A Contemporary and Comprehensive Survey on Streaming Tensor Decomposition

Le Trung Thanh, Karim Abed-Meraim, Fellow, IEEE, Nguyen Linh Trung, Senior Member, IEEE, and Adel Hafiane, Member, IEEE

Abstract—Tensor decomposition has been demonstrated to be successful in a wide range of applications, from neuroscience and wireless communications to social networks. In an online setting, factorizing tensors derived from multidimensional data streams is however nontrivial due to several inherent problems of real-time stream processing. In recent years, many research efforts have been dedicated to developing online techniques for decomposing such tensors, resulting in significant advances in streaming tensor decomposition or tensor tracking. This topic is emerging and enriches the literature on tensor decomposition, particularly from the data stream analytics perspective. Thus, it is imperative to carry out an overview of tensor tracking to help researchers and practitioners understand its developments and achievements, summarise the current trends and advances, and identify challenging problems. In this article, we provide a contemporary and comprehensive survey on different types of tensor tracking techniques. We particularly categorize the state-of-the-art methods into three main groups: streaming CP decompositions, streaming Tucker decompositions, and streaming decompositions under other tensor formats (i.e., tensor-train, t-SVD, and BTD). In each group, we further divide the existing algorithms into sub-categories based on their main optimization framework and model architectures. Finally, we present several applications, research challenges, open problems, and potential directions of tensor tracking in the future.

Index Terms—Tensor decomposition, CP, Tucker, tensor-train, t-SVD, BTD, data stream, online optimization, low-rank tensor approximation.

1 INTRODUCTION

Tensor decomposition (TD) has attracted much attention from the signal/image processing and machine learning communities [1]. As a tensor is a multiway array, it provides a natural representation for multidimensional data. Accordingly, TD which factorizes a tensor into a set of basis components (e.g., vectors, matrices, or simpler tensors) has become a popular tool for multivariate and high-dimensional data analysis. In particular, we have witnessed significant advances in TD and a rapid growth in its applications over the last two decades [2]. Several types of TD, such as CANDECOMP/PARAFAC (CP) [3], high-order SVD (HOSVD)/Tucker [4], tensor train/network [5], t-SVD [6], and block-term decomposition (BTD) [7], have been developed and successfully applied to various domains, from neuroscience [8], [9] and wireless communications [10], [11] to social networks [12], [13].

The demand for (near) real-time stream processing has been increasing over the years since many modern applications (e.g., Internet-of-Things) generate massive amounts of streaming data over time and analytical insights from such data can bring several benefits, e.g., for real-time decision making [14]. As its name implies, (near) real-time stream processing needs to immediately deliver and analyse data streams upon their arrival. Since streaming data grow bigger, faster, and become more complex by the time, there exist several inherent problems which are still challenging issues, such as (i) the unbounded size of streaming data, (ii) time-varying model, concept drift, or dataset shift, and (iii) uncertainty and imperfection, etc. We refer the readers to [14], [15] for good surveys on data stream analysis.

When using tensors to represent data streams, TD is generally referred to as dynamic tensor analysis, tensor tracking or adaptive/online/streaming tensor decomposition. Specifically, factorizing a streaming tensor is nontrivial due to several computational challenges. First, as tensor streams are continuously generated, their volume grows significantly over time and possibly to infinity. Applying the conventional batch TD methods to such tensors is not possible as they require data to be stored and processed offline. Second, properties of streaming tensors (e.g., the low-rank approximation model) can vary with time in unforeseen ways. Moreover, tensor streams often happen in real-time, so retransmission of a stream is difficult, even impossible. Accordingly, batch tensor estimation and decomposition become less accurate when time passes. Last but not least, some modern applications require high-speed data acquisition systems to rapidly acquire and process massive data streams. In such a case, very fast and (near) real-time processing is highly important. However, batch TDs are often of high complexity, and hence turn out to be inefficient. These characteristics make tensor tracking much different from batch tensor decomposition and lead to several distinguishing requirements for tensor trackers, such as low latency and memory storage, high scalability, adaptation to time variation, and robustness, to name a few.

As the literature of tensor tracking has dramatically expanded in recent years, it is imperative to conduct an extensive overview of the state-of-the-art tensor tracking algorithms to help researchers and practitioners to identify: (i) which topics in tensor tracking are significant and emerging, (ii) what kind of tracking models and related analysis techniques have already been deployed to date and how to apply them in specific tasks, and (iii) main research challenges, open problems, and potential directions of tensor tracking in the future.

1.1 State-of-the-art Tensor Surveys

The very first introduction to TD was provided by Rasmus in [16] two decades ago. This reference offered a tutorial on CP decomposition covering features, properties, methods, and applications in chemometrics. Since then, there have been many published survey papers which provided different points of view on tensor computation. We can broadly divide them into three classes, including (i) surveys on models, methods, and tools for factorizing tensors, (ii) surveys on general tensor problems, e.g., tensor operations, uniqueness, ranks, filtering, spectral analysis,
and complexity, and (iii) surveys on tensor applications. We refer the readers to Tab. 1 for the main contributions of the state-of-the-art surveys on tensors.

Among them, the most cited survey paper is the work of Kolda et al. in [18] that was published more than a decade ago. The survey presented basic multiway models (i.e., CP family and Tucker family) and workhorse algorithms for factorizing tensors under these models. Some applications and software for tensors were also mentioned. The second key survey in the literature is the work of Sidiropoulos et al. in [1] that appeared five years ago. To fill some gaps in the existing surveys on CP and Tucker decompositions of that time, the authors provided an in-depth overview of tensors with respect to the following aspects: uniqueness, ranks, bounds, algorithms, and applications. Moreover, an up-to-date list of software and toolboxes for tensor computation was provided therein. To extend beyond the two standard multiway models, Cichocki et al. conducted a comprehensive tutorial on tensor networks in [22], [37]. Particularly, its coverage includes tensor network models, the associated operations and algorithms, and their applications. Besides, it also highlighted connections of tensor networks to dimensionality reduction and large-scale optimization problems. Very recently, Liu et al. provided a general overview of tensors from the engineer’s point of view in the book [2]. It covers various aspects of tensor computations and decompositions, from operations and well-known multiway representations to tensor-based data analysis techniques and practical applications.

However, to date, we are not aware of any survey paper...
Decompositions

survey on the state-of-the-art online techniques which are capable of factoring tensors in an online fashion. Fig. 1 depicts the organization of the paper. For ease of reference, we summarize in Tab. 2 notations which are frequently used in this paper.

2 Tensor Decompositions

In this section, we briefly describe the background of the five popular tensor decompositions which have already been deployed to factorize streaming tensors in an online fashion. They are CP, Tucker, BTD, tensor-train, and t-SVD.

2.1 CP Decomposition

Under the CP format [16], a tensor \( \mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N} \) can be decomposed into a set of \( N \) matrices \( \{U^{(n)}\}_{n=1}^{N} \) sharing the same number of columns as follows

\[
\mathbf{X} \triangleq \left\{ \{U^{(n)}\}_{n=1}^{N} \right\} = \sum_{i=1}^{r} U^{(1)}(:,i) \circ U^{(2)}(:,i) \circ \cdots \circ U^{(N)}(:,i),
\]

where the so-called tensor factor \( U^{(n)} \) is of size \( I_n \times r \) with \( 1 \leq n \leq N \). The smallest integer \( r \) satisfying (1) is referred to as the CP-rank of \( \mathbf{X} \). CP is among the best memory-saving format for representing high-order tensors, and hence, it can overcome the curse of dimensionality which particularly limits the order of tensors to be analysed. Under certain conditions, CP decomposition is essentially unique up to a permutation and scale. However, its main disadvantage is that finding the true rank \( r \) is known as an NP-hard problem [27]. Even though the CP-rank is given in advance, the best rank-\( r \) approximation of a tensor may not exist [42]. To compute the CP decomposition, one of the most widely-used approaches is based on the alternating least-squares (ALS) technique [18].

2.2 Tucker Decomposition

Under the Tucker format [4], we can factorize \( \mathbf{X} \) into a core tensor \( \mathcal{G} \) of a smaller size and \( N \) factors \( \{U^{(n)}\}_{n=1}^{N} \) as

\[
\mathbf{X} \triangleq \{\mathcal{G}; \{U^{(n)}\}_{n=1}^{N}\} = \mathcal{G} \times_1 U^{(1)} \times_2 \cdots \times_N U^{(N)},
\]

where \( \mathcal{G} \) is of size \( r_1 \times r_2 \times \cdots \times r_N \) \( (r_n \leq I_n) \) and \( U^{(n)} \in \mathbb{R}^{I_n \times r_n} \) is an orthogonal matrix. The vector \( r = [r_1, r_2, \ldots, r_N] \) is called the multilinear rank or rank-\( (r_1, r_2, \ldots, r_N) \) of \( \mathbf{X} \). Tucker
2.3 Block-Term Decomposition

Block-term decomposition (BTD) allows to represent $\mathbf{X}$ as a sum of low multilinear-rank tensors [7]:

$$\mathbf{X} = \sum_{i=1}^{L} \mathbf{G}_i; \{\mathbf{U}_I^{(n)}\}_{n=1}^{N},$$

where $\{\mathbf{G}_i\}_{i=1}^{L}$ with $\mathbf{G}_i \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_N}$ is the set of core tensors, $\{\mathbf{U}_I^{(n)}\}_{n=1}^{N}$ with $\mathbf{U}_I^{(n)} \in \mathbb{R}^{I_n \times r_n}$ is the $n$-th tensor factor, and $r_n \leq I_n \forall n$. The BTD format (3) can be considered as a combination of CP and Tucker. As its name reveals, the basic components in BTD are rank-$(r_1, r_2, \ldots, r_N)$ blocks while they are rank-1 terms in CP. When these blocks are rank-1 tensors (i.e., $r_n = 1 \forall n$), it boils down to CP. When it has only one block (i.e., $L = 1$), BTD becomes the standard Tucker decomposition. It is worth noting that the number of blocks $L$ relies on the block’s size. Like CP, BTD is essentially unique [7]. The common approach to find (3) is also based on the ALS technique [44].

2.4 Tensor-train Decomposition

Tensor-train (TT) decomposition expresses $\mathbf{X}$ as a multilinear product of third-order tensors $\{\mathbf{G}^{(n)}\}_{n=1}^{N}$ according to

$$\mathbf{X} = \mathbf{G}^{(1)} \otimes \mathbf{G}^{(2)} \otimes \cdots \otimes \mathbf{G}^{(N)},$$

where $\mathbf{G}^{(n)} \in \mathbb{R}^{r_{n-1} \times I_n \times r_n}$ is the $n$-th TT-core (aka tensor carriage) with $n = 1, 2, \ldots, N$. Here, $r_0 = r_N = 1$ and the quantities $\{r_n\}_{n=1}^{N}$ are called TT-ranks [5]. This type of TD offers several appealing benefits for representing tensors, especially high-order tensors. Given an arbitrary tensor $\mathbf{X}$, we always find a set of TT-ranks $\{r_n\}_{n=1}^{N}$ satisfying (4) with suitable TT ranks. Besides, its TT-ranks can be effectively estimated in a stable way in contrast to the CP-rank determination [27]. TT also offers a memory-saving representation for tensors and can break the curse of dimensionality like CP. With respect to the implementation, the workhorse algorithm to compute TT is TT-SVD [5].

2.5 T-SVD Decomposition

Tensor SVD (t-SVD) is another multiway extension of SVD for decomposing tensors in which $\mathbf{X}$ is factorized into three tensors $\mathbf{U}$, $\mathbf{G}$, and $\mathbf{V}$ of the same order:

$$\mathbf{X} = \mathbf{U} \ast \mathbf{G} \ast \mathbf{V}^H,$$

where $\mathbf{U}$ and $\mathbf{V}$ are unitary tensors, and $\mathbf{G}$ is a rectangle $f$-diagonal tensor whose frontal slices are diagonal matrices [6]. To define the low-rank tensor approximation under the t-SVD format, the so-called tubal-rank $r_t$ is determined as the number of non-zero tubes of $\mathbf{G}$ (e.g., when the tensor $\mathbf{X}$ is of order 3, $r_t(\mathbf{X}) = \sum_i |\mathbf{G}(i, i, :)| \neq 0$) where 1 is an indicator function). The t-SVD algebraic framework is quite different from the classical multilinear algebra in other types of TD. Thanks to the t-product and Fourier transform, several linear, multilinear operators and other transformations are successfully extended from matrices to tensors, such as transpose, orthogonality, and inverse. In particular, t-SVD can be obtained by computing SVDs in Fourier domain and its performance (i.e., exact recovery with high probability) can be guaranteed under mild conditions [6].

3 Tensor Tracking Formulation

In this section, the problem of tensor tracking is formulated. Specifically, we first divide streaming tensor tracking models into two classes and then construct some terminologies to support the problem statement. They are single-aspect and multi-aspect streaming models. After that, we formulate the general tensor tracking problem which is suitable for many applications.

3.1 Single-aspect Streaming Model

In the classical online setting, we are interested in the decomposition of an $N$-order streaming tensor $\mathbf{X}_t$ fixing all but one dimension. Without loss of generality, we suppose the last dimension is temporal, and hence, we can write $\mathbf{X}_t \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{N-1} \times I_N}$ where $I_N$ is increasing with time.

The following definition of temporal slices is useful to formulate the problem of single-aspect tensor tracking.

**Definition 1 (Temporal slice).** Given a streaming tensor $\mathbf{X}_t \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, we say $\mathbf{Y}_t = \mathbf{X}_t((\cdot, \ldots, i, \cdot)) \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{N-1}}$ is the $t$-th temporal slice of $\mathbf{X}_t$ for $1 \leq t \leq I_N$. Without loss of generality, we assume that $I_N = t$ meaning that at each time instant one new slice of the tensor is observed. Accordingly, the streaming tensor $\mathbf{X}_t$ can be viewed as a set of temporal slices $\{\mathbf{Y}_t\}_{t=1}^{I_N}$. In other words, $\mathbf{X}_t$ is derived from appending the new coming temporal slice $\mathbf{Y}_t$ to the previous observations $\mathbf{X}_{t-1}$ along the time dimension, i.e.,

$$\mathbf{X}_t = \mathbf{X}_{t-1} \oplus \mathbf{Y}_t$$

Generally, $\mathbf{Y}_t$ has the form

$$\mathbf{Y}_t = \mathbf{P}_t \oplus (\mathbf{L}_t + \mathbf{N}_t + \mathbf{O}_t),$$

where $\mathbf{L}_t$ is a low-rank component, $\mathbf{P}_t$ is a binary tensor, $\mathbf{N}_t$ is a noise tensor, and $\mathbf{O}_t$ is a sparse tensor. The data model (7) is a general form which is suitable for many scenarios. For example, $\mathbf{P}_t$ represents missing and observed entries of $\mathbf{Y}_t$; $\mathbf{N}_t$ is an additive white Gaussian noise; and $\mathbf{O}_t$ denotes the sparse outliers. Meanwhile, the low-rank $\mathbf{L}_t$, which can be formulated by any tensor formats, can be static or time-varying.

Based on these terminologies, the problem of single-aspect tensor tracking can be formally stated as follows:

**Single-aspect Tensor Tracking:** At time $t$, given a temporal slice $\mathbf{Y}_t$ and old estimates of $\mathbf{X}_{t-1}$ (e.g., core tensors and tensor factors), we want to track the new estimates of $\mathbf{X}_t = \mathbf{X}_{t-1} \oplus \mathbf{Y}_t$ in time.

3.2 Multi-aspect Streaming Model

In some modern online applications, tensor data may evolve in multiple dimensions/modes over time, and thus, the single-aspect streaming model is not useful for modelling such streaming data. In [45], Fanan-T et al. for the first time introduced the concept of multi-aspect streaming tensors to represent streaming data having more than one dimension increasing with time. Since then, some online algorithms have been developed to deal with the problem of multi-aspect streaming tensor decomposition.

For convenience, we first introduce the definitions of multi-aspect streaming tensors and temporal tubes.

**Definition 2 (Multi-aspect streaming tensor).** A set of $N$-order tensors $\{\mathbf{X}_i\}_{i \geq 1}$ is called a multi-aspect streaming tensor sequence denoted as $\{\mathbf{X}\}$ when $\mathbf{X}_i \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$,
Definition 3 (Temporal tube). Given two successive tensors \( X_{t-1} \) and \( X_t \) derived from the same multi-aspect streaming tensor sequence \( \{X_t\} \), the coming data at time \( t \) can be represented by \( Y_t = X_t \setminus X_{t-1} \) of the same size as \( X_t \) with entries
\[
[Y_t]_{i_1, \ldots, i_N} = \begin{cases} 
[X_t]_{i_1, \ldots, i_N} & \text{if } I_{1n}^{t-1} < i_n \leq I_{1n}^t, \\
0 & \text{otherwise}
\end{cases}
\]
for \( 1 \leq n \leq N \). We say that the non-zero entries in \( Y_t \) are temporal tubes.

Now, we can state the problem of multi-aspect tensor tracking as follows:

**Multi-aspect Tensor Tracking:** At time \( t \), given temporal tubes in \( Y_t \), and old estimates of \( X_{t-1} \) (e.g., core tensors and tensor factors), we want to track the new estimates of \( X_t = X_{t-1} \cup Y_t \) in time.

It is worth noting that the single-aspect tensor tracking problem also belongs to the class of multi-aspect tensor tracking where most of the tensor dimensions \( I_n \) are constant by setting \( W_n^t = 0 \), except the last one \( I_N^t \). Besides, temporal slices may be regarded as frontal slices of the tensor \( Y_t \) defined as in (8).

### 3.3 General Formulation of Optimization

We here provide a general formulation of tensor tracking which can be used in many applications. In particular, the optimization problem can be written as
\[
\min_{(g, \{u_t\}, O)} \left[ \sum_{r=1} \beta_r \ell(Y_t, P_r, \{g\}, \{u_t\}, O) + \rho_c R_c(\{g\}) + \rho_i R_i(\{u_t\}) + \rho_o R_o(O) \right] + \lambda_c L_c(\{g\}) + \lambda_o L_o(\{u_t\}),
\]
subject to application dependent constraints.

Here, \( \{g\} \) and \( \{u_t\} \) denote the set of core tensors and tensor factors respectively, while \( O \) is to represent data corruptions by impulsive noise or outliers. Specifically, the three terms in the second line of (9) are used to present regularizations or penalty terms imposed on parameters of interest. The last two penalty terms of (9) are for the application orientation. The main loss function \( \ell(.) \) is defined to minimize the residual errors between the estimations and observations.

### 4 Streaming CP Decomposition

The primary objective of this section is to provide technical descriptions of the state-of-the-art online techniques for factorizing streaming tensors under the CP format. In the literature, there are many streaming CP algorithms and they can be categorized into the following groups: (i) subspace-based methods, (ii) block-coordinate descent methods, (iii) Bayesian inference, and (iv) multi-aspect streaming CP decompositions. The three former groups are particularly developed for single-aspect streaming models, while the latter is dedicated to the factorization of tensors having more than one temporally varying mode. The readers are referred to Tabs. 3 and 4 for quick comparisons of the existing streaming CP decomposition algorithms. In what follows, we take each group into consideration.
The second approach is based on the block-coordinate descent (BCD) framework in which we decompose the main optimization into two main stages at each time $t$: (i) estimate the temporal factor $U^{(N)}_t$ given $\{U^{(n)}_{t-1}\}_{n=1}^{N-1}$, and (ii) update the non-temporal factor $U^{(n)}_t$ with $1 \leq n \leq N-1$ in sequential or parallel given $U^{(N)}_t$ and the remaining factors. Many tracking algorithms adopt this approach for estimating the low-rank approximation of streaming tensors over time in the literature. We can list here some: TeCPDG [48], OLCP [49], OLSTEC [56], CP-stream [54], SPADE [59], SOFIA [61], iCP-AM [70], ACP [64], and RACP [65]. In what follows, we review their strategy in each stage.

In stage 1, the general formulation of the optimization to estimate the last row $u^{(N)}_t$ of $U^{(N)}_t$ can be given by:

$$
\min_{u^{(N)}_t} \| P_t \odot (y_t - \{U^{(1)}_{t-1}\}_{n=1}^{N-1} u^{(N)}_t) \|_2^2 + \rho_1 \| u^{(N)}_t \|_2^2 + \rho_2 \| O \|_1, \tag{12}
$$

where $\rho_1 \| u^{(N)}_t \|_2^2$ is for avoiding the ill-posed computation and $\rho_2 \| O \|_1$ promotes the sparsity in $O$. Then, the temporal factor $U^{(N)}_t$ is obtained by appending the recent updated $u^{(N)}_t$ to the singular vector of the reshaped matrix from $H^{(1)}_t$; $i$ can provide a good estimate of $U^{(1)}_t$ and $U^{(2)}_t$ respectively, i.e.,

$$
\begin{align*}
\{ b_t, \lambda_t, a_i \} &\leftarrow \text{SVD} (\text{reshape}(H^{(1)}_t, i, [I, I])) \\
\{ U^{(1)}_t, i \} &\leftarrow a_i^* \text{ and } U^{(2)}_t (i, i) = \lambda_t b_t
\end{align*}
$$

Computation of SVD may be expensive when dealing with large-scale streaming tensors, we can use the alternative update based on power iteration as follows

$$
\begin{align*}
\{ H^{(1)}_t \} &\leftarrow \text{reshape} (H^{(1)}_t, i, [J \times I]) \\
\{ U^{(1)}_t \} &\leftarrow (H^{(1)}_t)^T U^{(2)}_{t-1} \\
\{ U^{(2)}_t \} &\leftarrow \frac{H^{(1)}_t U^{(1)}_t}{\| H^{(1)}_t U^{(1)}_t \|_2}
\end{align*}
$$

As these algorithms are only designed for tracking third-order streaming tensors, there are still rooms to develop subspace-based methods capable of handling $N \geq 4$.

### 4.2 Block-Coordinate Descent

The second approach is based on the block-coordinate descent (BCD) framework in which we decompose the main optimization into two main stages at each time $t$: (i) estimate the temporal factor $U^{(N)}_t$ given $\{U^{(n)}_{t-1}\}_{n=1}^{N-1}$, and (ii) update the non-temporal factor $U^{(n)}_t$ with $1 \leq n \leq N-1$ in sequential or parallel given $U^{(N)}_t$ and the remaining factors. Many tracking algorithms adopt this approach for estimating the low-rank approximation of streaming tensors over time in the literature. We can list here some: TeCPDG [48], OLCP [49], OLSTEC [56], CP-stream [54], SPADE [59], SOFIA [61], iCP-AM [70], ACP [64], and RACP [65]. In what follows, we review their strategy in each stage.

In stage 1, the general formulation of the optimization to estimate the last row $u^{(N)}_t$ of $U^{(N)}_t$ can be given by:

$$
\min_{u^{(N)}_t} \| P_t \odot (y_t - \{U^{(n)}_{t-1}\}_{n=1}^{N-1} u^{(N)}_t) \|_2^2 + \rho_1 \| u^{(N)}_t \|_2^2 + \rho_2 \| O \|_1, \tag{12}
$$

where $\rho_1 \| u^{(N)}_t \|_2^2$ is for avoiding the ill-posed computation and $\rho_2 \| O \|_1$ promotes the sparsity in $O$. Then, the temporal factor $U^{(N)}_t$ is obtained by appending the recent updated $u^{(N)}_t$ to the
old estimate $U_{t-1}^{(N)}$. Most of the existing BCD-based tracking algorithms suppose that observations are outlier-free (i.e., without outliers) and, hence, they apply the regularized/randomized least-squares methods for solving (12). In the presence of sparse outliers, (12) can be effectively minimized by ADMM or shrinkage-thresholding solvers, as presented in SOFIA [61] and RACP [65].

In stage 2, the non-temporal factors $\{U_i^{(n)}\}_{n=1}^{N-1}$ can be derived from solving the following optimization

$$
\min_{U_i^{(n)}} \sum_{t=1}^{T} \beta_t^{-1} \left\| P_t^{(n)} U_i^{(n)} W_t^{(n)} + Q_t^{(n)} - Y_t^{(n)} \right\|_F^2 + \rho_i \mathcal{R}_i(U_i^{(n)}),
$$

where $\mathcal{R}_i(U_i^{(n)})$ is a regularization term on $U_i^{(n)}$ and

$$
W_t^{(n)} = \left\{ \begin{array}{ll}
(1-n^{-1}) U_i^{(n)} + u_i^T & [\text{Jacobi}],
\end{array} \right.
$$

Here, we can apply one of the two iterative schemes to update $U_i^{(n)}$: the Jacobi scheme supports the parallel and/or distributed processing while the Gauss-Seidel scheme is useful for a sequential (serial) one. The regularization can be $\|U_i^{(n)}\|_F^2$ for smoothness, $\|U_i^{(n)} - U_{i-1}^{(n)}\|_F^2$ for slow time-variation, or $U_i^{(n)} \geq 0$ for non-negativity constraints. Next, we review two common types of solver for optimizing (13): adaptive least-squares filters and stochastic gradient solvers.

**a) Adaptive Least-Squares (LS) Filters.** We can see that the first term of (13) is of a weighted LS form very common in adaptive filtering while the second one is to regularize the estimators. Accordingly, (13) can be effectively minimized by adaptive LS filters in general and recursive least-squares (RLS) filters in particular.

In [56], Kasai proposed an exponential RLS algorithm called OLSTEC to minimize (13) when the observations are outlier-free. OLSTEC is, however, designed for third-order streaming tensors only and its complexities are relatively high compared to other algorithms. Thanh et al. in [64] proposed another RLS algorithm called ACP which is capable of dealing with big streaming tensors of higher order ($N \geq 4$). ACP is fast and requires much lower complexity than OLSTEC. Very recently, the same authors in [65] proposed a sliding-window version of ACP robust to both sparse outliers and missing data, namely RACP. Interestingly, three algorithms belong to the class of provable online CP algorithms in which their convergence is guaranteed under certain conditions. In [51], Vandecappelle et al. introduced a nonlinear least-squares (NLS) algorithm for computing the streaming CP decomposition of third-order tensors. In particular, the authors recast the objective function of (13) into a truncated exponential window one by incorporating a weighting matrix $L = \text{diag}(\{0, \ldots, 0, \beta^{L-2}, \beta, 1\})$ and applied a NLS solver to track the tensor factors with time. Following the same line, Smith et al. in [54] proposed another online CP algorithm called CP-stream. This algorithm has the potential to factorize higher-order streaming tensors as well as support constraints on the CP tracking such as smoothness and nonnegativity.

**b) Stochastic Gradient Solvers.** Instead of optimizing (13) directly, we can minimize its $t$-th summand:

$$
\min_{U_i^{(n)}} \left\| P_t^{(n)} U_i^{(n)} W_t^{(n)} - Q_t^{(n)} U_i^{(n)} W_t^{(n)} + \rho_i \mathcal{R}_i(U_i^{(n)}) \right\|_F^2
$$

Three algorithms TeCPSGD [48], OLCP [49], and SOFIA [61] adopt this replacement for tracking tensor factors with time. The main difference among them is the type of $\mathcal{R}_i(.)$. Besides, they obtain different forms of update:

- [SOFIA]: $U_i^{(n)} = U_i^{(n)} + \gamma_t \Delta U_i^{(n)}$,
- [TeCPSGD]: $U_i^{(n)} = (1 - \frac{\beta_t}{\eta_t}) U_i^{(n)} + \frac{1}{\eta_t} \Delta U_i^{(n)}$,
- [OLCP]: $U_i^{(n)} = \mathcal{P}_t^{(n)} (Q_t^{(n)})^{-1}$ with

$$
\mathcal{P}_t^{(n)} = \mathcal{P}_t^{(n-1)} + \Delta \mathcal{P}_t^{(n)}
$$

and

$$
Q_t^{(n)} = Q_t^{(n-1)} + \Delta Q_t^{(n)}.
$$

Here, $\gamma_t, \eta_t, \Delta U_i^{(n)}, \Delta P_t^{(n)}$, and $\Delta Q_t^{(n)}$ can be obtained from

$$
\{U_i^{(n)}\}_{n=1}^{N-1}
$$

and the error tensor $\Delta Y_t = \mathcal{P}_t \ominus (Y_t - \mathcal{O}_t - \mathcal{O}_t \ominus \{U_i^{(n)}\}_{n=1}^{N-1} \circ U_i^{(n)})$. It is worth noting that SOFIA is capable of dealing with sparse corruptions. TeCPSGD has the ability to track tensors from missing observations, while OLCP can handle streaming tensors of order greater than 3.

In [70], Zeng et al. proposed an incremental ALS algorithm called iCP-AM to minimize a reinforced version of (14) which is defined as

$$
\min_{U_i^{(n)}} \left\| \mathbf{Y}_t^{(n)} - \mathbf{U}_t^{(n)} \circ \mathbf{V}_t^{(n)} \right\|_F^2,
$$

where

$$
\mathbf{V}_t^{(n)} = (\bigodot_{i=1}^{n-1} \mathbf{U}_i^{(i)}) \circ (\bigodot_{i=n+1}^{N-1} \mathbf{U}_t^{(i)}).
$$

An appealing feature of iCP-AM against other online CP algorithms is that it has a strategy to deal with the variation of the CP rank over time, i.e., to change the number of low-rank components throughout the tracking process.

In parallel, Gujral et al. in [59] proposed an online algorithm called SPADE for tracking tensors under the PARAFAC2 format. Specifically, SPADE tracks a fixed (non-temporal) factor along one mode and allows the other tensor factors (modes) to vary with time. Thanks to its stochastic design, SPADE is fast and memory-efficient. However, the stationary assumption that time variation or concept drift is not allowed limits its applicability.

### 4.3 Bayesian Inference

Besides, another good approach for dealing with the problem of streaming CP decomposition is Bayesian inference. The state-of-the-art Bayesian-based streaming CP decomposition algorithms are POST [55], BRST [52], and SBDT [71]. In general, three algorithms start with a prior distribution of unknown parameters and then infer a posterior that best approximates the joint distribution of these parameters on the arrival of new streaming data. The estimated posterior is then used as the prior for the next update. In this subsection, we briefly describe the two online Bayesian inference frameworks which were already used for tensor tracking: (i) streaming variational Bayes (SVB) and (ii) assumed-density filtering (ADF). Also, prior distributions of parameters of interest are reviewed.

#### a) Streaming Variational Bayes

The two former algorithms POST and BRST adopted the SVB framework [72] which is based on the following Bayes’ rule:

$$
p(\Theta | \mathbf{X}_{t-1} \in \mathcal{N}, \mathbf{Y}_t) \propto p(\mathbf{Y}_t | \Theta) p(\Theta | \mathbf{X}_{t-1}),
$$

where $\Theta$ denotes the parameters of interest, e.g., tensor factors, CP rank, and other parameters. On the arrival of $\mathbf{Y}_t$, SVB first uses the current posterior $q_{t-1}(\Theta) := p(\Theta | \mathbf{X}_{t-1})$ as the prior of $\Theta$, and then integrates with the likelihood of $\mathbf{Y}_t$ to obtain

$$
\hat{p}_t(\Theta) = p(\mathbf{Y}_t | \Theta) q_{t-1}(\Theta),
$$
which can be served as an approximation of the joint distribution \( p(\Theta, \mathbf{Y}) \) up to a scale factor. The variational posterior \( q_t(\Theta) \) is derived from maximizing the variational model evidence lower bound (ELBO) \( \mathcal{L}(q(\Theta)) = \mathbb{E}_q[\log \frac{p(\mathbf{Y}, \Theta)|q(\Theta)}{q(\Theta)}] \) which is equivalent to minimizing the Kullback-Leibler (KL) divergence:

\[
\min_q \left[ \text{KL} \left( q(\Theta) \parallel \hat{p}_t(\Theta) \right) \right] = \int q(\Theta) \log \frac{q(\Theta)}{\hat{p}_t(\Theta)} d\Theta.
\]

The optimized form of \( q_t(\Theta) \) of (21) can be given by

\[
\log q_t(\Theta) = \mathbb{E}_q[\log \frac{p(\mathbf{Y}, \Theta)|q(\Theta)}{q(\Theta)}] + \text{const},
\]

where \( \mathbb{E}_q[\log \frac{p(\mathbf{Y}, \Theta)|q(\Theta)}{q(\Theta)}] \) is an expectation w.r.t. \( q \) over all but \( \Theta_t \).

b) Assumed-Density Filtering. The latter algorithm, SBDT, applied the ADF framework to infer the posterior distribution \( q_t(\Theta) \) over \( \Theta \). Particulary, ADF is an incremental learning framework that allows for computing the approximate posteriors in Bayesian inference for stochastic processes [73]. The ADF framework is also grounded on the Bayes’ rule (19) but utilizes a distribution from the exponential family (e.g., Gaussian distribution) to approximate the current posterior. Instead of minimizing the KL divergence or maximizing the variational ELBO like SVB, ADF projects \( \hat{p}_t(\Theta) \) into the selected distribution through moment matching to obtain \( q_t(\Theta) \).

c) Prior Distributions over \( \Theta \). We list common prior distributions over \( \Theta \) which were used by POST, BRST, and SBDT.

Prior distribution of tensor factors: All three algorithms assume that the prior over tensor factors is derived from the following Gaussian distribution which is controlled by the hyperparameter \( \lambda = [\lambda_1, \lambda_2, \ldots, \lambda_r] \):

\[
p(U^{(n)}|\lambda) = \prod_{i=1}^{I_n} N(u_i^{(n)}|0, \Lambda^{-1}), \forall n \in [1, N],
\]

where \( u_i^{(n)} \) is the \( i \)-th row of \( U^{(n)} \) and \( \Lambda = \text{diag}(\lambda) \) denotes the inverse covariance matrix. Here, \( \lambda \) is supposed to follow a Gamma distribution:

\[
p(\lambda) = \prod_{j=1}^{r} \text{Gam} \left( \lambda_j | c_j, d_j \right),
\]

where \( \text{Gam}(\lambda_j | c_j, d_j) = \frac{d_j c_j^{d_j} \lambda_j^{-c_j-1} e^{-d_j \lambda_j}}{\Gamma(c_j)} \) with \( \Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx \). Specifically, the mean and variance of \( \text{Gam}(\lambda_j | c_j, d_j) \) are, respectively, \( c_j/d_j \) and \( c_j/d_j^2 \) which aim to control the magnitude of \( \lambda \).

Prior distribution of noises: The noise is often assumed to be Gaussian, i.e., \( \mathcal{N}_r \sim \prod_{i=1}^{I_r} \mathcal{N}(0, \tau^{-1}) \) with a noise precision \( \tau > 0 \). The parameter \( \tau \) is further assigned to another Gamma distribution as follows:

\[
p(\tau) = \prod_{i=1}^{I_r} N(\tau_i | c_i, d_i),
\]

where \( \text{Gam}(\tau_i | c_i, d_i) = \frac{d_i c_i^{d_i} \tau_i^{-c_i-1} e^{-d_i \tau_i}}{\Gamma(c_i)} \). Specifically, the mean and variance of \( \text{Gam}(\tau_i | c_i, d_i) \) are, respectively, \( c_i/d_i \) and \( c_i/d_i^2 \) which aim to control the sparsity precision parameter. If the value of \( \gamma_{i_{1:i_N}} \) is large, the corresponding entry in \( \Omega \) is likely to have a small magnitude. By controlling the value of \( \gamma_{i_{1:i_N}} \), we can control the sparsity of \( \Omega \).

Prior distribution of NN’s weights: SBDT in [71] incorporates neural networks (NN) into tensor factorization. SBDT assigns a spike-and-slab prior distribution over NN weights to sparsify the network. Each weight \( \omega_{mjt} = |W_{mjt}| \) of NN is particularly sampled from

\[
p(\omega_{mjt}|s_{mjt}) = s_{mjt} N(\omega_{mjt}|0, \sigma_\omega^2) + (1 - s_{mjt}) \delta(\omega_{mjt}),
\]

where \( \delta(.) \) denotes the delta function and the binary selection indicator \( s_{mjt} \) is derived from \( p(s_{mjt}) = \text{Bern}(s_{mjt} | \rho_0) = \rho_0^{s_{mjt}} (1 - \rho_0)^{1-s_{mjt}} \).

4.4 Multi-aspect Streaming CP Decomposition

In the literature, there are some algorithms capable of tracking multi-aspect streaming tensors under the CP format, such as MAST [74], OR-MSTC [75], InParTen2 [76], and DisMASTD [77]. We refer the readers to Tab. 4 for their key features. In what follows, we first describe the main dynamic tensor decomposition (DTD) framework shared by most of these algorithms and then highlight their characteristics in the following text.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Outliers?</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>High-order? (N ≥ 4)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Distributed?</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

TABLE 4

Main Features of Multi-aspect Streaming CP Decomposition Algorithms.
η > 0 is a regularization parameter. Since terms of (29) are all convex, it can be effectively minimized by several methods. In particular, MAST applies an ADMM solver to minimize (29) in order to balance the trade-off between effectiveness and efficiency in tracking process.

Since MAST is not designed for handling sparse outliers, Najafi et al. in [75] introduced a robust version of it called OR-MSTC. In the presence of sparse outliers, they proposed to regularize the objective function of (27) by adding an ℓ1-norm regularization term λ∥O∥1 and replacing Yt with Yt − O in the first term of (ℓ.) in (29). Because the term λ∥O∥1 is convex, OR-MSTC also adopts the well-known ADMM method in a similar way to MAST.

In [76], Yang et al. proposed a distributed version of MAST called InParTen2. Thanks to Apache Spark3, it can handle large-scale streaming tensors efficiently with a limited memory. However, the use of InParTen2 is limited for 3-order streaming tensors only. In [77], Yang et al. introduced another distributed method called DisMASTD capable of dealing with tensors of higher order. One of appealing feature of DisMASTD is that it can avoid repetitive computation and reduce network communication cost.

5 STREAMING TUCKER DECOMPOSITION

We can broadly categorize the streaming Tucker decomposition algorithms into three main classes: (i) online tensor dictionary learning, (ii) tensor subspace tracking, and (iii) multi-aspect streaming Tucker decomposition. Specifically, the first two classes are designed for two specific cases of single-aspect streaming Tucker decompositions, while the latter class is for multi-aspect streaming tensors.

5.1 Online Tensor Dictionary Learning

In the class of online tensor dictionary learning methods, we are particularly interested in a specific case of single-aspect streaming Tucker decomposition where the underlying tensor XT ∈ R I1×⋯×IN−1×T – which represents a set of T data streams {Yi(T)}T t=1 of the same size I1 × I2 ×⋯×IN−1 is supposed to be modelled by

\[ \mathbf{X}_T = \left[ \mathbf{G}_T; \left\{ U^{(n)}_{t} \right\}_{n=1}^{N-1}, I_T \right] \]

where the core GT is of size r1 ×⋯×rN−1 × T (i.e., rN = T), the tensor factors \( \{ U^{(n)}_{t} \}_{n=1}^{N-1} \) ∈ R I1×r1 are of fixed size, and the last factor \( U^{(N)}_{T} \) is an identify matrix. Specifically, the t-th temporal slice of YT is expressed as

\[ Y_t = \left[ \mathbf{G}_t; \left\{ U^{(n)}_{t} \right\}_{n=1}^{N-1} \right], t = 1, 2, \ldots, T \]

where \( \mathbf{G}_t \in \mathbb{R} I_1 × I_2 ×⋯×I_N−1 \) is the t-th slice of the core tensor GT. The primary objective here is to estimate GT and incrementally update \( \{ U^{(n)}_{t} \}_{n=1}^{N-1} \) on the arrival of YT at each time t. In what follows, we review two main approaches to handle this problem.

1. Apache Spark: https://spark.apache.org/
background and detect anomalies in applications of computer vision. Since RTSL still applies directly the DTA framework, its complexity is relatively high. Thus, it may become inefficient for handling large-scale and high dimensional streaming data.

Some other algorithms for streaming Tucker decomposition belonging to this group were presented in [87]–[89], [102], focusing on specific applications such as dynamic brain analysis, smart city services, cyber-physical-social networks and systems.

**b) Online Multimodal Dictionary Learning.** Another good strategy for the problem of single-aspect tensor tracking is to apply online multimodal dictionary learning (OMDL) techniques. As OMDL is a stochastic version of the multimodal dictionary (multilinear subspace) learning [103], it allows estimating dictionaries (i.e., tensor factors) with one-pass processing. In the literature, there exist some algorithms applying OMDL for tracking the low multilinear-rank component of streaming tensors with time, such as OTDL [92], OD1 [104], ORLT[M [105], ORLTR [106], D-L1-Tucker [96], and ROLTD [107].

The two former algorithms OTDL and OD1 adopt the typical two-step learning procedure to track the tensor factors over time,
namely (i) tensor coding or inference of coefficients in the core tensor and (ii) dictionary update per each tensor mode.

**Step 1: Tensor Coding.** When $Y_t$ is observed, the general formulation of optimization for this step is given by:

$$\min_{U_t} \left\| Y_t - \left[ G; \{ U_{t-1}^{(n)} \}_{n=1}^{N-1} \right] \right\|_F^2 + \rho_c \mathcal{R}_c(G),$$

(35)

where $\mathcal{R}_c(\cdot)$ is a regularization term on the core tensor $G$ to promote sparsity or nonnegativity for instance. Since the first term of (35) is differentiable while the second term may admit a proximal operator (e.g., $\ell_p$-norm), OTDL, ODL, and ROLTD applied proximal methods to minimize it.

**Step 2: Dictionary Update.** When $G_t$ is estimated, the BCD framework can be used to update $U_t^{(n)}$. Specifically, both algorithms optimize the following minimization:

$$\min_{U_t} \sum_{t=1}^{t} \beta^{t-1} \left\| Y_t - \left[ G_t; \{ U_{t-1}^{(n)} \}_{n=1}^{N-1} \right] \right\|_F^2 + \rho_c \mathcal{R}_c(U^{(n)}),$$

(36)

with a penalty term $\rho_c \mathcal{R}_c(\cdot)$ on $U^{(n)}$. Interestingly, (36) can be recast into the standard least-squares cost function which is very common in adaptive filtering theory. Accordingly, OTDL developed an effective recursive least-squares (RLS) solver to optimize it. Meanwhile, ODL used the stochastic gradient descent method to estimate $U_t^{(n)}$ with a low cost.

The next two algorithms ORLTM and OLRTR, on the other hand, estimated the tensor factors without the need of tensor coding. In particular, the tensor factor $U^{(n)}$ is directly derived from the following optimization

$$\min_{U_t} \sum_{t=1}^{t} \beta^{t-1} \ell(Y_t, U^{(n)}) + \rho_c \mathcal{R}_c(U^{(n)}),$$

(37)

where the loss function $\ell(\cdot, \cdot)$ is defined as

$$\ell(Y_t, U^{(n)}) = \min_{R^{(n)}, O^{(n)}} \left\| Y_t - U^{(n)} R^{(n)} - O^{(n)} \right\|_F^2 + \lambda_1 |O^{(n)}|_1 + \lambda_2 \mathcal{R}_c(R^{(n)}).$$

(38)

Here, $R^{(n)}$ and $O^{(n)}$ play the role of the coefficient and the error, respectively. The main difference between ORLTM and OLRTR is the type of $\mathcal{R}_c(\cdot)$ used. Specifically, OLRTR uses the simple Frobenius norm regularization $\mathcal{R}_c(R^{(n)}) = ||R^{(n)}||_F$, while ORLTM reinforces $R^{(n)} = W^{(n)} Z^{(n)}$ and then forms $\mathcal{R}_c(R^{(n)}) = ||W^{(n)}||_F^2 + ||Z^{(n)}||_F^2$. Intuitively, the minimization (37) may be regarded as a robust version of (36) which aims to deal with sparse corruptions. Also, the minimization (38) is not difficult to solve since its terms are all convex. Hence, both ORLTM and OLRTR applied the RLS method to update $U_t^{(n)}$ over time.

In [96], Chachlakis et al. proposed a streaming Tucker decomposition called D-L1-Tucker for dealing with streaming tensors. D-L1-Tucker shares the same objective function with ORLTM and OLRTR, but adopts a different approach to handle data corruptions. Particularly on the arrival of $Y_t$, D-L1-Tucker first identifies whether $Y_t$ is an anomaly or not based on its reliability which is defined as

$$r_t = \left\| Y_t; \{ U_{t-1}^{(n)} \}_{n=1}^{N-1} \right\|_F^2. \left| Y_t \right|_{Y_t}. \left(39\right)$$

If $r_t \leq \tau$ where $\tau \in [0, 1]$ is a predefined threshold, $Y_t$ is labelled as an outlier slice and then it is disregarded. Otherwise, $X_t$ is considered as reliable and useful for tracking process. In such a case, D-L1-Tucker appends $Y_t$ to the memory set $\mathcal{Z}_t = \mathcal{Z}_{t-1} \cup \mathcal{Y}_t$ and then applies the batch L1-HOOI algorithm proposed in [108] for factorizing $\mathcal{Z}_t$ in order to obtain tensor factors. After that, $\mathcal{Z}_t$ is re-updated by removing the oldest measurement for the next processing. D-L1-Tucker requires a good batch initialization and its tracking ability is dependent on the threshold $\tau$ and the memory size $M$ to store $\mathcal{Z}_t$.

**5.2 Tensor Subspace Tracking**

Apart from the model (30), $X_T \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times T}$ and its $t$-th temporal slice $Y_t$ with $t \leq T$ can be modelled as follows

$$X_t = \left[ G; \{ U^{(n)} \}_{n=1}^{N} \right], \quad Y_t = \left[ G; \{ U^{(n)} \}_{n=1}^{N-1} \right] U_t^{(N)},$$

(40)

where the core tensor $G \in \mathbb{R}^{r \times r \times \cdots \times r \times N}$ and $U^{(n)}_{N-1} \in \mathbb{R}^{I_1 \times r_n}$ with $U^{(n)} \in \mathbb{R}^{I_1 \times r_n}$ are of fixed size except the last factor $U^{(N)} \in \mathbb{R}^{T \times r_n}$, and $U_t^{(N)} \in \mathbb{R}^{I_1 \times r_n}$ is the $t$-th row of $U^{(N)}$, see Fig. 4 for an illustration (when $N = 3$). At time $t$, given old estimations $Y_{t-1}$ and $U_{t-1}$, we are interested in tracking $G_t$, $U_t^{(N)}$, and $U_{t-1}^{(N)}$ which can compactly represent the temporal slice $Y_t$. We refer this problem to as tensor subspace tracking.

It is worth mentioning that single-aspect streaming CP methods also belong to this class as the core tensor $G$ is constrained to be identity. In the literature, there exist some tensor subspace tracking methods which have the potential to deal with a general case of $G$. Each method adopts a different strategy to factorize streaming tensors. In what follows, we briefly describe their main features in chronological order.

**a) Augmented Projection**. In [85], Baskaran et al. introduced the so-called LRUT algorithm (which stands for Low-Rank Updates to Tucker decomposition) using a randomized projection technique for tracking the low multilinear-rank approximation of streaming tensors over time. When a data stream arrives, LRUT first projects it onto an extended tensor subspace and then forms an augmented core tensor. Specifically, LRUT adds a few more random dimensions to the current tensor subspace defined by old estimations of the tensor factors. The inclusion of some random vectors here plays a role of noise perturbation aimed to prevent the main optimization from getting stuck in local optima. Next, LRUT performs the standard Tucker decomposition (e.g., batch HOSVD or HOOI) on the resulting augmented core tensor to update tensor factors. In this way, we can avoid the computation of SVD on unfolding matrices of the full tensor which is highly expensive in an online setting. However, its computational complexity is still relatively high since LRUT uses several orthogonalization operations on augmented tensor factors and unfolding matrices of the projected tensor slice.

**b) Riemannian Optimization**. In [86], Kasai et al. developed a Riemannian manifold preconditioning approach for tensor completion. Specifically, its stochastic version can be adapted for

2. This name stems from the following observation: we can recast (40) into the standard vector-matrix form $y_t = D u_t$, where $y_t = vec(Y_t)$, $u_t = \left( u_t^{(N)} \right)^T$ and $D$ is the transpose of the mode-$N$ unfolding matrix $\left[ G; \{ U^{(n)} \}_{n=1}^{N-1} \right]$. Intuitively, it may be regarded as the data model which is very common and widely used in the problem of subspace tracking where we wish to incrementally update $D$ on the arrival of $y_t$ at each time $t$. Since the subspace matrix $D$ has a tensor structure, we label this problem as “tensor subspace tracking” without hesitation.
factorizing incomplete streaming tensors in an online fashion. Since the Tucker format provides an effective representation for tensors in the manifold $\mathcal{M}_r = \{ \mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N} | \text{rank}(\mathbf{X}) = r = [r_1, r_2, \ldots, r_N] \}$, Riemannian optimization can offer a good approach for tensor decomposition and completion [109]. Accordingly, the authors proposed an efficient Riemannian gradient based method to estimate the low multilinear-rank component of tensors. The proposed method consists of a rank-one Riemannian gradient computation and a retraction step. Specifically, a novel Riemannian metric on the tangent space of $\mathcal{M}_r$ and its quotient manifold was introduced to enable the Riemannian optimization framework. Furthermore, a map that combines all retractions on the individual manifolds of tensor factors was used to transform the estimations to the tensor manifold.

c) Bayesian Inference. In [97], Fang et al. introduced a Bayesian streaming Tucker decomposition method called BASS-Tucker for handling streaming sparse tensors. Similar to Bayesian methods for streaming CP decomposition, BASS-Tucker adopts the streaming variational Bayes (SVB) framework to infer the posterior of parameters of interest (e.g., tensor core, Tucker) and nuisance parameters) over time. In addition, BASS-Tucker also utilizes the same priors for the tensor factors and noise variance except that of the core tensor. Here, the following spike-and-slab prior is used to model the core tensor:

$$ p(\mathbf{S}|\rho_0) = \prod_{j_1=1}^{r_1} \cdots \prod_{j_N=1}^{r_N} \text{Bern}(s_{j_1,\ldots,j_N}|\rho_0), \quad (41) $$

$$ p(\mathbf{G}_S) = \prod_{j_1=1}^{r_1} \cdots \prod_{j_N=1}^{r_N} s_{j_1,\ldots,j_N} N(g_{j_1,\ldots,j_N}|0, \sigma_0^2) + (1-s_{j_1,\ldots,j_N})\delta(g_{j_1,\ldots,j_N}), \quad (42) $$

where $\mathbf{S} \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_N}$ is a binary tensor, $\text{Bern}(\cdot|\rho_0)$ is the Bernoulli distribution with probability $\rho_0$, and $\delta(\cdot)$ is the Delta function. We refer the readers to subsection 4.3 for details on prior distributions of $\{\mathbf{U}^{(n)}\}_{n=1}^{N-1}$ and other model parameters as well as how the SVB framework works.

d) Block-Coordinate Descent. There are three online Tucker algorithms using the BCD framework, including ATD [64], RTNTD [99] and BK-NTD [99]. In general, they go through the following stages when $\mathbf{Y}_t$ arrives:

**Stage 1:** Estimate the factor $\mathbf{U}^{(N)}_t$ given old estimations $\mathbf{G}_{t-1}$ and $\{\mathbf{U}^{(n)}_{t-1}\}_{n=1}^{N-1}$. Generally, $\mathbf{U}^{(N)}_t$ can be derived from

$$ \min_{\mathbf{U}^{(N)}_t} \left\| \mathbf{Y}_t - \left[ \mathbf{G}_{t-1}: \{\mathbf{U}^{(n)}_{t-1}\}_{n=1}^{N-1}, \mathbf{U}^{(N)}_t \right] \right\|_F^2 + \rho_c R_c(\mathbf{U}^{(N)}_t). \quad (43) $$

**Stage 2:** Estimate the tensor factor $\mathbf{U}^{(n)}_t$ given $\mathbf{U}^{(N)}_t$, old estimation of $\mathbf{U}^{(n)}_{t-1}$ and the remaining factors, $1 \leq n \leq N-1$. The main optimization can be given by

$$ \min_{\mathbf{U}^{(n)}_t} \sum_{\tau=1}^t \beta^{t-\tau} \ell(\mathbf{Y}_t, \mathbf{U}^{(n)}_t) + \rho_u R_u(\mathbf{U}^{(n)}_t), \quad (44) $$

where $\ell(\mathbf{Y}_t, \mathbf{U}^{(n)}_t) = \| \mathbf{Y}_t - \mathbf{U}^{(n)}_t \mathbf{W}^{(n)}_t \|_F^2$, $\mathbf{Y}^{(n)}_t$ and $\mathbf{W}^{(n)}_t$ are respectively the mode-$n$ unfolding matrices of $\mathbf{Y}_t$ and $\mathbf{W}_t$. Here, $\mathbf{W}_t$ is defined as $\mathbf{W}_t = [\mathbf{G}_{t-1}; \{\mathbf{U}^{(n)}_{t-1}\}_{n=1}^{N-1}, \mathbf{U}^{(N)}_t]$. Here, $\mathbf{W}_t$ is defined as $\mathbf{W}_t = [\mathbf{G}_{t-1}; \{\mathbf{U}^{(n)}_{t-1}\}_{n=1}^{N-1}, \mathbf{U}^{(N)}_t]$.

**Stage 3:** Estimate the core tensor $\mathbf{G}_t$ given $\mathbf{G}_{t-1}$, $\mathbf{U}^{(N)}_t$, and $\{\mathbf{U}^{(n)}_{t-1}\}_{n=1}^{N-1}$ particularly from

$$ \min_{\mathbf{G}_t} \sum_{\tau=1}^t \beta^{t-\tau} \| \mathbf{Y}^{(1)}_t - \mathbf{U}^{(1)}_t \mathbf{G}^{(1)}_t \mathbf{Z}^T \|_F^2 + \rho_c R_c(\mathbf{G}_t), \quad (45) $$

where $\mathbf{Z}_t = \mathbf{u}_t \otimes \bigotimes_{n=2}^{N} \mathbf{U}^{(n)}_t$.

Here, $R_u(\cdot)$, $R_c(\cdot)$, and $R_{ul}(\cdot)$ are regularization terms on the coefficient $\mathbf{U}^{(N)}_t$, the factor $\mathbf{U}^{(n)}_t$, and the core tensor $\mathbf{G}_t$, respectively. These penalties can be nonnegativity, smoothness, or sparsity depending on the specific application.

The former ATD algorithm was proposed by Thanh et al. in [64] which is capable of tracking the low multilinear-rank approximation of streaming tensors from highly incomplete observations. In stage 1, ATD particularly recasts (43) into a standard LS optimization and then applies a randomized LS technique to minimize it. In stage 2, ATD introduces a recursive LS solver to optimize (44) in an efficient way. Instead of solving (45) directly, ATD applies the stochastic gradient descent to obtain its solution.

The two latter RI-NTD and BK-NTD algorithms were proposed by Zdunek et al. in [99] for factorizing nonnegative tensors from streaming data. Both algorithms perform nonnegative least-square (NNLS) solvers to incrementally update the tensor factors and the core tensor. Particularly, RI-NTD utilizes a recursive strategy involving the nonnegatively constrained Gauss–Seidel method while BK-NTD adopts the block Kaczmarz method. Similar to ATD, both RI-NTD and BK-NTD estimate the core tensor using only the new coming data via a stochastic optimization.

5.3 Multi-aspect Streaming Tucker Decomposition

Besides single-aspect Streaming Tucker decomposition methods, few online techniques are capable of tracking multi-aspect streaming tensors under the Tucker format over time, such as SIITA in [90] and eOTD in [91].

SIITA in [90] offers an online inductive framework for tracking the low-rank tensor approximation of multi-aspect streaming tensors as well as completing their missing data with side information. On the arrival of new data, SIITA particularly minimizes the following optimization

$$ \min_{\mathbf{G}_t, \{\mathbf{U}^{(n)}_t, \mathbf{A}^{(n)}_t\}_{n=1}^{N-1}} f_t(\mathbf{Y}_t, \{\mathbf{S}^{(n)}_t\}_{n=1}^{N-1}, \mathbf{G}_t, \{\mathbf{U}^{(n)}_t\}_{n=1}^{N-1}), $$

with

$$ f_t(\mathbf{Y}_t, \{\mathbf{S}^{(n)}_t\}_{n=1}^{N-1}, \mathbf{G}_t, \{\mathbf{U}^{(n)}_t\}_{n=1}^{N-1}) = \left\| \mathbf{P}_{\Omega_t} \mathbf{Y}_t - \mathbf{P}_{\Omega_t} \left( \left[ \mathbf{G}_t; \{\mathbf{S}^{(n)}_t\}_{n=1}^{N-1} \right] \right) \right\|_F^2 + \rho_{s^{(t)}} \left\| \mathbf{S}^{(t)}_t \right\|_F^2 + \sum_{n=1}^{N} \rho_{u^{(n)}} \left\| \mathbf{U}^{(n)}_t \right\|_F^2, \quad (46) $$

where $\{\mathbf{S}^{(n)}_t\}_{n=1}^{N-1}$ with $\mathbf{S}^{(n)}_t \in \mathbb{R}^{M_n \times I_n}$ is the set of side information matrices and $\rho_{s^{(t)}}, \rho_{u^{(n)}}$ are regularization parameters. Here, SIITA incorporates the side information into the data model by using $\{\mathbf{S}^{(n)}_t\}_{n=1}^{N-1}$ as multiplicative terms. Accordingly, SIITA can accelerate the tracking process because the product $\mathbf{S}^{(n)}_t \mathbf{U}^{(n)}_t$ transforms the dimensionality of variables from $I_n$ to $M_n$, typically with $M_n \ll I_n$. As every term of (46) are convex, SIITA adopts the gradient descent to minimize it. Besides, a simple variant of SIITA namely NN-SIITA was also obtained for nonnegative tensor decomposition. NN-SIITA is specifically derived...
from projecting the estimates of SIITA into their nonnegative orthant at each time $t$.

In [91], Xiao et al. proposed the so-called eOTD algorithm for the multi-aspect tensor tracking problem. Unlike SIITA, eOTD adopts the divide and conquer paradigm to deal with multi-aspect streaming tensors. In particular, it divides the underlying tensor $X_t$ into $2^N$ sub-tensors $X_{t(N)}^{(i_1, ..., i_N)}$ with $i_n \in \{0,1\}$, $1 \leq n \leq N$, and $X_{t(0,0,0,...,0)} = \mathcal{X}_t$, see Fig. 5 for an illustration. These sub-tensors are grouped into $N$ classes $\{D_n\}_{n=1}^N$ based on the sum of sub-indices. For example, for a third-order tensor, we have $D_1 = \{X_{t(1,0,0)}^{(1,0,0)}, X_{t(1,0,0)}^{(0,1,0)}, X_{t(0,0,0)}^{(0,0,0)}\}$, $D_2 = \{X_{t(1,1,0)}^{(1,1,0)}, X_{t(1,0,1)}^{(1,0,1)}, X_{t(0,1,1)}^{(0,1,1)}\}$, and $D_3 = \{X_{t(1,1,1)}^{(1,1,1)}\}$. If a sub-tensor $X_{t(i_1, ..., i_N)}^{(i_1, ..., i_N)} \in \mathcal{X}_t$, factorizing it will result in $X_{t(i_1, ..., i_N)}^{(i_1, ..., i_N)} = \mathcal{G}_t \{V_{t(n)}^{(1)}\}_{n=1}^N$ where $V_{t(n)}^{(1)} = U_{t(n)}^{(1)}$ if $i_n = 1$ and $V_{t(n)}^{(1)} = U_{t(n)}^{(0)}$ if $i_n = 0$. Here, the matrix $U_{t(n)}^{(1)}$ is constantly updated as follows

$$
U_{t(n)}^{(1)} = \tilde{U}_{t(n)}^{(1)} + (1 - \alpha) \sum_{(i_1, ..., i_N) \neq (0,0,0, ..., 0)} (\mathcal{G}_t^{(1)}(n \times 1)_{n=1}^N) \#.
$$

The factor $U_{t(n)}^{(1)}$ is derived from $U_{t(n)}^{(1)} = \text{orth} (\{U_{n-1}^{(1)}; U_{t(n)}^{(1)}\}) = (\{U_{t(n)}^{(1)}; U_{t(n)}^{(1)}\})$ where the modified Gram-Schmidt process was applied to compute the ortho operation. Finally, the tensor core $\mathcal{G}_t$ of fixed size is estimated by

$$
\mathcal{G}_t = \{\tilde{U}_{t(n)}^{(1)}\}^{N}_{n=1} + \sum_{(i_1, ..., i_N) \neq (0,0,0, ..., 0)} [X_{t(i_1, ..., i_N)}^{(i_1, ..., i_N)}; \{U_{t(n)}^{(1)}\}_{n=1}^N]^{N}_{n=1}.
$$

An appealing feature of eOTD is that throughout the tracking process, eOTD only uses cheap tensor-matrix multiplications and pseudo-inverse operations instead of computing the expensive SVDs on big matrices. This makes eOTD easy for applying to large-scale applications.

## 6 OTHER STREAMING TENSOR DECOMPOSITIONS

Apart from the two most popular streaming CP and Tucker decompositions, some methods are capable of tracking tensors under other multiway models. This section focuses on tracking algorithms that exploit TT, BTD, and t-SVD formats to construct the low-rank tensor approximation in the streaming setting.

### 6.1 Streaming Tensor-Train Decomposition

Despite success in the batch setting, TT decomposition has not gained in popularity as CP and Tucker for tensor tracking. In the literature, there exist few tracking algorithms developed for the problem of single-aspect tensor tracking under the TT format, see Fig. 6 for an illustration.

In [110]-[112], Thanh et al. proposed three adaptive TT algorithms called TT-FOA, ATT, and ROBOT for factorizing tensors in an online fashion. Particularly, TT-FOA in [110] is, to the best of our knowledge, the very first of its kind in the literature. However, its practical use is limited due to the lack of robustness to data corruption. To overcome the drawback, ATT in [111] and ROBOT in [112] were developed to deal with missing data and sparse outliers, respectively.

All three algorithms share the same optimization framework where BCD and RLS methods are utilized to minimize the cost function. In particular, a general formulation of the optimization problems can be written as

$$
\min_{\{\mathcal{G}(n)\}_{n=1}^N} \sum_{\tau=1}^{T} \beta^{t-\tau} \left( \| \mathcal{P}_{\tau} \circ \left( \mathcal{G}(1)_{1 \times 2 \times \cdots \times N-1} + \frac{1}{n} \| \mathcal{G}(n)_{1 \times 1 \times \cdots \times 1\times N-1} - \mathcal{G}(n-1)_{1 \times 1 \times \cdots \times 1\times N-1}\|_F^2 + \rho_1 \mathcal{R}_1(\mathcal{G}(\tau)) \right) \|_F^2 + \rho_2 \mathcal{R}_2(\mathcal{G}(\tau)) \right),
$$

where $\beta \in (0,1]$ is a forgetting factor to reduce the impact of old observations; $\rho_1 \mathcal{R}_1(\mathcal{G}(\tau))$ and $\rho_2 \mathcal{R}_2(\mathcal{G}(n)_{1 \times 1 \times \cdots \times 1\times N-1})$ are two regularization terms. Specifically, TT-FOA does not impose the two penalties; ATT adopts $\mathcal{R}_1(\mathcal{G}(n)_{1 \times 1 \times \cdots \times 1\times N-1}) = \sum_{n=1}^{N-1} \| \mathcal{G}(n)_{1 \times 1 \times \cdots \times 1\times N-1} - \mathcal{G}(n-1)_{1 \times 1 \times \cdots \times 1\times N-1}\|_F^2$ to control the smoothness of TT-cores over time; and ROBOT applies the $\ell_1$-norm regularization $\mathcal{R}_2(\mathcal{G}(\tau)) = \| \mathcal{G}(\tau) \|_1$ to promote the sparsity on $\mathcal{G}(\tau)$.

Thanks to the BCD framework, (49) can be effectively decomposed into two main stages: (i) estimate the temporal TT-core $\mathcal{G}(n)_{1 \times 1 \times \cdots \times 1\times N-1}$ and outlier $\mathcal{O}(\tau)$, and (ii) update non-temporal TT-cores $\mathcal{G}(n)_{1 \times 1 \times \cdots \times 1\times N-1}$. In stage 1, TT-FOA and ATT apply the regularized least-squares method to estimate $\mathcal{G}(n)_{1 \times 1 \times \cdots \times 1\times N-1}$ under the assumption that $\mathcal{O}(\tau)$ is outlier-free. Meanwhile ROBOT adopts an effective ADMM solver to account for the sparse outlier $\mathcal{O}(\tau)$. In stage 2, an effective RLS solver was introduced to estimate $\mathcal{G}(n)_{1 \times 1 \times \cdots \times 1\times N-1}$ when $\mathcal{G}(n)_{1 \times 1 \times \cdots \times 1\times N-1}$ and $\mathcal{O}(\tau)$ (if any) are given in stage 1.

In parallel, Liu et al. in [113] proposed an incremental TT method called iTTD to factorize tensors having one temporal mode. Specifically, iTTD considers incoming data streams as individual tensors and then factorizes them into TT-cores. The results are appended to old estimates derived from past observations. In [114], Wang et al. also developed an incremental TT method called AITT to decompose tensors from industrial IoT data streams. By exploiting a relationship between the directly reshaped matrix and integration of tensor unfolding matrices, AITT can estimate effectively the underlying TT-cores. However, the two frameworks of iTTD and AITT are not really online streaming learning ones but incremental batch learning. Therefore, they are not useful for data streams from dynamical observations in time-varying environments.

### 6.2 Streaming Block-Term Decomposition

The block-term decomposition (BTD) unifies the two well-known CP and Tucker decompositions, and thus, the tracking algorithms under the CP and Tucker formats principally belong to the class of the streaming BTD with one block. When the number of blocks is greater than 2, there are only two BTD methods able to deal with streaming tensors, including OnlineBTD [115] and O-BTD-RLS [116].

The former method was proposed by Gujral et al. in [115] for tensor tracking under the generalized BTD format of $L$ blocks and a multilinear rank-$(r_1, r_2, \ldots, r_N)$. On the arrival of the temporal slice $\mathcal{Y}_t$, OnlineBTD performs the following minimization:

$$
\min_{\{\mathcal{G}(n)\}_{n=1}^N} \sum_{\tau=1}^{T} \left( \| \mathcal{Y}_t - \sum_{l=1}^{L} \mathcal{G}(\tau)_{1 \times 2 \times \cdots \times N-1} \|_F^2 \right),
$$

where $\mathcal{Y}_t = [\mathcal{Y}^{(1)}_{(1)}; \mathcal{Y}^{(2)}_{(1)}; \mathcal{Y}^{(3)}_{(1)}; \ldots; \mathcal{Y}^{(n)}_{(1)}]$ with $\mathcal{Y}^{(n)}_{(1)} \in \mathbb{R}^{I_{1,n} \times r_n}$ and $\mathcal{G}(\tau)_{1 \times 2 \times \cdots \times N-1} = \mathcal{G}(\tau)_{1 \times 2 \times \cdots \times N-1}$. Here, $\{\mathcal{Y}^{(n)}_{(1)}\}_{n=1}^{N-1}$ are supposed to remain unchanged with time except the last tensor factor $\mathcal{Y}^{(N)}_{(1)}$. Prior information of $r$ and rank-$(r_1, r_2, \ldots, r_N)$ are known in advance. Old estimates of the core tensors and tensor factors of
\[ \mathbf{X}_{t-1} \approx U_1^{(1)} \cdot U_2^{(2)} \cdot \cdots \cdot U_n^{(n)} \]

Fig. 7. Tracking the rank-\((r, r, 1)\) BTD of the 3-rd order streaming \(\mathbf{X}_t\).

\( \mathbf{X}_{t-1} \) are used as a “warm start” for OnlineBTD at each time \( t \). To speed up the tracking process, OnlineBTD utilizes (i) an accelerated matricized tensor times Kronecker product, (ii) the pseudo-inverse operator using LU decomposition, and (iii) a dynamic programming strategy introduced by Zhou et al. in [49] to avoid the re-computation of duplicated Kronecker products.

The second method was introduced by Rontogiannis et al. [116]. Specifically, T-BTD-RLS is designed for tracking the low rank-\((r, r, 1)\) terms of three-order streaming tensors (i.e., \( r_1 = r_2 = r \) and \( r_3 = 1 \)), see Fig. 7 for an illustration. In particular, the tensor factors of the underlying tensor are incrementally updated by minimizing the following objective function:

\[
\begin{align*}
\min_{\{U^{(n)}\}_{n=1}^N} & \sum_{\tau=1}^{t} \sum_{\tau=1}^{t} \beta^{\tau-t} \left\| Y_{\tau} - U^{(1)} U^{(2)} \right\|_F^2 + \rho_1 \sum_{l=1}^{n} \sum_{k=1}^{L} \sqrt{\| U_l^{(k)} \|_2^2 + \| U_{l+1}^{(k)} \|_2^2 + \eta^2} + \rho_2 \sqrt{\| \Xi U_l \|_2^2 + \eta^2},
\end{align*}
\]

Here, \( U^{(n)} = [U_1^{(n)}, U_2^{(n)}, \ldots, U_{L}^{(n)}] \) with \( U_l^{(n)} \in \mathbb{R}^{r \times r} \) is the \( l \)-th order streaming tensor of dimensionality, width \( \times \) height \( \times \) channel \( \times \) time. Accordingly, there are several studies devoted to developing tensor-based visual trackers for better modeling the appearance of target objects, such as [80], [121]–[123], to name a few. For example, Hu et al. [80] proposed the so-called IRTSA tracker using incremental tensor subspace learning to capture the appearance of objects. Zhang et al. [121] introduced another visual tracker called DTAMU which stands for dynamic tensor analysis with mean and update. Weiming et al. in [122] developed a semi-supervised tensor-based visual tracker using graph embedding. Khan et al. in [123] built an online spatio-temporal tensor learning model for visual tracking using Bayesian inference. It is worth noting that most of the existing tensor-based visual trackers correspond to the streaming Tucker decomposition and its variants.

Another notable application of tensor tracking in computer vision is video background and foreground separation which is quite related to visual tracking, but with a different aim of modeling the scene background and detecting the information of changes in the scene. Similar to visual tracking, many tensor-based separators were proposed, such as [65], [105], [112], [124], [125]. Particularly in [65], Thanh et al. proposed a robust adaptive CP method called RACP which is capable of modeling video background and detecting moving objects. Li et al. in [105] introduced an online robust low-rank tensor modeling (ORLTM) method and found its success in video background subtraction. Andrews et al. [124] developed an online stochastic tensor decomposition for background subtraction in multispectral video sequences. A robust streaming tensor-train algorithm was developed in [112] which also has the potential to detect foreground in video. Salut et al. [125] proposed an online tensor robust principal component analysis and validated its effectiveness with the problem of background and foreground separation.
In parallel, there are other interesting computer vision applications of dynamic tensor analysis, such as visual data recovery [56], [126], online video denoising [127], [128], and segmentation/classification [94], [129].

7.2 Neuroscience

The brain can be viewed as a complex system with various interacting regions that can produce large multivariate data over time [130]. Many types of brain data can be represented by tensors, such as electroencephalography (EEG), magnetoencephalography (MEG), functional magnetic resonance imaging (fMRI), and near-infrared spectroscopy (NIRS) [131]. Apart from three intrinsic modes (i.e., frequency, channel, and time), brain data can have higher-order modes, such as, subjects, conditions, and trials [131]. Together with the fact that brain activities can change over time, dynamic tensor analysis has become an useful tool to study the structure and function of brain from such data.

In what follows, we list some appealing brain-computer interface applications to demonstrate the use of dynamic tensor analysis in neuroscience. First, for the problem of detecting dynamic functional connectivity networks (DFCNs), Ozdemir et al. in [87] introduced a recursive tensor-based framework capable of tracking DFCNs over time. The proposed framework was then applied for studying error-related negativity – a brain potential response when patients make errors during cognitive tasks [132]. Mahyari et al. in [133] developed a two-step approach using incremental tensor subspace analysis for detecting DFCNs. Particularly, they first detect change points at which the functional connectivity across subjects presents abrupt changes and then summarize DFCNs between successive change points. Recently, Acar et al. in [134] proposed to use the Parafac2 model for tracking the evolution of connectivity networks and compared its performance with ICA and IVA. For the problem of localizing dynamic brain sources over time, Ardeshir et al. in [135] utilized the boundary element method (BEM) [136] and the adaptive PARAFAC-RLIST tracker [46] with two operational windowing schemes. A variant using augmented complex statistics in [137] also has the ability to track moving EEG sources with time. For the problem of online EEG completion, Trung et al. in [138] proposed an adaptive CP algorithm called NL-PETRELS capable of tracking and imputing incomplete EEG data. Than et al. in [64], [65] also demonstrated the use of ACP and RACP with real data by applying them for online EEG completion. Other neuroscience applications of tensor analysis were reviewed in [9], [33], [35].

7.3 Anomaly Detection

Anomaly detection, which corresponds to identifying patterns and data points that do not conform to normal behavior, plays an essential role in many applications, such as cyber security, statistics, and finance, to name a few [139]. Here, we provide some notable tensor-based anomaly detectors which are customized to specific online applications.

Shi et al. in [140] developed the so-called STenSr algorithm for anomaly detection and pattern discovery in spatio-temporal tensor streams from sensor networks. STenSr utilizes an incremental HOSVD and a metric based on Euclidean distance to detect abrupt changes when new data comes. Kasai et al. in [141] introduced an online time-structured traffic tensor tracking framework to detect network-level anomalies from link indirect measurements over time. In particular, it is based on a robust adaptive CP decomposition that uses RLS for tensor tracking and ADMM for detecting abnormal flows. Cao et al. in [142] designed an interactive system called Voila for detecting and monitoring visual anomalies. Voila is a tensor-based anomaly detector with an interaction design that can ranks anomalous patterns based on user input. Lin et al. in [143] proposed a novel method called TBAD to localize anomalous events. TBAD employs a spatial-feature-temporal tensor model and analyses latent patterns through unsupervised learning. Xu et al. in [144] introduced a tensor-based framework, namely SWTF, capable of detecting multiple types of anomalies in road networks. We refer the readers to [34] for a broader interdisciplinary survey of tensors for anomaly detection.

7.4 Others

Apart from online applications in the domains above, tensor tracking also found success in some other research fields, namely wireless communications (e.g., channel tracking [145], DOA tracking [146], and time delay estimation [147]), network analysis (e.g., link prediction [12], internet scale monitoring [148], and bot activities and network intrusions [149]), data analytics of chemical and biological manufacturing processes and components [150], [151], performance monitoring [152], [153], and transportation [154], [155].

8 Research Challenges, Open Problems, and Future Directions

In this section, we first discuss some advantages and disadvantages of the state-of-the-art tensor tracking models. Then, we present several research challenges and open problems that should be considered for the development of tensor tracking in the future. They are data imperfection and corruption; rank revealing and tracking; efficient and scalable tensor tracking; and other aspects such as theoretical analysis, symbolic data, and tracking under some less common tensor formats. Possible solutions for these challenges are also discussed.

8.1 Discussions on State-of-the-art Tensor Tracking Models

As reviewed in previous sections, most of the state-of-the-art tensor tracking methods were developed for streaming CP and Tucker decompositions. Indeed, the CP tracking model can be seen as a special case of streaming Tucker decomposition when the core tensor is constrained to be an identity tensor. Therefore, the complexity (w.r.t. computation and memory storage) of streaming CP decomposition is generally much lower than that of Tucker. In other words, CP trackers are often faster than Tucker ones in practice. Another interesting feature of the CP tracking model over Tucker’s online variants is the uniqueness in which tensor factors of CP are unique up to a permutation and scale under certain conditions, similar to batch CP decomposition. This feature is a useful property in several applications, e.g., for source separation or to recover exact components and individuals hidden in the underlying data. However, CP is not really a practical model because finding the true CP rank of streaming tensors is nontrivial even when we have enough resources to process multiple snapshots at a time. The problem of tensor rank tracking will be discussed later in subsection 8.3.

Streaming Tucker decomposition is, however, more flexible than CP. It stems from the fact that we here only need to track the column spaces of tensor factors over time, instead of a particular basis. In addition, the Tucker rank determination may be “easier” than that of CP as we can take advances of rank estimation in the problem of subspace tracking (e.g., [156], [157]). However, when we deal with large-scale and high-dimensional streaming tensors, the size of the core tensor can be large which makes Tucker trackers less efficient for stream processing. The BTD tracking model lies between CP and Tucker in the sense that
the core tensor is block diagonal. Thus, it has the advantages of both CP and Tucker tracking models. Despite having that, tracking the underlying BTD approximation of streaming tensors may be harder than CP and Tucker as it takes several types of rank into account, i.e., the number of blocks and their size. The tensor-train tracking model shares the same drawback with BTD as it must estimate several TT-cores and their rank. An appealing advantage of streaming TT decomposition is that it can deal with tensors of a very high order, thanks to its memory-saving representation which is linear to the order of tensor [5]. The t-SVD tracking model has its own advantage and disadvantage. As its algebraic framework is in the Fourier domain, t-SVD can utilize fast operations (e.g., FFT and its inverse) to speed up the tracking process. Generalizing the t-SVD to tensors of order greater than three can be done via recursion [158] or hot-HOSVD [159] which are quite expensive. It becomes a bottleneck that can make tracking higher-order streaming tensors under the t-SVD model inefficient.

8.2 Data Imperfection and Corruption

Dealing with data imperfection and corruption has been a critical issue in many applications and tracking problems in particular [160]. We here present two main types of imperfect data that either remain unsolved or are still challenging for tensor tracking: (i) non-Gaussian and colored noises; (ii) outliers and missing data.

a) Non-Gaussian and Colored Noises. Most of the existing tensor tracking algorithms were proposed under the additive white Gaussian noise assumption. This assumption however does not always hold in practice. For example, impulsive noises (e.g., burst, alpha-stable, and spherically invariant random variable noise), which are introduced by human activities and natural sources, are one of the most common non-Gaussian noises that often appear in tracking applications such as direction of arrivals [161], OFDM systems [162] and adaptive system identification [163]. This type of noise can significantly impact the tracking ability of estimators and it requires specific treatments [164]. In parallel, colored noises that indicate types of noise that are correlated in space and/or time may reduce the performance of tracking algorithms [165]. Accordingly, standard tracking algorithms may be less effective in estimation accuracy in the presence of these noises. They need to be readapted or redesigned for more robustness.

To the best of our knowledge, we are not aware of any tensor tracking algorithm capable of handling such noises in the literature. Some potential approaches have been successfully demonstrated in subspace tracking problems (i.e., tracking tensors of order 2), see [164] for a brief survey. In particular, adaptive Kalman filtering and weighted RLS approaches can be adopted for dealing with impulsive noises. Oblique projection and instrumental variable-based techniques can handle colored noises. Therefore, it is desirable to extend these approaches from subspace tracking to tensor tracking.

b) Outliers and Missing Data. They are now becoming more and more ubiquitous in modern datasets. Outliers are data points that appear to be inconsistent with or exhibit abnormal behaviour different from others. Missing observations are often encountered during the data acquisition and collection. Both outliers and missing data can cause several issues (e.g., they introduce bias in estimation) for knowledge discovery from data in general and data streams in particular [166]. Accordingly, dealing with them is an essential task in the analysis of corrupted datasets which has been still a hot topic in data mining for decades. In general, handling such corruptions involves removing/ignoring them after detection or replacing them with alternative values.

There exist few tensor tracking algorithms robust to sparse outliers in the literature. Under the CP format, SOFIA [61] applies the robust Holt-Winters forecasting model using a pre-cleaning mechanism to identify and down-weight outliers. RACP [65] introduces a $\ell_1$-norm penalty to promote the sparsity on outliers and then uses an ADMM solver to estimate them. Under the Tucker format, ORLTM [105], OLRTR [106], and D-L1-Tucker [96] are able to deal with sparse outliers. Both ORLTM and OLRTR propose to regularize the main objective function with a $\ell_1$-norm regularization. Meanwhile, D-L1-Tucker adopts a threshold-based method to detect outliers. Except for RACP, most of the mentioned algorithms above are not designed for dealing with missing data. In parallel, most of the existing online tensor completion and tracking are sensitive to outliers, such as TeCP-SGD [48], OLSTEC [56], and ACP [64]. Accordingly, there are plenty of opportunities to develop robust tensor tracking from incomplete observations as it is still in its early stage.

8.3 Rank Revealing and Tracking

Most of the state-of-the-art tensor tracking algorithms suppose that the tensor rank (e.g., CP, Tucker, BTD, TT, or tubal rank) is given as prior information. In practice, it is however a difficult assumption due to the facts that: (i) the tensor rank may change over time and (ii) a good rank determination at the initialization stage is not always guaranteed when the number of training samples is limited and (iii) the exact rank determination may be intractable (e.g., CP rank is NP-hard [27]). Therefore, it is essential to develop tracking algorithms that are capable of revealing the rank over time.

In the literature, there have been many heuristic methods developed for the problem of tensor rank estimation. Most of them adopt the Bayesian approach to infer the tensor rank from data, such as [167]–[169]. Theoretically, Bayesian inference offers a good recipe for the tensor rank estimation as we can integrate the low-rank promoting prior as well as the tensor rank into the learning framework. Another possible approach to determine the tensor rank is to use neural networks (NNs), such as [170]–[172]. Since the rank can be considered as one type of data feature, NNs which can extract hidden features within data can be used to solve the tensor rank determination. Although these methods often require the tensor data to be fully observed, it is possible to readapt or modify them such that their variant are able to handle tensors in an online fashion. For example, we can adopt online Bayesian inference or online learning algorithms for training NNs. Finally, robustness against rank overestimation errors is an issue that, to the best of our knowledge, has not been considered yet in practice.

8.4 Efficient and Scalable Tensor Tracking

Tabs. 3 and 5 indicate that most of the existing tensor tracking algorithms are of high complexity. When we deal with large-scale and high-multidimensional streams, they may become less efficient. Thus, it is necessary to develop efficient and scalable tracking techniques of low cost w.r.t. both computational complexity and memory storage. In what follows, we present three potential approaches which are theoretically capable of accelerating the tracking process, namely (a) randomized sketching, (b) parallel and distributed computing, and (c) neural networks-based methods.

a) Randomized Sketching. It is very well-known that randomized methods can reduce the computational cost of their counterparts while still achieving reasonable estimation [173].
Accordingly, many attempts have been made to take their advantages in computation for tensor decomposition in the literature, we refer the readers to [23] for a good overview. Among them, there are a few online algorithms utilizing successfully randomized techniques to speed up the tracking process, such as [57], [63], [64], [174]. Particularly, these algorithms involve solving several overdetermined least-squares (LS) problems. Thanks to the CP and Tucker structures, they use random sampling to build the sampled Khatri-Rao and Kronecker products, and then, recast the original LS problems into randomized ones. Solving the new LS problems can save a lot of computational complexity. Other randomized techniques (e.g., random projections and count sketch) with other tensor formats have not yet been investigated for tensor tracking and they deserve next investigations in the future.

b) Parallel and Distributed Computing. The second approach is to develop parallel and distributed computing frameworks for streaming tensor decomposition. It stems from the fact that we can leverage several computational resources to facilitate the tracking process. Moreover, computing systems in a parallel and distributed environment can offer more reliability than their counterparts in a central one as they can avoid the single point of failure which is a fundamental mistake from flaws in the implementation or design of a system. Besides, another appealing advantage of this computing is the scaling up-and-out process in which we can add and/or replace computational resources to the system. We refer the readers to [175] for a good reference.

In the tensor literature, there are several parallel and distributed systems for processing large-scale tensors. We can list here some efficient tools for: (a) distributed CP decomposition (e.g., DFacTo [176], SPLATT [177]), (b) distributed Tucker decomposition (e.g., DHOSVD [88], SGD-Tucker [178]), and (c) distributed TT decomposition (e.g., ADIT [114], ATTAC [179]), etc. These tools mainly distribute the unfolding matrices or sub-tensors among several clusters and integrate their low-rank tensor approximations to find the overall low-rank approximation of the underlying tensor. However, most of the existing distributed tensor decompositions are not suitable for handling streaming data. Therefore, it is of great interest to develop practical distributed systems for tracking tensors from data streams.

c) Neural Networks-based Methods. Another potential approach is to incorporate neural networks (NNs) into tensor factorization to benefit from their significant advances in computational power. On the one hand, the connection between TDs and NNs has been established in some studies, such as [180], [181]. For example, Cohen et al. in [180] showed that the convolutional NNs with ReLU activation and max/average pooling can be represented by tensor decomposition models. Wang et al. in [181] introduced two NN models for finding the low-tubal-rank approximation of three-order tensors. Accordingly, NN tools can be used to model and learn high-order interactions for tensors, and hence, for tensor factorization and tracking. On the other hand, NNs can directly map data streams (temporal slices) as input to the approximation result as output by applying some online learning techniques. In the literature of machine learning, there exist several kinds of learning capable of dealing with data streams, such as incremental learning, lifelong learning, and online continual learning, to name a few. They can be specifically adapted for tensor tracking.

8.5 Others

Next, we present some other issues and problems which also deserve future investigations.

9 Conclusions

Tensor tracking has recently gained increasing attention as a powerful tool for multidimensional data stream analysis. In this survey, we have provided a technical overview of online techniques for tracking streaming tensors over time. We highlighted the two most popular streaming CP and Tucker decompositions. Specifically, four main groups of streaming CP decomposition algorithms were emphasized, which are subspace-based, block-coordinate descent, Bayesian inference, and multi-aspect streaming decompositions. We categorized the current streaming Tucker decomposition methods into three major classes based on their model architecture. They are online tensor dictionary learning, tensor subspace tracking, and multi-aspect streaming decompositions. Recent years have also witnessed significant advances in other types of tensor decomposition such as tensor-train, BTD, and t-SVD. A brief survey on the existing methods which are capable of tracking tensors under these formats was presented. Finally, we discussed several research challenges, open problems, and future directions for tensor tracking.