). CANDECOMP/PARAFAC (CP) (Harshman et al., 1970) decomposition is one of the most widely used tensor decomposition models, which factorizes a tensor into a set of factor matrices and a core tensor which is restricted to be diagonal.

Previous CP decomposition methods (Kolda et al., 2005; Sun et al., 2006, 2015; Kolda & Sun, 2008; Maruhashi et al., 2011; Matsubara et al., 2012; Kang et al., 2012; Symeonidis, 2016; de Araujo et al., 2017; Park et al., 2017; Oh et al., 2018) do not consider temporal property when it comes to factorizing tensors while most time-evolving tensors exhibit time dependency. Recently, Yu et al. (2016) have tried to address the above problem, but it considers only the past information of a time step to model the time dependency. A model needs to examine both the past and the future information to capture accurate temporal properties since information at each time point is heavily related to that in the past and the future. In addition, they do not consider the time-varying sparsity, one of the main properties in real-world temporal tensors. The main challenges to design an accurate temporal tensor decomposition method for sparse temporal tensors are (1) how to harness the time dependency in real-world data, and (2) how to exploit the varying sparsity of time slices.

In this paper, we propose Time-Aware Tensor Decomposition (TATD), a time-aware tensor decomposition method for analyzing real-world temporal tensors. Our main observation is that values of adjacent time slices are mostly similar to each other since time slices in a temporal tensor are closely related to each other. Based on this observation, TATD employs a kernel smoothing regularization to make time factor vectors reflect time dependency. Moreover, TATD imposes a time-dependent sparsity penalty to strengthen the smoothing regularization. The sparsity penalty modulates the amount of the regularization using the sparsity of time slices. TATD further improves accuracy using an effective alternating optimization scheme that incorporates an analytical solution and Adam optimizer. Through extensive experiments, we show that TATD effectively considers the time dependency for tensor decomposition, and achieves higher accuracy compared to existing methods. Our main contributions are as follows:

- *Method* We propose TATD, an accurate tensor decomposition method considering time dependency. TATD exploits a smoothing regularization for effectively modeling time factor with time dependency, and adjusts it by utilizing a time-varying sparsity.
- *Optimization* We propose an alternating optimization strategy suitable for our smoothing regularization. The strategy alternatively optimizes factor matrices with an analytical solution and Adam optimizer.
- *Performance* Extensive experiments show that exploiting time dependency and sparsity is crucial for accurate tensor decomposition of temporal tensors. TATD provides the state-of-the-art error (in terms of RMSE and MAE) for decomposing temporal tensors.

The rest of the paper is organized as follows. In Sect. 2, we explain preliminaries on tensor decomposition. Section 3 describes our proposed method TATD. We demonstrate our

experimental results in Sect. 4. After reviewing related work in Sect. 5, we conclude in Sect. 6. The source code of TATD and datasets are available at https://github.com/snudatalab/TATD/.

2 Preliminaries

We describe the preliminaries of tensor and tensor decomposition. We use the symbols listed in Table 1.

2.1 Tensor and notations

Tensors are defined as multi-dimensional arrays that generalize the one-dimensional arrays (or vectors) and two-dimensional arrays (or matrices) to higher dimensions. Specifically, the dimension of a tensor is referred to as order or way; the length of each mode is called 'dimensionality' and denoted by $I_1, \ldots, I_n$. We use boldface Euler script letters (e.g., $\boldsymbol{\mathcal{X}}$) to denote tensors, boldface capitals (e.g., $\mathbf{A}$) to denote matrices, and boldface lower cases (e.g., $\mathbf{a}$) to denote vectors. The $\alpha = (i_1, \ldots, i_N)$th entry of tensor $\boldsymbol{\mathcal{X}}$ is denoted by x_α.

A slice of a 3-order tensor is a two-dimensional subset of it. There are the horizontal, lateral, and frontal slices in a 3-order tensor $\boldsymbol{\mathcal{X}}$, denoted by $\mathbf{X}_{i_1::}$, $\mathbf{X}_{:i_2:}$, and $\mathbf{X}_{::i_3}$. A tensor containing a mode representing time is called a *temporal tensor*.

A time slice in a 3-mode temporal tensor represents a two-dimensional subset disjointed by each time index. For example, $\mathbf{X}_{i_t::}$ is an i_tth time slice when the first mode is the time mode. For brevity, we express $\mathbf{X}_{i_t::}$ as $\mathbf{X}_{i_t}$. Our proposed method is not limited to a 3-mode tensor so that a time slice in an N-order temporal tensor corresponds to an (N-1)-dimensional subset of the tensor sliced by each time index. We formally define a time slice $\boldsymbol{\mathcal{X}}_{i_t}$ as follows:

Table 1 Table of symbols

Symbol	Definition
$\boldsymbol{\mathcal{X}}$	Input tensor $\in \mathbf{R}^{I_1 \times \cdots \times I_N}$
α	Index $(i_1, \ldots, i_N)$ of $\boldsymbol{\mathcal{X}}$
x_α	Entry of $\boldsymbol{\mathcal{X}}$ with index α
N	Order of tensor $\boldsymbol{\mathcal{X}}$
I_n	Length of the nth mode of tensor $\boldsymbol{\mathcal{X}}$
$\mathbf{A}^{(n)}$	nth factor matrix ($\in \mathbf{R}^{I_n \times K}$)
$\mathbf{a}^{(n)}_{i_n}$	i_nth row of $\mathbf{A}^{(n)}$
$a^{(n)}_{i_n k}$	(i_n, k)th entry of $\mathbf{A}^{(n)}$
K	Rank of tensor $\boldsymbol{\mathcal{X}}$
t	Time mode of $\boldsymbol{\mathcal{X}}$
$\boldsymbol{\mathcal{X}}_{i_t}$	i_tth time slice of size $I_1 \times \cdots \times I_{t-1} \times I_{t+1} \cdots \times I_N$
ω_{i_t}	Number of observed entries of time slice $\boldsymbol{\mathcal{X}}_{i_t}$
$\|\boldsymbol{\mathcal{X}}\|_F$	Frobenius norm of tensor $\boldsymbol{\mathcal{X}}$
λ_t, λ_r	Regularization parameter

Definition 1 (*Time slice* $\mathcal{X}_{i_t}$) Given an N-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and a time mode t, we extract time slices of size $I_1 \times \cdots \times I_{t-1} \times I_{t+1} \cdots \times I_N$ by slicing the tensor $\mathcal{X}$ so that an i_tth time slice $\mathcal{X}_{i_t} \in \mathbb{R}^{I_1 \times \cdots \times I_{t-1} \times I_{t+1} \cdots \times I_N}$ is an $N-1$ order tensor obtained at time i_t where $1 \le i_t \le I_t$. □

The *Frobenius norm* of a tensor $\mathcal{X}$ ($\in \mathbf{R}^{I_1 \times \cdots \times I_N}$) is given by $||\mathcal{X}||_F = \sqrt{\sum_{\alpha \in \Omega} x_\alpha^2}$, where Ω is the set of indices of entries in $\mathcal{X}$, $\alpha = (i_1, \ldots, i_N)$ is an index included in Ω, and x_α is the $(i_1, \ldots, i_N)$th entry of the tensor $\mathcal{X}$.

2.2 Tensor decomposition

We provide the definition of CP decomposition (Harshman et al., 1970; Kiers 2000) which is one of the most representative decomposition models. Figure 1 illustrates CP decomposition of a 3-way sparse tensor. Our model TATD is based on CP decomposition.

Definition 2 (*CP decomposition*) Given a rank K and an N-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ with observed entries, CP decomposition approximates $\mathcal{X}$ by finding latent factor matrices $\{\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times K} \mid 1 \le n \le N\}$. The factor matrices are obtained by minimizing the following loss function:

$$\mathcal{L}\left(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}\right) = \sum_{\forall \alpha \in \Omega} \left(x_\alpha - \sum_{k=1}^{K} \prod_{n=1}^{N} a_{i_n k}^{(n)} \right)^2 \tag{1}$$

where Ω indicates the set of the indices of the observed entries, x_α indicates the $\alpha = (i_1, \ldots, i_N)$th entry of $\mathcal{X}$, and $a_{i_n k}^{(n)}$ indicates (i_n, k)th entry of $\mathbf{A}^{(n)}$. □

The standard CP decomposition method is not specifically designed to deal with time dependency; thus CP decomposition does not discover accurate temporal patterns in a time-evolving tensor. Although a previous work (Yu et al., 2016) has tried to capture temporal interaction, it does not (1) capture time dependency between adjacent time steps, and (2) exploit the sparsity of time slices. Our proposed TATD carefully captures temporal information and considers sparsity of time slices for better accuracy in decomposing temporal tensors.

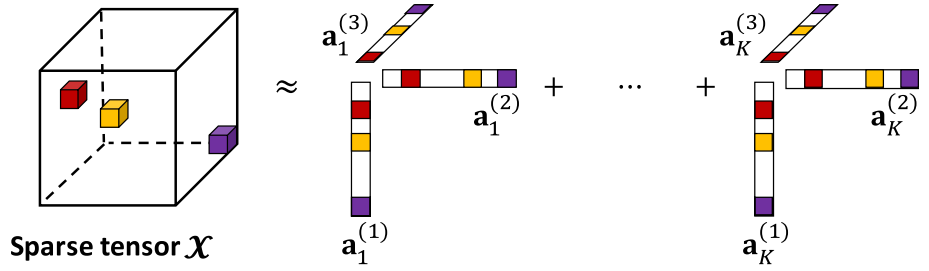

Fig. 1 CP decomposition of a 3-way sparse tensor into K components

3 Proposed method

In this section, we propose TATD (Time-Aware Tensor Decomposition), an accurate tensor decomposition method for sparse temporal tensors. We first introduce the overview of TATD in Sect. 3.1. We then explain the details of TATD in Sects. 3.2 and 3.3, and the optimization technique in Sect. 3.4.

3.1 Overview

TATD is a tensor decomposition method designed for sparse temporal tensors. There are several challenges in designing an accurate tensor decomposition method for temporal tensors.

1. **Model time dependency.** Time dependency is an essential structure of temporal tensor. How can we design a tensor decomposition model to reflect the time dependency?
2. **Exploit sparsity of time slices.** Time-evolving tensor has varying sparsity for its time slices. How can we exploit the temporal sparsity for better accuracy?
3. **Optimization.** How can we efficiently train our model and minimize its loss function?

To overcome the aforementioned challenges, we propose the following main ideas.

1. *Smoothing regularization* (Sect. 3.2). We propose a smoothing regularization on time factor to capture time dependency.
2. *Time-dependent sparsity penalty* (Sect. 3.3). We propose a time-dependent sparsity penalty to further improve the accuracy.
3. Careful optimization (Sect. 3.4). We propose an optimization strategy utilizing an analytical solution and Adam optimizer to efficiently and accurately train our model.

Figure 2 illustrates overview of TATD. We observe that adjacent time slices in a temporal tensor are closely related with each other due to a temporal trend of the tensor. Based on the observation, TATD uses smoothing regularization such that time factor vectors for adjacent time slices become similar. We also observe that different time slices have different densities. Instead of applying the same amount of regularization for all the time slices, we control the amount of regularization based on the sparsity of time slices such that sparse slices are affected more from the regularization. It is also crucial to efficiently optimize our objective function. We propose an optimization strategy exploiting alternating minimization to expedite training and improve the accuracy.

3.2 Smoothing regularization

We describe how we formulate the smoothing regularization on tensor decomposition to capture time dependency. Our main observation is that temporal tensors have temporal trends, and adjacent time slices are closely related. For example, consider an air quality tensor containing measurements of pollutant at a specific time and location; it is modeled as a 3-mode tensor $\mathcal{X}$ (time, location, type of pollutants; measurements). Since the amount of pollutants at nearby time steps are closely related, the time slice $\mathcal{X}_t$ at time t is closely

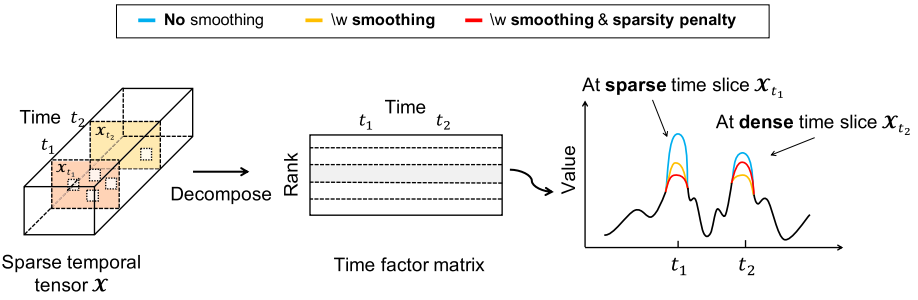

Fig. 2 Illustration of smoothing regularization and sparsity penalty by TATD. TATD accurately trains the time factor matrix with the smoothing regularization considering the time-varying sparsity. For a sparse time slice at t_1, TATD fits the time factor via strong regularization to actively consider nearby slices. For a dense time slice at t_2, TATD makes the time factor with weak regularization by paying little attention to nearby slices

related to the time slices $\mathcal{X}_{t-1}$ at time $t-1$ and $\mathcal{X}_{t+1}$ at time $t+1$. This implies the time factor matrix after tensor decomposition should have related rows for adjacent time steps.

Based on the observation, our objective function is as follows. Given an N-order temporal tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ with observed entries Ω, the time mode t, and a window size S, we find factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times K}$, $1 \le n \le N$ that minimizes

$$\mathcal{L} = \sum_{\alpha \in \Omega} \left(x_\alpha - \sum_{k=1}^{K} \prod_{n=1}^{N} a_{i_n k}^{(n)} \right)^2 + \lambda_t \sum_{i_t=1}^{I_t} \|\mathbf{a}_{i_t}^{(t)} - \tilde{\mathbf{a}}_{i_t}^{(t)}\|_2^2 + \lambda_r \sum_{n \neq t}^{N} \|\mathbf{A}^{(n)}\|_{\mathrm{F}}^2 \tag{2}$$

where we define

$$\tilde{\mathbf{a}}_{i_t}^{(t)} = \sum_{i_s \in \mathcal{M}(i_t, S)} w(i_t, i_s) \mathbf{a}_{i_s}^{(t)}, \tag{3}$$

and $\mathcal{M}(i_t, S)$ indicates adjacent indices i_s of i_t in a window of size S. λ_t and λ_r are regularization constants to adjust the effect of time smoothing and weight decay, respectively. $\tilde{\mathbf{a}}_{i_t}^{(t)}$ in Eq. (3) denotes the smoothed row of the time factor. The $\sum_{i_t=1}^{I_t} \|\mathbf{a}_{i_t}^{(t)} - \tilde{\mathbf{a}}_{i_t}^{(t)}\|_2^2$ term in Eq. (2) means that we regularize the i_tth row of the time factor to the smoothed vector from the neighboring rows in the factor. The weight $w(i_t, i_s)$ denotes the weight to give to the i_sth row of the time factor matrix for the smoothing the i_tth row of the time factor.

An important question is, how to determine the weight $w(i_t, i_s)$? We use the Gaussian kernel for the weight function due to the following two reasons. First, it does not require any parameters to tune, and thus we can focus more on learning the factors in tensor decomposition. Second, it fits our intuition that a row closer to the i_tth row should be given a higher weight. In Sect. 4, we show that TATD with Gaussian kernel outperforms all the competitors; however we note that other weight function can possibly replace the Gaussian kernel to further improve the accuracy, and we leave it as a future work.

Given a target row index i_t, an adjacent row index i_s, and a window size S, the weight function based on the Gaussian kernel is as follows:

$$w(i_t, i_s) = \frac{\mathcal{K}(i_t, i_s)}{\sum_{i_{s'} \in \mathcal{M}(i_t, S)} \mathcal{K}(i_t, i_{s'})} \tag{4}$$

where $\mathcal{K}$ is defined by

$$\mathcal{K}(i_t, i_s) = \exp\left(-\frac{(i_t - i_s)^2}{2\sigma^2}\right)$$

Note that σ affects the degree of smoothing; a higher value of σ imposes more smoothing. For each i_tth time slice, the model constructs a smoothed time factor vector $\tilde{\mathbf{a}}_{i_t}$ based on nearby factor vectors $\mathbf{a}_{i_s}$ and the weights $w(i_t, i_s)$.

Our model then aims to reduce the smoothing loss between the time factor vector $\mathbf{a}^{(t)}_{i_t}$ and the smoothed one $\tilde{\mathbf{a}}^{(t)}_{i_t}$.

3.3 Sparsity penalty

We describe how to further improve the accuracy of our method by considering the sparsity of time slices. The loss function (2) uses the same smoothing regularization penalty λ_t for all the time factor vectors. However, different time slices have different sparsity due to the different number of nonzeros in time slices (see Fig. 3), and it is thus desired to design our method so that it controls the degree of regularization penalty depending on the sparsity. For example, consider the 3-mode air quality tensor $\boldsymbol{\mathcal{X}}$ (time, location, type of pollutants; measurements), introduced in Sect. 3.2, containing measurements of pollutant at a specific time and location. Assume that the time slice $\boldsymbol{\mathcal{X}}_{t_1}$ at time t_1 is very sparse containing few nonzeros, while the time slice $\boldsymbol{\mathcal{X}}_{t_2}$ at time t_2 is dense with many nonzeros. The factor row $\mathbf{a}^{(1)}_{i_{t_2}}$ at time t_2 can be updated easily using its many nonzeros. However, the factor row $\mathbf{a}^{(1)}_{i_{t_1}}$ at time t_1 does not have enough nonzeros at its corresponding time slice, and thus it is hard to train $\mathbf{a}^{(1)}_{i_{t_1}}$ using only its few nonzeros; we need to actively use nearby slices to

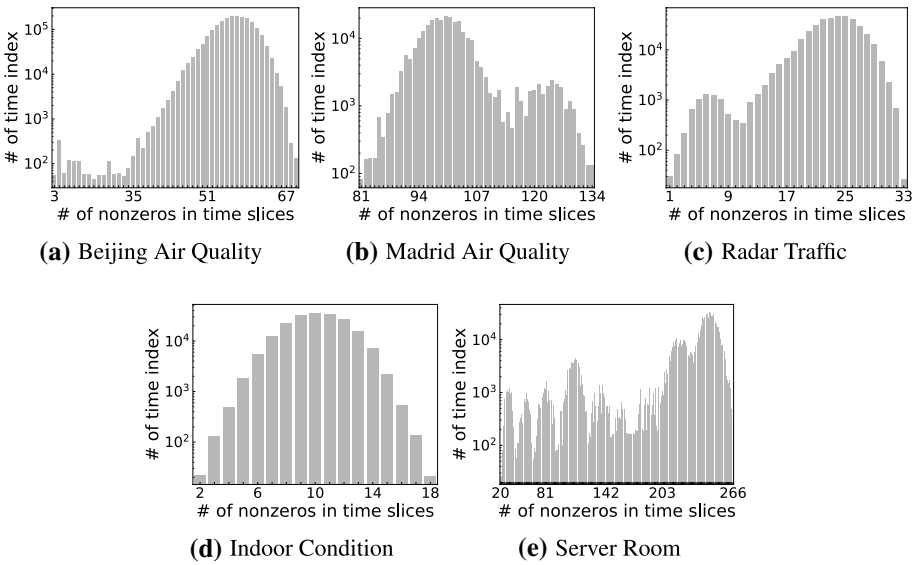

Fig. 3 Time-varying density of five real-world datasets. The horizontal axis represents the unique number of nonzero entries in time slices. The vertical axis represents the number of time indices with such number of nonzeros. Note that time slices have varying densities

make up for the lack of data. Thus, it is desired to impose more smoothing regularization at time t_1 than at time t_2.

Based on the motivation, TATD controls the degree of smoothing regularization based on the sparsity of time slices. Let the *time sparsity* β_{i_t} of the i_tth time slice be defined as

$$\beta_{i_t} = 1 - d_{i_t} \tag{5}$$

where a *time density* d_{i_t} is defined as follows:

$$d_{i_t} = (0.999 - 0.001)\frac{\omega_{i_t} - \omega_{min}}{\omega_{max} - \omega_{min}} + 0.001 \tag{6}$$

ω_{i_t} indicates the number of nonzeros at i_tth time slice; ω_{max} and ω_{min} are the maximum and the minimum values of the number of nonzeros in time slices, respectively. The *time density* d_{i_t} can be thought of as a min-max normalized version of ω_{i_t}, with its range regularized to [0.001, 0.999].

Using the defined time sparsity, we modify our objective function as follows.

$$\mathcal{L} = \sum_{\alpha \in \Omega} \left(x_\alpha - \sum_{k=1}^{K} \prod_{n=1}^{N} a_{i_n k}^{(n)} \right)^2 + \sum_{i_t=1}^{I_t} \lambda_t \beta_{i_t} ||\mathbf{a}_{i_t}^{(t)} - \tilde{\mathbf{a}}_{i_t}^{(t)}||_2^2 + \lambda_r \sum_{n \neq t}^{N} \|\mathbf{A}^{(n)}\|_{\mathrm{F}}^2 \tag{7}$$

Note that the second term is changed to include the time sparsity β_{i_t}; this makes the degree of the regularization vary depending on the sparsity of time slices.

Given the modified objective function in Eq. (7), we focus on minimizing the difference between $\mathbf{a}_{i_t}^{(t)}$ and $\tilde{\mathbf{a}}_{i_t}^{(t)}$ for time slices with a high sparsity rather than those with a low sparsity. TATD actively exploits the neighboring time slices when a target time slice is sparse, while it less exploits the neighboring ones for a dense time slice.

3.4 Optimization

To minimize the objective function in Eq. (7), TATD uses an alternating optimization method; it updates one factor matrix at a time while fixing all other factor matrices. TATD updates non-time factor matrices using the row-wise update rule (Shin et al., 2016) while updating the time factor matrix using the Adam optimizer (Kingma & Ba, 2014).

Updating non-time factor matrix. We note that updating a non-time factor matrix while fixing all other factor matrices is solved via the least square method, and we use the row-wise update rule (Shin et al., 2016) in ALS for it. The row-wise update rule is advantageous since it gives the optimal closed-form solution, and allows parallel update of factors. The update rule for i_nth row of the nth factor matrix $\mathbf{A}^{(n)} (n \neq t)$ is given as follows:

$$\mathbf{a}_{i_n}^{(n)} \leftarrow \arg\min_{\mathbf{a}_{i_n}^{(n)}} L(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}) = [\mathbf{B}_{i_n}^{(n)} + \lambda_r \mathbf{I}_K]^{-1} \times \mathbf{c}_{i_n}^{(n)} \tag{8}$$

where $\mathbf{B}_{i_n}^{(n)}$ is a $K \times K$ matrix whose entries are

$$(\mathbf{B}_{i_n}^{(n)})_{k_1 k_2} = \sum_{\forall \alpha \in \Omega_{i_n}^{(n)}} \prod_{l \neq n} a_{i_l k_1}^{(l)} \prod_{l \neq n} a_{i_l k_2}^{(l)}, \forall k_1, k_2, \tag{9}$$

$\mathbf{c}_{i_n}^{(n)}$ is a length K vector whose entries are

$$\sum_{\forall \alpha \in \Omega_{i_n}^{(n)}} x_\alpha \prod_{l \neq n} a_{i_l k}^{(l)}, \forall k \tag{10}$$

and $\Omega_{i_n}^{(n)}$ denotes the subset of Ω whose nth mode's index is i_n.

Updating time factor matrix. Updating the time factor matrix while fixing all other factor matrices is not the least square problem any more, and thus we turn to gradient based methods. We use the Adam optimizer which has shown superior performance for recent machine learning tasks. We verify that using the Adam optimizer only for the time factor leads to faster convergence compared to other optimization methods in Sect. 4.

Overall training. Algorithm 1 describes how we train TATD. We first initialize all factor matrices (line 1). For each iteration, we update a factor matrix while keeping all others fixed (lines 4–11). The time factor matrix is updated with Adam optimizer (line 7) until the validation RMSE increases, which is our convergence criterion (line 8) for Adam. Each of the non-time factor matrices is updated with the row-wise update rule (line 11) in ALS. We repeat this process until the validation RMSE continuously increases for five iterations, which is our global convergence criterion (line 12).

Algorithm 1: Training TATD

Input : Tensor $\mathcal{X} \in \mathbf{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with observed entries $\forall \alpha = (i_1, \ldots, i_n) \in \Omega$, rank K, window size S, sparsity penalty $\lambda_t \beta_{i_t}$, regularization parameter λ_r, and learning rate η

Output: Updated factor matrices $\mathbf{A}^{(n)} \in R^{I_n \times k}$ $(n = 1 \ldots N)$

```
1  initialize all factor matrices A^(n) for n = 1...N
2  repeat
3      for n = 1...N do
4          loss L ← Eq. (7)
5          if n is a time mode then
6              repeat
7                  update a factor A^(n) using Adam optimizer with a learning rate η
8              until convergence criterion is met;
9          else
10             for i_n = 1...I_n do
11                 update a row factor a_{i_n}^(n) using the row-wise update rule
12 until convergence criterion is met;
```

4 Experiment

We perform experiments to answer the following questions.

- Q1 *Accuracy* (Sect. 4.2). How accurately does TATD factorize real-world temporal tensors compared to other methods?
- Q2 *Effect of smoothing regularization* (Sect. 4.3). How accurately does TATD generate the time factor matrix and non-time factor matrices?
- Q3 *Effect of data sparsity* (Sect. 4.4). How does the sparsity of input tensors affect the decomposition performance of TATD and other methods?

Q4 *Running time* (Sect. 4.5). How fast is TATD compared to competitors?
Q5 *Effect of optimization* (Sect. 4.6). How effective is our proposed optimization approach for training TATD?
Q6 *Hyper-parameter study* (Sect. 4.7). How do the different hyper-parameter settings affect the performance of TATD?

4.1 Experimental settings

4.1.1 Machine

All experiments are performed on a machine equipped with Intel Xeon E5-2630 CPU.

4.1.2 Datasets

We evaluate TATD on five real-world datasets summarized in Table 2.

- *Beijing Air Quality* (Zhang et al., 2017) is a 3-mode tensor (hour, locations, atmospheric pollutants) containing measurements of pollutants. It was collected from 12 air-quality monitoring sites in Beijing between 2013 to 2017.
- *Madrid Air Quality* is a 3-mode tensor (day, locations, atmospheric pollutants) containing measurements of pollutants in Madrid between 2011 to 2018.
- *Radar Traffic* is a 3-mode tensor (hour, locations, directions) containing traffic volumes measured by radar sensors from 2017 to 2019 in Austin, Texas.
- *Indoor Condition* (Candanedo et al., 2017) is a 3-mode tensor (10 min, locations, ambient conditions) containing measurements. There are two ambient conditions defined: humidity and temperature. We construct a fully dense tensor from the original dataset and randomly sample 70% of the elements to make a tensor with missing entries. In Sect. 4.4, we sample from the fully dense version of it.
- *Server Room* is a 4-mode tensor (second, air conditioning, server power, locations) containing temperatures recorded in a server room. The first mode "air conditioning" means air conditioning temperature setups (24, 27, and 30 Celsius degrees); the second mode "server power" indicates server power usage scenarios (50%, 75%, and 100%).

Table 2 Summary of real-world tensors used for experiments

Name	Dimensionality	Nonzero	Granularity	Density
Beijing air quality[1]	**35,064** × 12 × 6	2,454,305	1 h	0.97
Madrid air quality[2]	**2678** × 24 × 14	337,759	1 day	0.37
Radar traffic[3]	**17,937** × 23 × 5	495,685	1 h	0.24
Indoor condition[4]	**19,735** × 9 × 2	241,201	10 min	0.70
Server room[5]	**4157** × 3 × 3 × 34	1,009,426	1 s	0.79

Bold text denotes time mode

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

[2] https://www.kaggle.com/decide-soluciones/air-quality-madrid

[3] https://data.austintexas.gov/Transportation-and-Mobility/Radar-Traffic-Counts/

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

[5] https://zenodo.org/record/3610078#.XlNpAigzaM8

Before applying tensor decomposition, we z-normalize the datasets. Each dataset is randomly split into training, validation, and test sets with the ratio of 8:1:1; the validation set is used for determining an early stopping point.

4.1.3 Competitors

We compare TATD with the following competitors which use only the observed entries of a given tensor.

- *CP-ALS* (Harshman et al., 1970): a standard CP decomposition method using ALS.
- *TRTF* (Yu et al., 2016): a temporally regularized tensor decomposition method (https://github.com/xinychen/transdim). TRTF is an extension of TRMF to deal with tensors.
- *CoSTCo* (Liu et al., 2019): a CNN-based tensor decomposition method (https://github.com/USC-Melady/KDD19-CoSTCo).

4.1.4 Metrics

We evaluate the performance using RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error) defined as follows.

$$\text{RMSE} = \sqrt{\frac{1}{|\Omega|}\sum_{\forall \alpha \in \Omega}\left(x_\alpha - \hat{x}_\alpha\right)^2} \qquad \text{MAE} = \frac{1}{|\Omega|}\sum_{\forall \alpha \in \Omega}|x_\alpha - \hat{x}_\alpha|$$

Ω indicates the set of the indices of observed entries. x_α stands for the entry with index α and $\hat{x}_\alpha$ is the corresponding reconstructed value. In addition, we evaluate the quality of generated factor matrices from decomposition methods using Factor Match Score (FMS) and Time Factor Match Score (TFMS) defined as follows.

$$\text{FMS} = \min_k \left(\left(1 - \frac{|\xi_k - \hat{\xi}_k|}{\max(\xi_k, \hat{\xi}_k)} \right) \prod_{n=1, n\neq t}^{N} |\mathbf{a}_{:k}^{(n)\intercal} \hat{\mathbf{a}}_{:k}^{(n)}| |\mathbf{a}_{:k}^{(t)\intercal} \hat{\mathbf{a}}_{:k}^{(t)}| \right)$$

$$\text{TFMS} = \min_k \left(\left(1 - \frac{|\xi_k - \hat{\xi}_k|}{max(\xi_k, \hat{\xi}_k)} \right) |\mathbf{a}_{:k}^{(t)\intercal} \hat{\mathbf{a}}_{:k}^{(t)}| \right)$$

where $\mathbf{a}_{:k}^{(n)}$ and $\mathbf{a}_{:k}^{(t)}$ denote the kth columns normalized to the unit norm of the original non-time factor matrix $\mathbf{A}^{(\mathbf{n})}(n \neq t)$ and time factor matrix $\mathbf{A}^{(t)}$, respectively, and ξ_k denotes the weight for each kth column factor for $k = 1, \ldots, K$. Similarly, $\hat{\mathbf{a}}_{:k}^{(n)}$ and $\hat{\mathbf{a}}_{:k}^{(t)}$ are the normalized kth columns of the extracted non-time factor matrix $\hat{\mathbf{A}}^{(n)}(n \neq t)$ and the time factor matrix $\hat{\mathbf{A}}^{(t)}$, respectively, and $\hat{\xi}_k$ denotes the corresponding weight for each kth column factor. Note that FMS and TFMS are closer to 1 if the extracted factors and the original ones are more similar (see Acar et al., 2011 for the details of the metric).

4.1.5 Hyper-parameter

We use hyper-parameters in Table 3 for TATD, except in Sect. 4.7 where we vary hyper-parameters. We use 0.5 for σ which adjusts the smoothing level in kernel function. We change the window size $S \in \{1, 3, 5, 7, 9\}$ and find the optimal value for each dataset. For

competitors, we follow their default hyperparameter settings suggested in their papers and tune them using grid search. Specifically, we find regularization parameters from $\{0.1, 1, 10, 100\}$ for TRTF. For CoSTCo, we find the learning rate from $\{0.001, 0.01\}$ and the batch size from $\{126, 256, 512\}$. We also change the activation function used in CoSTCo from ReLU to ELU. For all methods, we find optimal hyperparameter settings with evaluation results of validation datasets.

4.2 Accuracy (Q1)

We compare TATD with competitors in terms of RMSE and MAE in Table 4. TATD-0 indicates TATD without the sparsity penalty. TATD consistently gives the best accuracy for all the datasets; TATD-0 provides the second-best performance. TATD achieves up to 1.08× lower RMSE and 1.37× lower MAE compared to the second-best methods. Note that Indoor Condition dataset represents strong smoothness and the least amount of noise on its values. The gap in RMSE of TATD and CP-ALS on Indoor Condition dataset demonstrates that the smoothing regularization effectively decomposes the real-world tensors by capturing temporal patterns and adjusting the noise.

4.3 Effect of smoothing regularization (Q2)

We evaluate the effect of smoothing regularization of TATD, by comparing factor matrices extracted by TATD and CP-ALS with true factor matrices. For evaluation, we construct a synthetic tensor from true factor matrices, decompose it, and then compare factor matrices extracted from TATD and CP-ALS with the true ones. We create a 3-order synthetic temporal tensor of size $900 \times 30 \times 30$ with factor matrices of rank 5. To generate the tensor, we first make a time factor matrix $\mathbf{A} \in \mathbb{R}^{900\times5}$ by randomly sampling five time-series from Indoor Condition dataset, and non-time factor matrices $\mathbf{B}, \mathbf{C} \in \mathbb{R}^{30\times5}$ having random values from [0, 1). We then create a synthetic temporal tensor from those factor matrices with Eq. (1), add a noise tensor having random values from [0, 0.001) to it, and randomly sample 90% of the tensor to make it sparse.

Table 5 shows that TATD achieves higher FMS and TFMS than CP-ALS in decomposing the synthetic tensor. With the smoothing regularization, TATD precisely generates the time factor matrix, achieving 1.11× higher TFMS in the synthetic tensor. Furthermore, it leads to generating accurate non-time factor matrices, providing 1.59× higher FMS. Our smoothing regularization allows TATD to generate the time factor matrix and non-time factor matrices accurately.

Table 3 Default hyper-parameter setting

Dataset	Learning rate η	Rank K	Window S	Penalty λ_t
Beijing air quality	10^{-2}	10	1	10^3
Madrid air quality	10^{-2}	10	2	10^2
Radar traffic	10^{-2}	10	3	10^2
Indoor condition	10^{-2}	10	1	10^2
Server room	10^{-3}	10	1	10^{-1}

Table 4 Performance of tensor decomposition by TATD and competitors

Data	Beijing air quality		Madrid air quality		Radar traffic		Indoor condition		Server room	
Method	Test RMSE	Test MAE	Test RMSE	Test MAE	Test RMSE	Test MAE	Test RMSE	Test MAE	Test RMSE	Test MAE
CP-ALS	0.352	0.219	0.456	0.293	0.365	0.248	0.624	0.316	0.076	0.048
CoSTCo	0.360	0.223	0.461	0.303	0.298	0.197	0.609	0.303	0.306	0.090
TRTF	0.349	0.219	0.418	<u>0.275</u>	0.275	0.168	0.090	0.062	0.183	0.143
TATD-0	<u>0.327</u>	<u>0.204</u>	<u>0.416</u>	0.279	<u>0.257</u>	<u>0.160</u>	<u>0.088</u>	<u>0.057</u>	<u>0.058</u>	<u>0.039</u>
TATD **(proposed)**	**0.323**	**0.201**	**0.409**	**0.274**	**0.249**	**0.152**	**0.086**	**0.055**	**0.054**	**0.035**

The best method is in bold, and the second-best method is underlined. Our proposed TATD consistently shows the best performance in all datasets

4.4 Effect of data sparsity (Q3)

We evaluate the performance of TATD with varying data sparsity. We sample the data with the ratio of $\{10, 30, 50, 70, 90\}\%$ to identify how accurately the method decomposes real-world tensors even when the data are highly sparse. Figure 4 shows the errors of TATD and competitors for five datasets. Note that TATD precisely decomposes the tensors even when they are highly sparse, compared to competitors. There are two reasons for the best performance of TATD as the sparsity increases. First, TATD is designed to learn the factor for a target slice by using its neighboring slices; this is especially useful when the target slice is extremely sparse and has no information to train its factor. Second, TATD explicitly considers sparsity in its model through the sparsity penalty, and imposes more regularization for sparser slices.

4.5 Running time (Q4)

We compare the running times of TATD and competitors. TATD uses the ALS+Adam optimizer (see Sect. 4.6), while TATD w/ Adam uses the Adam optimizer. For a stopping condition, we set the maximum iterations to 500 and use the early stopping technique.

Figure 5 shows the running times of all methods. Note that TATD w/Adam shows the fastest running time; CoSTCo and TATD follow after that. Although TATD is slower than TATD w/Adam and CoSTCo, TATD gives the smallest errors (see Table 4 and Figure 6), and thus provides more accurate results. Note that the running times of methods vary over different datasets, since the convergence condition for each method varies depending on datasets.

4.6 Effect of optimization (Q5)

We evaluate our proposed optimization strategy in terms of error and running time. We call our strategy ALS+Adam and compare it with the following optimization strategies for TATD.

- *SGD* a standard stochastic gradient descent method which is widely used for optimization.
- *Adam* a recent gradient-based method using momentum and controlled learning rate.
- *Alternating Adam* an alternating minimization method which updates a single factor matrix with Adam while fixing other factor matrices.
- *ALS + SGD* an alternating minimization method which updates a time factor matrix with SGD and non-time factor matrices with the least square solution.

Table 5 FMS and TFMS of TATD and CP-ALS for a synthetic tensor

	FMS	TFMS
CP-ALS	0.603	0.865
TATD **(proposed)**	**0.964**	**0.964**

The closer the FMS and TFMS are to 1, the better they are, and the best method is in bold. TATD decomposes the tensor more accurately than CP-ALS

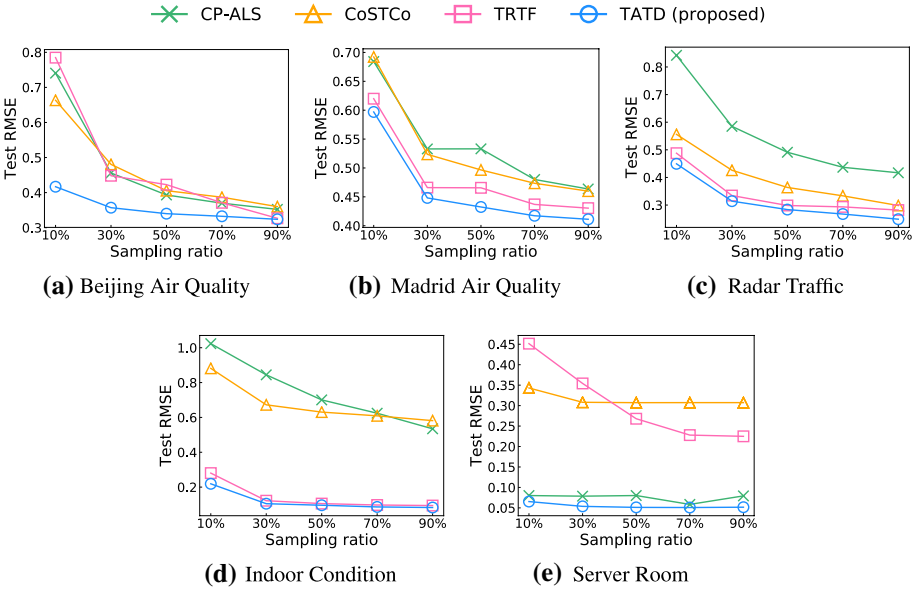

Fig. 4 Test RMSE of TATD and competitors for varying data sampling ratios. TATD shows the smallest errors when decomposing highly sparse tensors due to the careful consideration of sparsity

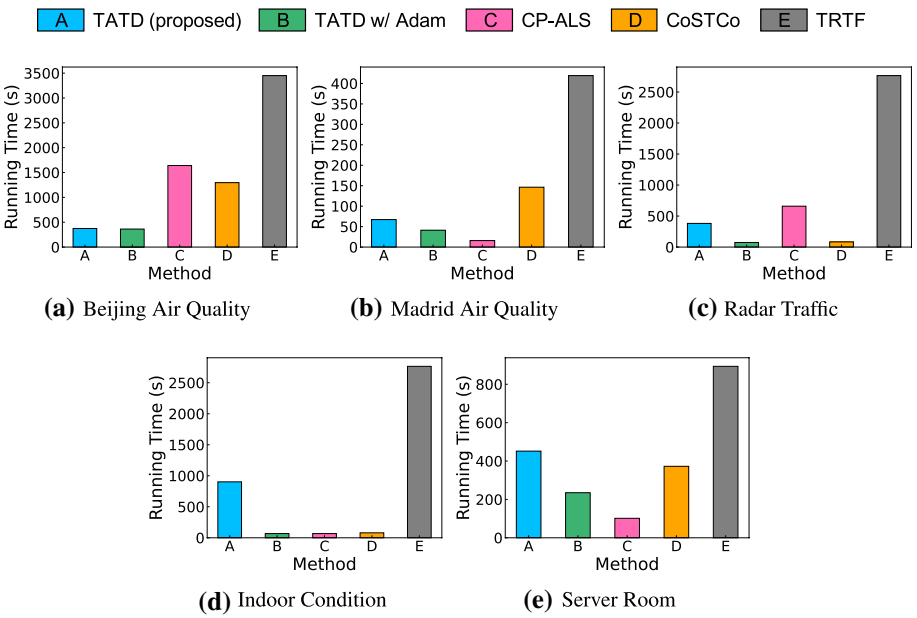

Fig. 5 Comparison of the running times between TATD and competitors

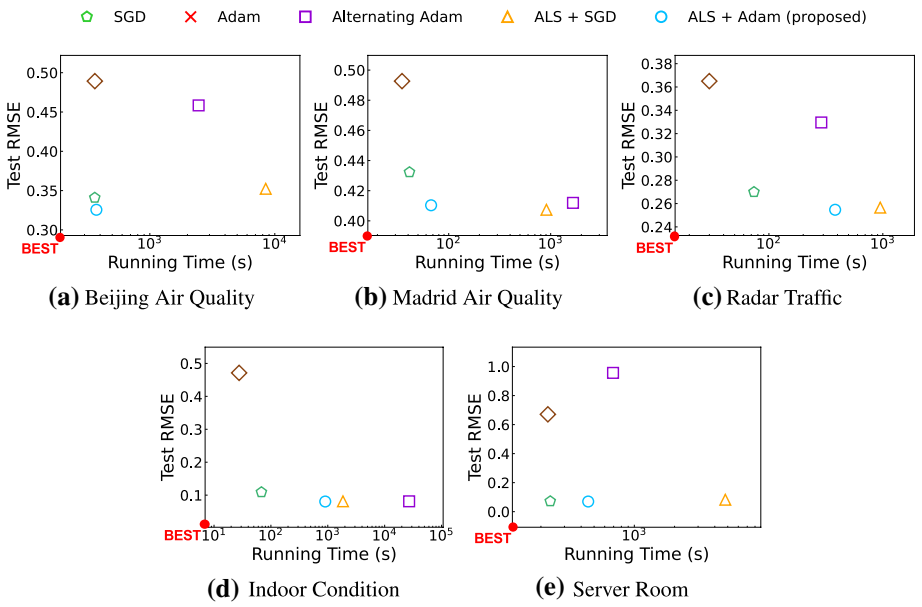

Fig. 6 Comparison of optimization strategies in TATD. Our proposed strategy ALS+Adam and Adam show the best trade-off between the error and the running time. Note also that ALS+Adam provides the smallest RMSE

We use the same stopping condition mentioned in Sect. 4.5. We use the learning rates in Table 3 for the optimization schemes using Adam and the learning rate from $\{10^{-3}, 10^{-4}, 10^{-5}\}$ for the schemes using SGD.

Figure 6 shows that our proposed strategy ALS+Adam and Adam show the best trade-off between the error and the running time. ALS+Adam produces the smallest error, but takes more time than Adam to train for better accuracy. On the other hand, Adam is faster, but less inaccurate than ALS+Adam. We select ALS+Adam as our optimization strategy since our main focus is to achieve a high accuracy.

4.7 Hyper-parameter study (Q5)

We evaluate the performance of TATD with regard to hyper-parameters: smoothing regularization penalty and rank size.

4.7.1 Smoothing regularization penalty

The smoothing regularization penalty λ_t has an important role in the proposed TATD 's performance; thus we vary the smoothing regularization penalty λ_t and evaluate the test RMSE in Fig. 7. Note that too small or too large values of λ_t do not give the best results; too small value of λ_t leads to overfitting, and too large value of it leads to underfitting. The results show that a right amount of smoothing regularization gives the smallest error, verifying the effectiveness of our proposed idea.

4.7.2 Rank

We increase the rank K from 5 to 50 and evaluate the test RMSE in Fig. 8. We have two main observations. First, TATD shows a stable performance improvement with increasing ranks, compared to CP-ALS which shows unstable performances. Second, the error gap between TATD and competitors increases with increasing ranks. Higher ranks may make the models overfit to a training dataset; however, TATD works even better for higher ranks since it exploits rich information from neighboring rows when regularizing a row of the time factor matrix. TRTF also works well for higher ranks on several datasets since it learns the parameters for each column factor. However, TATD achieves a similar or better effect without using excessive parameters thanks to the smoothing regularization.

5 Related work

We describe previous works that are closely related to our work. We present one of the major tensor decomposition methods, CP decomposition.

5.1 Tensor decomposition

In early stage, CP decomposition methods (Kang et al., 2012; Jeon et al., 2015; Choi et al., 2014) have been widely used for analyzing large-scale real-world tensors. Kang et al. (2012) and Jeon et al. (2015) propose distributed CP decomposition methods running on the MapReduce framework. Choi et al. (2014) propose a scalable CP decomposition

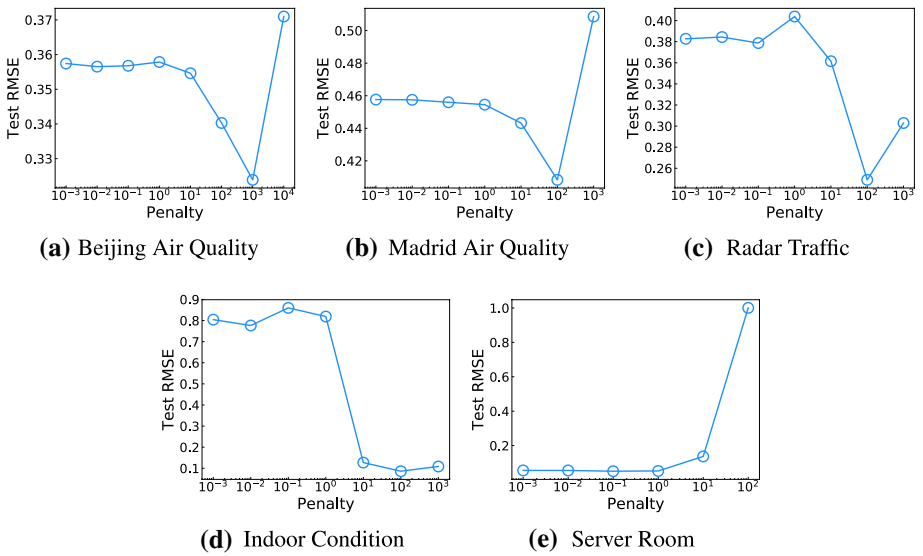

Fig. 7 Effect of the smoothing regularization penalty parameter λ_t in TATD. Note that too small or too large values of λ_t lead to overfitting and underfitting, respectively. A right amount of smoothing regularization gives the smallest error, verifying the effectiveness of our proposed idea

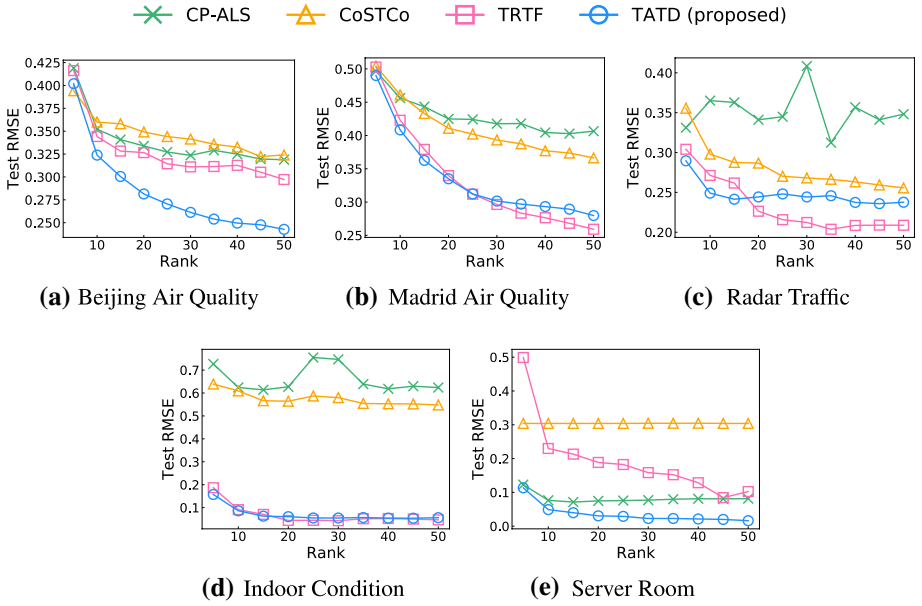

Fig. 8 Effect of rank on the performance of TATD. TATD works even better for higher ranks since it exploits rich information from neighboring rows when regularizing a row of the time factor matrix

method by exploiting properties of a tensor operation used in CP decomposition. Battaglino et al. (2018) propose a randomized CP decomposition method which reduces the overhead of computation and memory. However, they are not appropriate to deal with highly sparse tensors since they do assume non-observable entries are zero.

Several CP decomposition methods have been developed to handle sparse tensors without setting the values of the non-observable entries as zero. Papalexakis et al. (2012) propose ParCube to obtain sparse factor matrices using a sampling technique in parallel systems. Beutel et al. (2014) propose FlexiFaCT, which performs a coupled matrix-tensor decomposition using Stochastic Gradient Descent (SGD) update rules. Shin et al. (2016) propose CDTF and SALS, which are scalable CP decomposition methods for sparse tensors. Smith and Karypis (2017) improves the efficiency of CP decomposition for sparse tensors by exploiting a compressed data structure. The above CP decomposition methods do not consider time dependency and time-varying sparsity which are crucial for temporal tensors. On the other hand, TATD improves accuracy for temporal tensors by exploiting time dependency and time-varying sparsity.

Applications. CP decomposition have been used for various applications. Kolda et al. (2005) analyze a hyperlink graph modeled as 3-way tensor using CP decomposition. Tensor decomposition is also applied to tag recommendation (Rendle et al., 2009; Rendle & Schmidt-Thieme, 2010). Sun et al. (2009) develop a content-based network analysis framework for finding higher-order clusters. Lebedev et al. (2015) exploit CP decomposition to compress convolution filters of convolutional neural networks (CNNs). Several works (Lee et al., 2018; Perros et al., 2017, 2018) use tensor decomposition for analyzing Electronic Health Record (EHR) data.

5.2 Tensor decomposition on temporal tensors

Tensor decomposition methods have been used to deal with diverse real-world temporal tensors. Dunlavy et al. (2011) propose a tensor decomposition method with an exponential smoothing technique for temporal link prediction. Matsubara et al. (2012) propose a tensor decomposition method with a probabilistic inference to discover hidden topics of web-click logs and perform multi-level analysis for long-term forecasting with this method. Yu et al. (2016) propose a matrix/tensor decomposition method with an autoregressive temporal regularization to handle general time-series. de Araujo et al. (2017) present a non-negative coupled tensor decomposition for forecasting future links in evolving social networks.

In addition, there exist various tensor decomposition methods to analyze spatio-temporal tensors that include spatial information in addition to time information. Zhou et al. (2015) propose a tensor decomposition method with a spatio-temporal regularization to predict missing entries in real-world traffic data. Afshar et al. (2017) propose a non-negative tensor decomposition method to discover interpretable patterns in spatio-temporal tensors. Liu et al. (2019) propose a general tensor decomposition method by exploiting the expressive power of convolutional neural networks to model non-linear interactions inside spatio-temporal tensors.

Beyond the static tensor, many researchers have developed online tensor decomposition methods to deal with tensors collected in real-time. Kasai (2016) propose an online tensor decomposition method for sparse tensors corrupted by noises. Song et al. (2017) propose a dynamic tensor decomposition method for temporal multi-aspect streaming tensor. Zhou et al. (2018) propose a fast and memory-efficient online algorithm for sparse tensor decomposition.

However, those approaches are not designed for modeling time dependency from both past and future information, whereas TATD obtains a time factor considering neighboring factors for both past and future time steps, giving an accurate tensor decomposition result. Moreover, they do not exploit the temporal sparsity, a common characteristic of real-world temporal tensors, while TATD actively exploits the temporal sparsity.

6 Conclusion

We propose TATD (Time-Aware Tensor Decomposition), an accurate tensor decomposition method for sparse temporal tensors. To capture time dependency and sparsity in real world temporal tensors, we design a smoothing regularization on time factor, and adjust the amount of the regularization according to the sparsity of time slices. Moreover, we accurately optimize TATD with a carefully designed optimization strategy. Extensive experimental results show that TATD achieves higher accuracy in decomposing real-world tensors compared to competitors. Future works include extending TATD for an online or a distributed setting.

Author contributions Not applicable.

Funding This work was supported by the National Research Foundation of Korea (NRF) funded by MSIT(2019R1A2C2004990). The Institute of Engineering Research and ICT at Seoul National University provided research facilities for this work. U Kang is the corresponding author.

Data availability The datasets used during the current study are available at https://github.com/snudatalab/TATD/.Code availability The source code used during the current study is available at https://github.com/snudatalab/TATD/.

Declarations

Conflict of interest Not applicable.

References

Acar, E., Kolda, T. G., & Dunlavy, D. M. (2011). *All-at-once optimization for coupled matrix and tensor factorizations*. arXiv preprint arXiv:11053422

Afshar, A., Ho, J. C., Dilkina, B., Perros, I., Khalil, E. B., Xiong, L., & Sunderam, V. (2017). Cp-ortho: An orthogonal tensor factorization framework for spatio-temporal data. In *Proceedings of the 25th ACM SIGSPATIAL international conference on advances in geographic information systems*, pp. 1–4.

Battaglino, C., Ballard, G., & Kolda, T. G. (2018). A practical randomized CP tensor decomposition. *SIAM Journal on Matrix Analysis Applications, 39*(2), 876–901.

Beutel, A., Talukdar, P. P., Kumar, A., Faloutsos, C., Papalexakis, E. E., & Xing, E. P. (2014). Flexifact: Scalable flexible factorization of coupled tensors on hadoop. In *Proceedings of the 2014 SIAM international conference on data mining*, Philadelphia, Pennsylvania, USA, April 24–26, 2014 (pp. 109–117). SIAM.

Candanedo, L. M., Feldheim, V., & Deramaix, D. (2017). Data driven prediction models of energy use of appliances in a low-energy house. *Energy and Buildings, 140,* 81–97.

Choi, J. H., & Vishwanathan, S. (2014). Dfacto: Distributed factorization of tensors. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger(Eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014*, December 8–13, 2014, Montreal, Quebec, Canada.

de Araujo, M. R., Ribeiro, P. M. P., & Faloutsos, C. (2017). Tensorcast: Forecasting with context using coupled tensors (best paper award). In *2017 IEEE International Conference on Data Mining (ICDM)* (pp. 71–80). IEEE.

Dunlavy, D. M., Kolda, T. G., & Acar, E. (2011). Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD), 5*(2), 10.

Harshman, R. A., et al. (1970). Foundations of the parafac procedure: Models and conditions for an " explanatory" multimodal factor analysis.

Jeon, I., Papalexakis, E. E., Kang, U., & Faloutsos, C. (2015). Haten2: Billion-scale tensor decompositions. In *2015 IEEE 31st international conference on data engineering* (pp. 1047–1058). IEEE.

Kang, U., Papalexakis, E. E., Harpale, A., & Faloutsos, C. (2012). Gigatensor: Scaling tensor analysis up by 100 times—Algorithms and discoveries. In *KDD*, pp. 316–324.

Kasai, H. (2016). Online low-rank tensor subspace tracking from incomplete data by CP decomposition using recursive least squares. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 2519–2523). IEEE.

Kiers, H. A. (2000). Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics: A Journal of the Chemometrics Society, 14*(3), 105–122.

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv preprint arXiv:14126980

Kolda, T. G., Bader, B. W., & Kenny, J. P. (2005). Higher-order web link analysis using multilinear algebra. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005)*, 27–30 November 2005, Houston, Texas, USA (pp. 242–249). IEEE Computer Society.

Kolda, T. G., & Sun, J. (2008). Scalable tensor decompositions for multi-aspect data mining. In *2008 eighth IEEE international conference on data mining* (pp. 363–372). IEEE.

Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I. V., & Lempitsky, V. S. (2015). Speeding-up convolutional neural networks using fine-tuned CP-decomposition. In Y. Bengio & Y. LeCun (Eds.), *3rd*

International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings.

Lee, J., Oh, S., & Sael, L. (2018). GIFT: Guided and interpretable factorization for tensors with an application to large-scale multi-platform cancer analysis. *Bioinformatics, 34*(24), 4151–4158.

Liu, H., Li, Y., Tsang, M., & Liu, Y. (2019). Costco: A neural tensor completion model for sparse tensors. *Training, 10*(4), 10–3.

Maruhashi, K., Guo, F., & Faloutsos, C. (2011). Multiaspectforensics: Pattern mining on large-scale heterogeneous networks with tensor analysis. In *2011 international conference on advances in social networks analysis and mining* (pp. 203–210). IEEE.

Matsubara, Y., Sakurai, Y., Faloutsos, C., Iwata, T., & Yoshikawa, M. (2012). Fast mining and forecasting of complex time-stamped events. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 271–279). ACM.

Oh, S., Park, N., Sael, L., & Kang, U. (2018). Scalable tucker factorization for sparse tensors—Algorithms and discoveries. In *34th IEEE International Conference on Data Engineering, ICDE 2018*, Paris, France, April 16–19, 2018.

Papalexakis, E. E., Faloutsos, C., & Sidiropoulos, N. D. (2012). Parcube: Sparse parallelizable tensor decompositions. In *ECML-PKDD, Springer, Lecture Notes in Computer Science*, Vol. 7523, pp. 521–536.

Park, N., Oh, S., & Kang, U. (2017). Fast and scalable distributed Boolean tensor factorization. In *33rd IEEE International Conference on Data Engineering, ICDE 2017*, San Diego, CA, USA, April 19–22, 2017, pp. 1071–1082.

Perros, I., Papalexakis, E. E., Park, H., Vuduc, R. W., Yan, X., deFilippi, C., Stewart, W. F., & Sun, J. (2018). Sustain: Scalable unsupervised scoring for tensors and its application to phenotyping. In: Y. Guo & F. Farooq (Eds.), *Proceedings of the 24th ACM SIGKDD international conference on Knowledge Discovery & Data Mining, KDD 2018*, London, UK, August 19–23, 2018 (pp. 2080–2089). ACM.

Perros, I., Papalexakis, E. E., Wang, F., Vuduc, R. W., Searles, E., Thompson, M., & Sun, J. (2017). Spartan: Scalable PARAFAC2 for large & sparse data. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, Halifax, NS, Canada, August 13–17, 2017 (pp. 375–384). ACM.

Rendle, S., Marinho, L. B., Nanopoulos, A., & Schmidt-Thieme, L. (2009). Learning optimal ranking with tensor factorization for tag recommendation. In *SIGKDD*, pp. 727–736.

Rendle, S., & Schmidt-Thieme, L. (2010). Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM*, pp. 81–90.

Shin, K., Sael, L., & Kang, U. (2016). Fully scalable methods for distributed tensor factorization. *IEEE Transactions on Knowledge and Data Engineering, 29*(1), 100–113.

Smith, S., & Karypis, G. (2017). Accelerating the tucker decomposition with compressed sparse tensors. In F. F. Rivera, T. F. Pena & J. C. Cabaleiro (Eds.), *Euro-Par 2017: Parallel Processing—23rd International Conference on Parallel and Distributed Computing*, Santiago de Compostela, Spain, August 28–September 1, 2017, *Proceedings, Springer, Lecture Notes in Computer Science*, Vol. 10417, pp. 653–668.

Song, Q., Huang, X., Ge, H., Caverlee, J., & Hu, X. (2017). Multi-aspect streaming tensor completion. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 435–443.

Sun, J., Papadimitriou, S., Lin, C., Cao, N., Liu, S., & Qian, W. (2009). Multivis: Content-based social network exploration through multi-way visual analysis. In *Proceedings of the SIAM international conference on data mining, SDM 2009*, April 30–May 2, 2009, Sparks, Nevada, USA (pp. 1064–1075). SIAM.

Sun, J., Papadimitriou, S., & Philip, S. Y. (2006). Window-based tensor analysis on high-dimensional and multi-aspect streams. In *Sixth International Conference on Data Mining (ICDM'06)* (pp. 1076–1080). IEEE.

Sun, Y., Gao, J., Hong, X., Mishra, B., & Yin, B. (2015). Heterogeneous tensor decomposition for clustering via manifold optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 38*(3), 476–489.

Symeonidis, P. (2016). Matrix and tensor decomposition in recommender systems. In: *Proceedings of the 10th ACM conference on recommender systems*, pp. 429–430.

Yu, H. F., Rao, N., & Dhillon, I. S. (2016). Temporal regularized matrix factorization for high-dimensional time series prediction. In *Advances in neural information processing systems*, pp. 847–855.

Zhang, S., Guo, B., Dong, A., He, J., Xu, Z., & Chen, S. X. (2017). Cautionary tales on air-quality improvement in Beijing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 473*(2205), 20170457.

Zhou, H., Zhang, D., Xie, K., & Chen, Y. (2015). Spatio-temporal tensor completion for imputing missing internet traffic data. In *2015 IEEE 34th international performance computing and communications conference (IPCCC)* (pp. 1–7). IEEE.

Zhou, S., Erfani, S., & Bailey, J. (2018). Online CP decomposition for sparse tensors. In *2018 IEEE International Conference on Data Mining (ICDM)* (pp. 1458–1463). IEEE.