CTD: Fast, accurate, and interpretable method for static and dynamic tensor decompositions

Jungwoo Lee[1], Dongjin Choi[1], Lee Sael[2,3,*]

1 Department of Computer Science and Engineering, Seoul National University, Seoul, Republic of Korea
2 Department of Computer Science, SUNY Korea, Incheon, Republic of Korea
3 Department of Computer Science, Stony Brook University, New York, USA

* sael@cs.stonybrook.edu

Abstract

How can we find patterns and anomalies in a tensor, i.e., multi-dimensional array, in an efficient and directly interpretable way? How can we do this in an online environment, where a new tensor arrives at each time step? Finding patterns and anomalies in multi-dimensional data have many important applications, including building safety monitoring, health monitoring, cyber security, terrorist detection, and fake user detection in social networks. Standard tensor decomposition results are not directly interpretable and few methods that propose to increase interpretability need to be made faster, more memory efficient, and more accurate for large and quickly generated data in the online environment. We propose two versions of a fast, accurate, and directly interpretable tensor decomposition method we call CTD that is based on efficient sampling method. First is the static version of CTD, i.e., CTD-S, that provably guarantees up to 11× higher accuracy than that of the state-of-the-art method. Also, CTD-S is made up to 2.3× faster and up to 24× more memory-efficient than the state-of-the-art method by removing redundancy. Second is the dynamic version of CTD, i.e. CTD-D, which is the first interpretable dynamic tensor decomposition method ever proposed. It is also made up to 82× faster than the already fast CTD-S by exploiting factors at previous time step and by reordering operations. With CTD, we demonstrate how the results can be effectively interpreted in online distributed denial of service (DDoS) attack detection and online troll detection.

Introduction

Given a tensor, or multi-dimensional array, how can we find patterns and anomalies in an efficient and directly interpretable way? How can we do this in an online environment, where new data arrive at each time step? Many real-world data are multi-dimensional and can be modeled as sparse tensors. Examples include network traffic data (source IP - destination IP - time), movie rating data (user - movie - time), IoT sensor data, and healthcare data. Finding patterns and anomalies in those tensor data is a very important problem with many applications such as building safety monitoring [1], patient health monitoring [2–5], cyber security [6], terrorist detection [7–9], and fake user detection in social networks [10,11]. Tensor decomposition method, a widely-used tool in tensor analysis, has been used for this task. However, the standard tensor decomposition methods such as PARAFAC [12] and Tucker [13] do not provide interpretability and are not applicable for real-time analysis in environments with high-velocity data.

Sampling-based tensor decomposition methods [14–16] arose as an alternative due to their direct interpretability. The direct interpretability not only reduces time and effort involved in finding patterns and anomalies from the decomposed tensors but also provides clarity in interpreting the result. A sampling-based decomposition method for sparse tensors is also memory-efficient since it preserves the sparsity of the original tensors on the sampled factor matrices. However, existing sampling-based tensor decomposition methods are slow, have high memory usage, and produce low accuracy. For example, tensor-CUR [16], the state-of-the-art sampling-based static tensor decomposition method, has many redundant fibers including duplicates in its factors. These redundancy cause higher memory usage and longer running time. Tensor-CUR is also not accurate enough for real-world tensor analysis.

In addition to interpretability, demands for online method applicable in a dynamic environment, where multi-dimensional data are generated continuously at a fast rate, are also increasing. A real-time analysis is not feasible with static methods since all the data, i.e., historical and incoming tensors, need to be decomposed over again at each time step. There are a few dynamic tensor decomposition methods proposed [17–19]. However, proposed methods are not directly interpretable and do not preserve sparsity. To the best of our knowledge, there has been no sampling-based dynamic tensor decomposition method proposed.

In this paper, we propose CTD (Compact Tensor Decomposition), a fast, accurate, and interpretable sampling-based tensor decomposition method. CTD has two versions: CTD-S for static tensors, and CTD-D for dynamic tensors. CTD-S is optimal after sampling, and results in a compact tensor decomposition through careful sampling and redundancy elimination, thereby providing much better running time and memory efficiency than previous methods. CTD-D, the first sampling-based dynamic tensor decomposition method in literature, updates and modifies minimally on the components altered by the incoming data, making the method applicable for real-time analysis on a dynamic environment. Table 1 shows the comparison of CTD and the existing method, tensor-CUR.

Table 1. Comparison of our proposed CTD and the existing tensor-CUR. The static method CTD-S outperforms the state of-the-art tensor-CUR in terms of time, memory usage, and accuracy. The dynamic method CTD-D is the fastest.

	Existing	[Proposed]	
	Tensor-CUR [16]	**CTD-S**	**CTD-D**
Interpretability	✓	✓	✓
Time	fast	faster	**fastest**
Memory usage	low	**lower**	low
Accuracy	low	**high**	**high**
Online			✓

Our main contributions are as follows:

- **Method.** We propose CTD, a fast, accurate, and directly interpretable tensor decomposition method. We prove the optimality of the static method CTD-S which makes it more accurate than the state-of-the-art method. Also, to the best of our knowledge, the dynamic method CTD-D is the first sampling-based dynamic tensor decomposition method.

- **Performance.** CTD-S is up to 11× more accurate, 2.3× faster, and 24× more memory-efficient compared to tensor-CUR, the state-of-the-art competitor as shown in Fig 1. CTD-D is up to 82× faster than CTD-S as shown in Fig 2.
- **Interpretable Analysis.** We show how CTD results are directly interpreted to successfully detect DDoS attacks in network traffic data and trolls in social network data.

Fig 1. Error, running time, and memory usage of CTD-S compared to those of tensor-CUR. CTD-S is more accurate, faster and more memory-efficient than tensor-CUR.

Fig 2. Error, running time, and memory usage relation of CTD-D compared to those of CTD-S. CTD-D is faster and has smaller error while using the same or slightly larger memory space compared to CTD-S.

The codes and datasets used in this paper are available at https://github.com/leesael/CTD. The rest of this paper is organized as follows. We first describe preliminaries and related works for tensor and sampling-based decomposition. We then describe our proposed method CTD and the experimental results. After presenting CTD at work, we conclude this paper.

Preliminaries and Related Works

In this section, we describe preliminaries and related works for tensor and sampling-based decompositions. Table 2 lists the definitions of symbols used in this paper.

Table 2. Table of symbols.

Symbol	Definition	Symbol	Definition
$\boldsymbol{\mathcal{X}}$	tensor (Euler script, bold letter)	$\mathbf{X}^{\dagger}$	pseudoinverse of $\mathbf{X}$
$\mathbf{X}$	matrix (uppercase, bold letter)	N	order of a tensor
$\mathbf{x}$	column vector (lower case, bold letter)	$\times_n$	n-mode product
x	scalar (lower case, italic letter)	$\lVert \bullet \rVert_F$	Frobenius norm
$\mathbf{X}_{(n)}$	mode-n matricization of a tensor $\boldsymbol{\mathcal{X}}$	$nnz(\boldsymbol{\mathcal{X}})$	number of nonzero elements in $\boldsymbol{\mathcal{X}}$

Tensor

A tensor is a multi-dimensional array and is denoted by the boldface Euler script, e.g. $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ where N denotes the order (the number of axes) of $\boldsymbol{\mathcal{X}}$. Each axis of a tensor is also known as mode or way. A fiber is a vector (1-mode tensor) which has fixed indices except one. Every index of a mode-n fiber is fixed except n-th index. A fiber can be regarded as a higher-order version of a matrix row and column. A matrix column and row each correspond to mode-1 fiber and mode-2 fiber, respectively. A slab is an $(N-1)$-mode tensor which has only one fixed index. $\mathbf{X}_{(\alpha)} \in \mathbb{R}^{I_\alpha \times N_\alpha}$ denotes a mode-α matricization of $\boldsymbol{\mathcal{X}}$, where $N_\alpha = \prod_{n \neq \alpha} I_n$. $\mathbf{X}_{(\alpha)}$ is made by rearranging mode-α fibers of $\boldsymbol{\mathcal{X}}$ to be the columns of $\mathbf{X}_{(\alpha)}$. $\lVert \boldsymbol{\mathcal{X}} \rVert_F$ is the Frobenius norm of $\boldsymbol{\mathcal{X}}$ and is defined by Equation 1.

$$\|\mathcal{X}\|_F^2 = \sum_{i_1, i_2, \cdots, i_N} x_{i_1 i_2 \cdots i_N}^2 \tag{1}$$

$\mathcal{X} \times_n \mathbf{U} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$ denotes the n-mode product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ with a matrix $\mathbf{U} \in \mathbb{R}^{J \times I_n}$. Elementwise,

$$(\mathcal{X} \times_n \mathbf{U})_{i_1 \cdots i_{n-1} j i_{n+1} \cdots i_N} = \sum_{i_n = 1}^{I_n} x_{i_1 \cdots i_{n-1} i_n i_{n+1} \cdots i_N} u_{j i_n} \tag{2}$$

$\mathcal{X} \times_n \mathbf{U}$ has a property shown in Equation 3.

$$\mathcal{Y} = \mathcal{X} \times_n \mathbf{U} \Leftrightarrow \mathbf{Y}_{(n)} = \mathbf{U}\mathbf{X}_{(n)} \tag{3}$$

We assume that a matrix or tensor is stored in a sparse-unordered representation (i.e. only nonzero entries are stored in a form of pair of indices and the corresponding value). $nnz(\mathcal{X})$ denotes the number of nonzero elements in $\mathcal{X}$.

We describe existing sampling-based matrix and tensor decomposition methods in the following subsections.

Sampling Based Matrix Decomposition

Sampling-based matrix decomposition methods sample columns or rows from a given matrix and use them to make their factors. They produce directly interpretable factors which preserve sparsity since those factors directly reflect the sparsity of the original data. In contrast, a singular value decomposition (SVD) generates factors which are hard to understand and dense because the factors are in a form of linear combination of columns or rows from the given matrix. Definition 1 shows the definition for CX matrix decomposition [20], a kind of sampling-based matrix decomposition.

Definition 1. *Given a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, the matrix $\tilde{\mathbf{A}} = \mathbf{CX}$ is a CX matrix decomposition of $\mathbf{A}$, where a matrix $\mathbf{C} \in \mathbb{R}^{m \times c}$ consists of actual columns of $\mathbf{A}$ and a matrix $\mathbf{X}$ is any matrix of size $c \times n$.*

We introduce well-known CX matrix decomposition methods: LinearTimeCUR, CMD, and Colibri.

LinearTimeCUR and CMD. Drineas et al. [21] proposed LinearTimeCUR and Sun et al. [22] proposed CMD. In the initial step, LinearTimeCUR and CMD sample columns from an original matrix $\mathbf{A}$ according to the probabilities proportional to the norm of each column with replacement. Drineas et al. [21] has proven that this biased sampling provides an optimal approximation. Then, they project $\mathbf{A}$ into the column space spanned by those sampled columns and use the projection as the low-rank approximation of $\mathbf{A}$. LinearTimeCUR has many duplicates in its factors because a column or row with a higher norm is likely to be selected multiple times. These duplicates make LinearTimeCUR slow and require a large amount of memory. CMD handles the duplication issue by removing duplicate columns and rows in the factors of LinearTimeCUR, thereby reducing running time and memory significantly.

Colibri. Tong et al. [23] proposed Colibri-S which improves CMD by removing all types of linear dependencies including duplicates. Colibri-S is much faster and memory-efficient compared to LinearTimeCUR and CMD because the dimension of factors is much smaller than that of LinearTimeCUR and CMD. Tong et al. [23] also proposed the dynamic version Colibri-D. Although Colibri-D can update its factors incrementally, it fixes the indices of the initially sampled columns which need to be updated over time. Our CTD-D not only handles general dynamic tensors but also does not have to fix those indices.

Sampling Based Tensor Decomposition

Sampling-based tensor decomposition method samples actual fibers or slabs from an original tensor. In contrast to PARAFAC and Tucker, the most famous tensor decomposition methods, the resulting factors of sampling-based tensor decomposition method are easy to understand and usually sparse. There are two types of sampling based tensor decomposition: one based on Tucker and the other based on LR tensor decomposition which is defined in Definition 2.2. In Tucker-type sampling based tensor decomposition (e.g., ApproxTensorSVD [14] and FBTD (fiber-based tensor decomposition) [15]), factor matrices for all modes are either sampled or generated; the overhead of generating a factor matrix for each mode makes these methods too slow for applications to real-time analysis. We focus on sampling methods based on LR tensor decomposition which is faster than those based on Tucker decomposition.

Definition 2. *(LR tensor decomposition) Given a tensor* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$*,* $\tilde{\boldsymbol{\mathcal{X}}} = \boldsymbol{\mathcal{L}} \times_\alpha \mathbf{R}$ *is a mode-*α *LR tensor decomposition of* $\boldsymbol{\mathcal{X}}$*, where a matrix* $\mathbf{R} \in \mathbb{R}^{I_\alpha \times c}$ *consists of actual mode-*α *fibers of* $\boldsymbol{\mathcal{X}}$ *and a tensor* $\boldsymbol{\mathcal{L}}$ *is any tensor of size* $I_1 \times \cdots \times I_{\alpha-1} \times c \times I_{\alpha+1} \times \cdots \times I_N$.

Tensor-CUR. Mahoney et al. [16] proposed tensor-CUR, a mode-α LR tensor decomposition method. Tensor-CUR is an n-dimensional extension of LinearTimeCUR. Tensor-CUR samples fibers and slabs from an original tensor and builds its factors using the sampled ones. The only difference between LinearTimeCUR and tensor-CUR is that tensor-CUR exploits fibers and slabs instead of columns and rows. Thus, tensor-CUR has drawbacks similar to those of LinearTimeCUR. Tensor-CUR has many redundant fibers in its factors and these fibers make tensor-CUR slow and use a large amount of memory.

Proposed Method

In this section, we describe our proposed CTD (Compact Tensor Decomposition), an efficient and interpretable sampling-based tensor decomposition method. We first describe the static version CTD-S, and then the dynamic version CTD-D of CTD.

CTD-S for Static Tensors

Overview. How can we design an efficient sampling-based static tensor decomposition method? Tensor-CUR, the existing state-of-the-art, has many redundant fibers in its factors and these fibers make tensor-CUR slow and use large memory. Our proposed CTD-S method removes all dependencies from the sampled fibers and maintains only independent fibers; thus, CTD-S is faster and more memory-efficient than tensor-CUR.

Algorithm. Fig 3 shows the scheme for CTD-S. CTD-S first samples fibers biased toward a norm of each fiber. Three different fibers (red, blue, green) are sampled in Fig 3. There are many duplicates after biased sampling process since CTD-S samples fibers multiple times with replacement and a fiber with a higher norm is likely to be sampled many times. There also exist linearly dependent fibers such as the green fiber which can be expressed as a linear combination of the red one and the blue one. Those linearly dependent fibers including duplicates are redundant in that they do not give new information when interpreting the result. CTD-S removes those redundant fibers and stores only the independent fibers in its factor $\mathbf{R}$ to keep result compact. CTD-S only keeps one red fiber and one blue fiber in $\mathbf{R}$ in Fig 3.

Fig 3. The scheme for CTD-S.

CTD-S decomposes a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ into one tensor $\mathcal{C} \in \mathbb{R}^{I_1 \times \cdots \times I_{\alpha-1} \times \tilde{s} \times I_{\alpha+1} \times \cdots \times I_N}$, and two matrices $\mathbf{U} \in \mathbb{R}^{\tilde{s} \times \tilde{s}}$ and $\mathbf{R} \in \mathbb{R}^{I_\alpha \times \tilde{s}}$ such that $\mathcal{X} \approx \mathcal{C} \times_\alpha \mathbf{RU}$. CTD-S is a mode-$\alpha$ LR tensor decomposition method and is interpretable since $\mathbf{R}$ consists of independent fibers sampled from $\mathcal{X}$.

Algorithm 1 CTD-S for Static Tensor

```
Input: Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, mode $\alpha \in \{1, \cdots, N\}$, sample size $s \in \{1, \cdots, N_\alpha\}$, and
    tolerance $\epsilon$
Output: $\mathcal{C} \in \mathbb{R}^{I_1 \times \cdots \times I_{\alpha-1} \times \tilde{s} \times I_{\alpha+1} \times \cdots \times I_N}$, $\mathbf{U} \in \mathbb{R}^{\tilde{s} \times \tilde{s}}$, $\mathbf{R} \in \mathbb{R}^{I_\alpha \times \tilde{s}}$
 1: Let $\mathbf{X}_{(\alpha)}$ be the mode-$\alpha$ matricization of $\mathcal{X}$
 2: Compute column distribution for $i = 1, \cdots, N_\alpha$:
    $P(i) \leftarrow \frac{|\mathbf{X}_{(\alpha)}(:,i)|^2}{\|\mathbf{X}_{(\alpha)}\|_F^2}$
 3: Sample $s$ columns from $\mathbf{X}_{(\alpha)}$ based on $P(i)$. Let $I = \{i_1, \cdots, i_s\}$
 4: Let $I' = \{i'_1, \cdots, i'_{s'}\}$ be a set consisting of unique elements in $I$
 5: Initialize $\mathbf{R} \leftarrow [\mathbf{X}_{(\alpha)}(:, i'_1)]$ and $\mathbf{U} \leftarrow 1/(\mathbf{X}_{(\alpha)}(:, i'_1)^T \mathbf{X}_{(\alpha)}(:, i'_1))$
 6: for $k = 2 : s'$ do
 7:     Compute the residual:
        $\overrightarrow{res} \leftarrow (\mathbf{X}_{(\alpha)}(:, i'_k) - \mathbf{RUR}^T \mathbf{X}_{(\alpha)}(:, i'_k))$
 8:     if $||\overrightarrow{res}|| \leq \epsilon ||\mathbf{X}_{(\alpha)}(:, i'_k)||$ then
 9:         continue
10:     else
11:         Compute: $\delta \leftarrow ||\overrightarrow{res}||^2$ and $\overrightarrow{\mathbf{y}} \leftarrow \mathbf{UR}^T \mathbf{X}_{(\alpha)}(:, i'_k)$
12:         Update $\mathbf{U}$:
            $\mathbf{U} \leftarrow \begin{pmatrix} \mathbf{U} + \overrightarrow{\mathbf{y}}\overrightarrow{\mathbf{y}}^T/\delta & -\overrightarrow{\mathbf{y}}/\delta \\ -\overrightarrow{\mathbf{y}}^T/\delta & 1/\delta \end{pmatrix}$
13:         Expand $\mathbf{R} : \mathbf{R} \leftarrow [\mathbf{R}, \mathbf{X}_{(\alpha)}(:, i'_k)]$
14:     end if
15: end for
16: Compute $\mathcal{C} \leftarrow \mathcal{X} \times_\alpha \mathbf{R}^T$
17: return $\mathcal{C}$, $\mathbf{U}$, $\mathbf{R}$
```

Algorithm 1 shows the procedure of CTD-S. First, CTD-S computes the probabilities of mode-α fibers of $\mathcal{X}$, which are proportional to the norm of each fiber, and then samples s fibers from $\mathcal{X}$ according to the probabilities with replacement, in lines 1-3. Redundant fibers exist in the sampled fibers in this step. CTD-S selects unique fibers from the initially sampled s fibers in line 4 where s' denotes the number of those unique fibers. This step reduces the number of iterations in lines 6-15 from $s - 1$ to $s' - 1$. $\mathbf{R}$ is initialized by the first sampled fiber in line 5. In lines 6-15, CTD-S removes redundant mode-α fibers in the sampled fibers. The matrices $\mathbf{U}$ and $\mathbf{R}$ are computed incrementally in this step. The columns of $\mathbf{R}$ always consist of independent mode-α fibers through the loop. In each iteration, CTD-S checks whether one of the sampled fibers is linearly independent of the column space spanned by $\mathbf{R}$ or not in lines 7-8, using the residual tolerance ϵ. If the fiber is independent, CTD-S updates $\mathbf{U}$ and expands $\mathbf{R}$ with the fiber in lines 10-13. Finally, CTD-S computes $\mathcal{C}$ with $\mathcal{X}$ and $\mathbf{R}$ in line 16.

Lemma 1 shows the computational cost of CTD-S.

Lemma 1. *The computational complexity of CTD-S is* $\mathcal{O}((\tilde{s}I_\alpha + s)N_\alpha + s'(\tilde{s}^2 + nnz(\mathbf{R})) + s\log s + nnz(\mathcal{X}))$*, where* N_α *is* $\prod_{n\neq\alpha} I_n$ *and* $\tilde{s} \ll s' \leq s$*.*

Proof. The mode-α matricization of $\mathcal{X}$ in line 1 needs $\mathcal{O}(nnz(\mathcal{X}))$ operations. Computing column distribution in line 2 takes $\mathcal{O}(nnz(\mathcal{X}) + N_\alpha)$ and sampling s columns in line 3 takes $\mathcal{O}(sN_\alpha)$. $\mathcal{O}(s\log s)$ operation is required in computing unique elements in I in line 4. Computing $\mathbf{R}$ and $\mathbf{U}$ in lines 5-15 takes $\mathcal{O}(s'(\tilde{s}^2 + nnz(\mathbf{R})))$ as proved in Lemma 1 in [23]. Computing $\mathcal{C}$ in line 16 takes $\mathcal{O}(\tilde{s}I_\alpha N_\alpha)$. Overall, CTD-S needs $\mathcal{O}((\tilde{s}I_\alpha + s)N_\alpha + s'(\tilde{s}^2 + nnz(\mathbf{R})) + s\log s + nnz(\mathcal{X}))$ operations. □

Lemma 2 shows that CTD-S has the optimal accuracy for given sampled fibers and $\epsilon = 0$, thus is more accurate than tensor-CUR.

Lemma 2. *CTD-S has the minimum error, thus is more accurate than tensor-CUR for a given* $\mathbf{R}_0$ *consisting of initially sampled fibers when the residual tolerance* $\epsilon = 0$*.*

Proof. CTD-S and tensor-CUR are both mode-α LR tensor decomposition methods. They both sample fibers from $\mathcal{X}$ in the same way in the initial step. Assume $\mathbf{R}_0$ be the matrix consisting of those initially sampled fibers, and the same $\mathbf{R}_0$ is given for CTD-S and tensor-CUR. Then, the reconstruction error of $\mathcal{X}$ given $\mathbf{R}_0$ is a function of $\mathbf{L}_{(\alpha)}$ as shown in Equation 4. The equality comes from Equation 3.

$$||\mathcal{X} - \mathcal{L} \times_\alpha \mathbf{R}_0||_F = ||\mathbf{X}_{(\alpha)} - \mathbf{R}_0\mathbf{L}_{(\alpha)}||_F \tag{4}$$

The reconstruction error is globally minimum when $\mathbf{L}_{(\alpha)} = \mathbf{R}_0^\dagger\mathbf{X}_{(\alpha)}$. Equation 5 shows the minimum reconstruction error.

$$\min_{\mathbf{L}_{(\alpha)}} ||\mathbf{X}_{(\alpha)} - \mathbf{R}_0\mathbf{L}_{(\alpha)}||_F = ||\mathbf{X}_{(\alpha)} - \mathbf{R}_0\mathbf{R}_0^\dagger\mathbf{X}_{(\alpha)}||_F \tag{5}$$

Let $\mathbf{R}$ be the factor of CTD-S. $\mathbf{R}$ consists of the independent columns of $\mathbf{R}_0$ since the tolerance $\epsilon = 0$. We show that CTD-S has the minimum reconstruction error in Equation 6.

$$\begin{aligned} ||\mathbf{X}_{(\alpha)} - \mathbf{R}_0\mathbf{R}_0^\dagger\mathbf{X}_{(\alpha)}||_F &= ||\mathbf{X}_{(\alpha)} - \mathbf{R}\mathbf{R}^\dagger\mathbf{X}_{(\alpha)}||_F \\ &= ||\mathbf{X}_{(\alpha)} - \mathbf{R}(\mathbf{R}^T\mathbf{R})^{-1}\mathbf{R}^T\mathbf{X}_{(\alpha)}||_F \\ &= ||\mathbf{X}_{(\alpha)} - \mathbf{R}\mathbf{U}\mathbf{C}_{(\alpha)}||_F \\ &= \text{ Error of CTD-S} \end{aligned} \tag{6}$$

The first equality in Equation 6 holds because $\mathbf{R}_0\mathbf{R}_0^\dagger\mathbf{X}_{(\alpha)}$ means the projection of $\mathbf{X}_{(\alpha)}$ onto the column space of $\mathbf{R}_0$, and $\mathbf{R}$ and $\mathbf{R}_0$ have the same column space. The third equality holds because CTD-S uses $(\mathbf{R}^T\mathbf{R})^{-1}$ for its factor $\mathbf{U}$ (theorem 1 in [23]), and $\mathbf{R}^T\mathbf{X}_{(\alpha)}$ for its factor $\mathcal{C}$. In contrast, tensor-CUR does not have the minimum reconstruction error because tensor-CUR has $\mathbf{L}_{(\alpha)}$ which is different from $\mathbf{R}_0^\dagger\mathbf{X}_{(\alpha)}$. Specifically, tensor-CUR further samples rows (called slabs) from $\mathbf{X}_{(\alpha)}$ to construct its $\mathbf{L}_{(\alpha)}$. □

CTD-D for Dynamic Tensors

Overview. How can we design an efficient sampling-based dynamic tensor decomposition method? In a dynamic setting, a new tensor arrives at every time step and we want to keep track of sampling-based tensor decomposition. The main challenge is to update factors quickly while preserving accuracy. Note that there has been no

Algorithm 2 CTD-D for Dynamic Tensor

```
Input: Tensor $\Delta\mathcal{X} \in \mathbb{R}^{I_1\times\cdots\times I_{N-1}\times 1}$, mode $\alpha \in \{1,\cdots,N-1\}$, $\mathcal{C}^{(t)}$, $\mathbf{U}^{(t)}$, $\mathbf{R}^{(t)}$, sample size
    $d \in \{1,\cdots,\Delta N_\alpha\}$, and tolerance $\epsilon$
Output: $\mathcal{C}^{(t+1)}$, $\mathbf{U}^{(t+1)}$, $\mathbf{R}^{(t+1)}$
 1: Let $\Delta\mathbf{X}_{(\alpha)}$ be the mode-$\alpha$ matricization of $\Delta\mathcal{X}$
 2: Compute column distribution for $i = 1,\cdots,\Delta N_\alpha$:
    $P(i) \leftarrow \frac{|\Delta\mathbf{X}_{(\alpha)}(:,i)|^2}{\|\Delta\mathbf{X}_{(\alpha)}\|_F^2}$
 3: Sample $d$ columns from $\Delta\mathbf{X}_{(\alpha)}$ based on $P(i)$. Let $I = \{i_1,\cdots,i_d\}$
 4: Let $I' = \{i'_1,\cdots,i'_{d'}\}$ be a set consisting of unique elements in $I$
 5: Initialize $\mathbf{R}^{(t+1)} \leftarrow \mathbf{R}^{(t)}$, $\mathbf{U}^{(t+1)} \leftarrow \mathbf{U}^{(t)}$, and $\Delta\mathbf{R} \leftarrow []$
 6: for $k = 1 : d'$ do
 7:     Let $\mathbf{x} \leftarrow \Delta\mathbf{X}_{(\alpha)}(:,i'_k)$
 8:     Compute the residual:
        $\overrightarrow{res} \leftarrow (\mathbf{x} - \mathbf{R}^{(t+1)}\mathbf{U}^{(t+1)}(\mathbf{R}^{(t+1)})^T\mathbf{x})$
 9:     if $||\overrightarrow{res}|| \le \epsilon||\mathbf{x}||$ then
10:         continue
11:     else
12:         Compute: $\delta \leftarrow ||\overrightarrow{res}||^2$ and $\overrightarrow{\mathbf{y}} \leftarrow \mathbf{U}^{(t+1)}(\mathbf{R}^{(t+1)})^T\mathbf{x}$
13:         Update $\mathbf{U}^{(t+1)}$:
            $\mathbf{U}^{(t+1)} \leftarrow \begin{pmatrix} \mathbf{U}^{(t+1)} + \overrightarrow{\mathbf{y}}\overrightarrow{\mathbf{y}}^T/\delta & -\overrightarrow{\mathbf{y}}/\delta \\ -\overrightarrow{\mathbf{y}}^T/\delta & 1/\delta \end{pmatrix}$
14:         Expand $\mathbf{R}^{(t+1)}$ and $\Delta\mathbf{R}$ : $\mathbf{R}^{(t+1)} \leftarrow [\mathbf{R}^{(t+1)},\mathbf{x}]$ and $\Delta\mathbf{R} \leftarrow [\Delta\mathbf{R},\mathbf{x}]$
15:     end if
16: end for
    Update $\mathbf{C}_{(\alpha)}^{(t+1)}$ :
17: if $\Delta\mathbf{R}$ is not empty then
18:     $\mathbf{C}_{(\alpha)}^{(t+1)} \leftarrow \begin{pmatrix} \mathbf{C}_{(\alpha)}^{(t)} & (\mathbf{R}^{(t)})^T\Delta\mathbf{X}_{(\alpha)} \\ (\Delta\mathbf{R})^T\mathbf{R}^{(t)}\mathbf{U}^{(t)}\mathbf{C}_{(\alpha)}^{(t)} & (\Delta\mathbf{R})^T\Delta\mathbf{X}_{(\alpha)} \end{pmatrix}$
19: else
20:     $\mathbf{C}_{(\alpha)}^{(t+1)} \leftarrow \begin{pmatrix} \mathbf{C}_{(\alpha)}^{(t)} & (\mathbf{R}^{(t)})^T\Delta\mathbf{X}_{(\alpha)} \end{pmatrix}$
21: end if
22: Fold $\mathbf{C}_{(\alpha)}^{(t+1)}$ into $\mathcal{C}^{(t+1)}$
23: return $\mathcal{C}^{(t+1)}$, $\mathbf{U}^{(t+1)}$, $\mathbf{R}^{(t+1)}$
```

sampling-based dynamic tensor decomposition method in the literature. Applying CTD-S at every time step is not a feasible option since it starts from scratch to update its factors, and thus running time increases rapidly as tensor grows. We propose CTD-D, the first sampling-based dynamic tensor decomposition method. CTD-D samples mode-α fibers only from the newly arrived tensor, and then updates the factors appropriately using those sampled ones. The main idea of CTD-D is to update the factors of CTD-S incrementally by (1) exploiting factors at previous time step and (2) reordering operations.

Algorithm. Fig 4 shows the scheme for CTD-D. At each time step, CTD-D samples fibers from newly arrived tensor and updates factors by checking linear dependency of sampled fibers with the factor at previous time step. Purple and green fiber are sampled from newly arrived tensor in Fig 4. Note that the purple fiber is added to the factor $\mathbf{R}$ since it is linearly independent of the fibers in the factor at the previous time step, while the linearly dependent green fiber is ignored.

Fig 4. The scheme for CTD-D.

For any time step t, CTD-D maintains its factors $\mathfrak{C}^{(t)} \in \mathbb{R}^{I_1\times\cdots\times I_{\alpha-1}\times\tilde{d}_t\times I_{\alpha+1}\times\cdots\times I_{N-1}\times t}$, $\mathbf{U}^{(t)} \in \mathbb{R}^{\tilde{d}_t\times\tilde{d}_t}$, and $\mathbf{R}^{(t)} \in \mathbb{R}^{I_\alpha\times\tilde{d}_t}$ such that $\mathfrak{X}^{(t)} \approx \mathfrak{C}^{(t)} \times_\alpha \mathbf{R}^{(t)}\mathbf{U}^{(t)}$, where the upper subscript (t) indicates that the factor is at time step t. $\mathfrak{X}^{(t)}$ grows along the time mode and we assume that N-th mode is the time mode in a dynamic setting, where N denotes the order of $\mathfrak{X}^{(t)}$. At the next time step $t+1$, CTD-D receives newly arrived tensor $\Delta\mathfrak{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times\cdots\times I_{N-1}\times 1}$ and updates $\mathfrak{C}^{(t)}$, $\mathbf{U}^{(t)}$, and $\mathbf{R}^{(t)}$ into $\mathfrak{C}^{(t+1)} \in \mathbb{R}^{I_1\times\cdots\times I_{\alpha-1}\times\tilde{d}_{t+1}\times I_{\alpha+1}\times\cdots\times I_{N-1}\times(t+1)}$, $\mathbf{U}^{(t+1)} \in \mathbb{R}^{\tilde{d}_{t+1}\times\tilde{d}_{t+1}}$, and $\mathbf{R}^{(t+1)} \in \mathbb{R}^{I_\alpha\times\tilde{d}_{t+1}}$, respectively such that $\mathfrak{X}^{(t+1)} \approx \mathfrak{C}^{(t+1)} \times_\alpha \mathbf{R}^{(t+1)}\mathbf{U}^{(t+1)}$.

Algorithm 2 shows the procedure of CTD-D. First, CTD-D computes the probabilities of mode-α fibers of $\Delta\mathfrak{X}$, which are proportional to the norm of each fiber, and then samples d fibers according to the probabilities with replacement in lines 1-3. CTD-D selects unique d' fibers in line 4 and initializes $\mathbf{R}^{(t+1)}$, $\mathbf{U}^{(t+1)}$, and $\Delta\mathbf{R}$ with $\mathbf{R}^{(t)}$, $\mathbf{U}^{(t)}$, and an empty matrix respectively in line 5, where $\Delta\mathbf{R}$ consists of differences between $\mathbf{R}^{(t)}$ and $\mathbf{R}^{(t+1)}$. In lines 6-16, CTD-D expands $\mathbf{R}^{(t+1)}$ with those sampled fibers by sequentially evaluating linear dependency of each fiber with the column space of $\mathbf{R}^{(t+1)}$. $\mathbf{R}^{(t+1)}$ and $\mathbf{U}^{(t+1)}$ are updated in this step. Finally, $\mathbf{C}^{(t+1)}_{(\alpha)}$ is updated in lines 17-21.

In the following, we describe two main ideas of CTD-D to update $\mathbf{C}^{(t+1)}_{(\alpha)}$, $\mathbf{R}^{(t+1)}$, and $\mathbf{U}^{(t+1)}$ efficiently while preserving accuracy: exploiting factors at previous time step, and reordering operations.

(1) Exploiting factors at previous time step: First, we explain how we update $\mathbf{R}^{(t+1)}$ and $\mathbf{U}^{(t+1)}$ using the idea. In line 5 of Algorithm 1, CTD-S initializes $\mathbf{R}$ and $\mathbf{U}$ using one of the sampled fibers. This is because CTD-S requires $\mathbf{R}$ to consist of linearly independent columns and it is satisfied when $\mathbf{R}$ has only one fiber. Since $\mathbf{R}^{(t)}$ already consists of linearly independent columns, we initialize $\mathbf{R}^{(t+1)}$ and $\mathbf{U}^{(t+1)}$ with $\mathbf{R}^{(t)}$ and $\mathbf{U}^{(t)}$ respectively in line 5 of Algorithm 2. In lines 6-16, we check linear independence of each sampled fiber from $\Delta\mathbf{X}_{(\alpha)}$ with $\mathbf{R}^{(t+1)}$. If the fiber is linearly independent, we expand $\mathbf{R}^{(t+1)}$ and update $\mathbf{U}^{(t+1)}$ as in the lines 11-13 of Algorithm 1.

Second, we describe how we update $\mathfrak{C}^{(t+1)}$ using the idea. We assume that $\Delta\mathbf{R}$ is not empty after line 16 of Algorithm 2. At time step t and its successor step $t+1$, CTD-S satisfies Equations 7 and 8, where $\mathbf{C}^{(t)}_{(\alpha)}$ has the size $\tilde{d}_t \times N^{(t)}_\alpha$ and $\mathbf{C}^{(t+1)}_{(\alpha)}$ has the size $\tilde{d}_{t+1} \times N^{(t+1)}_\alpha$.

$$\mathbf{C}^{(t)}_{(\alpha)} \leftarrow (\mathbf{R}^{(t)})^T\mathbf{X}^{(t)}_{(\alpha)} \tag{7}$$

$$\mathbf{C}^{(t+1)}_{(\alpha)} \leftarrow (\mathbf{R}^{(t+1)})^T\mathbf{X}^{(t+1)}_{(\alpha)} \tag{8}$$

We can rewrite $\mathbf{R}^{(t+1)}$ and $\mathbf{X}^{(t+1)}_{(\alpha)}$ as Equations 9 and 10 respectively, where $\Delta\mathbf{R}$ has the size $I_\alpha \times \Delta\tilde{d}$ and $\Delta\mathbf{X}_{(\alpha)}$ has the size $I_\alpha \times \Delta N_\alpha$ such that $N^{(t+1)}_\alpha = N^{(t)}_\alpha + \Delta N_\alpha$ and $\tilde{d}_{t+1} = \tilde{d}_t + \Delta\tilde{d}$.

$$\mathbf{R}^{(t+1)} = \begin{bmatrix}\mathbf{R}^{(t)} & \Delta\mathbf{R}\end{bmatrix} \tag{9}$$

$$\mathbf{X}^{(t+1)}_{(\alpha)} = \begin{bmatrix}\mathbf{X}^{(t)}_{(\alpha)} & \Delta\mathbf{X}_{(\alpha)}\end{bmatrix} \tag{10}$$

We replace $\mathbf{R}^{(t+1)}$ and $\mathbf{X}^{(t+1)}_{(\alpha)}$ in Equation 8 with those in Equations 9 and 10, respectively, to obtain the Equation 11.

$$\begin{aligned}\mathbf{C}^{(t+1)}_{(\alpha)} &\leftarrow \begin{bmatrix}(\mathbf{R}^{(t)})^T \\ (\Delta\mathbf{R})^T\end{bmatrix}\begin{bmatrix}\mathbf{X}^{(t)}_{(\alpha)} & \Delta\mathbf{X}_{(\alpha)}\end{bmatrix} \\ &= \begin{bmatrix}(\mathbf{R}^{(t)})^T\mathbf{X}^{(t)}_{(\alpha)} & (\mathbf{R}^{(t)})^T\Delta\mathbf{X}_{(\alpha)} \\ (\Delta\mathbf{R})^T\mathbf{X}^{(t)}_{(\alpha)} & (\Delta\mathbf{R})^T\Delta\mathbf{X}_{(\alpha)}\end{bmatrix}\end{aligned} \tag{11}$$

CTD-S computes all the 4 elements ($(\mathbf{R}^{(t)})^T\mathbf{X}^{(t)}_{(\alpha)}$, $(\mathbf{R}^{(t)})^T\Delta\mathbf{X}_{(\alpha)}$, $(\Delta\mathbf{R})^T\mathbf{X}^{(t)}_{(\alpha)}$, and $(\Delta\mathbf{R})^T\Delta\mathbf{X}_{(\alpha)}$) in Equation 11 from scratch, hence requires a lot of computations. To make computation of $\mathbf{C}^{(t+1)}_{(\alpha)}$ incremental, we exploit existing factors at time step t : $\mathbf{C}^{(t)}_{(\alpha)}$, $\mathbf{R}^{(t)}$, and $\mathbf{U}^{(t)}$. First, we use $\mathbf{C}^{(t)}_{(\alpha)}$ instead of $(\mathbf{R}^{(t)})^T\mathbf{X}^{(t)}_{(\alpha)}$ as in the Equation 7. Second, we should replace $\mathbf{X}^{(t)}_{(\alpha)}$ in $(\Delta\mathbf{R})^T\mathbf{X}^{(t)}_{(\alpha)}$ with the factors at time step t, since CTD-D does not have $\mathbf{X}^{(t)}_{(\alpha)}$ as its input unlike CTD-S. We substitute $\mathbf{R}^{(t)}\mathbf{U}^{(t)}\mathbf{C}^{(t)}_{(\alpha)}$ for $\mathbf{X}^{(t)}_{(\alpha)}$. This is because CTD-S ensures $\mathfrak{X}^{(t)} \approx \mathfrak{C}^{(t)} \times_\alpha \mathbf{R}^{(t)}\mathbf{U}^{(t)}$ which can be rewritten as $\mathbf{X}^{(t)}_{(\alpha)} \approx \mathbf{R}^{(t)}\mathbf{U}^{(t)}\mathbf{C}^{(t)}_{(\alpha)}$ by Equation 3. Equation 12 shows the final form of $\mathbf{C}^{(t+1)}_{(\alpha)}$ which is the same as line 18 in Algorithm 2.

$$\mathbf{C}^{(t+1)}_{(\alpha)} \leftarrow \begin{bmatrix} \mathbf{C}^{(t)}_{(\alpha)} & (\mathbf{R}^{(t)})^T\Delta\mathbf{X}_{(\alpha)} \\ (\Delta\mathbf{R})^T\mathbf{R}^{(t)}\mathbf{U}^{(t)}\mathbf{C}^{(t)}_{(\alpha)} & (\Delta\mathbf{R})^T\Delta\mathbf{X}_{(\alpha)} \end{bmatrix} \tag{12}$$

$(\Delta\mathbf{R})^T\mathbf{R}^{(t)}\mathbf{U}^{(t)}\mathbf{C}^{(t)}_{(\alpha)}$ and $(\Delta\mathbf{R})^T\Delta\mathbf{X}_{(\alpha)}$ are ignored when $\Delta\mathbf{R}$ is empty as expressed in line 20 of Algorithm 2.

(2) Reordering computations: The computation order for the element $(\Delta\mathbf{R})^T\mathbf{R}^{(t)}\mathbf{U}^{(t)}\mathbf{C}^{(t)}_{(\alpha)}$ is important since each order has a different computation cost. We want to determine the optimal parenthesization among possible parenthesizations. It can be shown that $(((\Delta\mathbf{R})^T\mathbf{R}^{(t)})\mathbf{U}^{(t)})\mathbf{C}^{(t)}_{(\alpha)}$ is the optimal one with $\mathcal{O}((\Delta\tilde{d})\tilde{d}_t(I_\alpha + \tilde{d}_t + N^{(t)}_\alpha))$ operations and can be done by parenthesizing from the left.

We prove that CTD-D is faster than CTD-S in Lemma 3.

Lemma 3. *CTD-D is faster than CTD-S. The computational complexity of CTD-D is* $\mathcal{O}((\Delta\tilde{d})\tilde{d}_t(N^{(t)}_\alpha + I_\alpha) + (\tilde{d}_{t+1}I_\alpha + d)\Delta N_\alpha + d'(\tilde{d}^2_{t+1} + nnz(\mathbf{R}^{(t+1)})) + d\log d + nnz(\Delta\mathfrak{X}))$.

Proof. The lines 1-4 of Algorithm 2 for CTD-D are similar to those of Algorithm 1 for CTD-S. The only difference is that CTD-D samples d columns from $\Delta\mathbf{X}_{(\alpha)}$ while CTD-S samples s columns from $\mathbf{X}_{(\alpha)}$. Thus, lines 1-4 takes $\mathcal{O}(nnz(\Delta\mathfrak{X}) + d\Delta N_\alpha + d\log d)$. Updating $\mathbf{R}^{(t+1)}$ and $\mathbf{U}^{(t+1)}$ in lines 5-16 needs $\mathcal{O}(d'(\tilde{d}^2_{t+1} + nnz(\mathbf{R}^{(t+1)})))$ operations as proved in Lemma 1 in [23]. In updating $\mathfrak{C}^{(t+1)}$ in lines 17-18, $(\mathbf{R}^{(t)})^T\Delta\mathbf{X}_{(\alpha)}$ takes computational cost of $\mathcal{O}(\tilde{d}_tI_\alpha\Delta N_\alpha)$. $(\Delta\mathbf{R})^T\Delta\mathbf{X}_{(\alpha)}$ takes $\mathcal{O}(\Delta\tilde{d}I_\alpha\Delta N_\alpha)$ and $(\Delta\mathbf{R})^T\mathbf{R}^{(t)}\mathbf{U}^{(t)}\mathbf{C}^{(t)}_{(\alpha)}$ takes $\mathcal{O}((\Delta\tilde{d})\tilde{d}_t(I_\alpha + \tilde{d}_t + N^{(t)}_\alpha))$. Overall, CTD-D takes $\mathcal{O}((\Delta\tilde{d})\tilde{d}_t(N^{(t)}_\alpha + I_\alpha) + (\tilde{d}_{t+1}I_\alpha + d)\Delta N_\alpha + d'(\tilde{d}^2_{t+1} + nnz(\mathbf{R}^{(t+1)})) + d\log d + nnz(\Delta\mathfrak{X}))$.

CTD-D is faster than CTD-S because CTD-S has $\tilde{s}I_\alpha N_\alpha$ in its complexity, which is much larger than all the terms in the complexity of CTD-D. □

Experiments

We perform experiments to answer the following questions.

Q1: What is the performance of our static method CTD-S compared to the competing method tensor-CUR?

Q2: How do the performance of CTD-S and tensor-CUR change with regard to the sample size parameter?

Q3: What is the performance of our dynamic method CTD-D compared to the static method CTD-S?

Q4: What are the results of applying CTD-D for online DDoS attack detection and online troll detection?

Experimental Settings

Measure. We define three metrics (1. *Relative Error*, 2. *Memory*, and 3. *Time*) as follows. First, a *Relative Error* is defined as Equation 13. $\mathcal{X}$ denotes the original tensor and $\tilde{\mathcal{X}}$ is the tensor reconstructed from the factors of $\mathcal{X}$. For example, $\tilde{\mathcal{X}} = \mathcal{C} \times_\alpha \mathbf{RU}$ in CTD-S.

$$Relative\ Error = \frac{||\tilde{\mathcal{X}} - \mathcal{X}||_F^2}{||\mathcal{X}||_F^2} \tag{13}$$

Second, *Memory* is defined as Equation 14. It measures the relative amount of memory needed for storing the resulting factors. The denominator and numerator indicate the amount of memory needed for storing the original tensor and the resulting factors, respectively.

$$Memory = \frac{nnz(\mathcal{C}) + nnz(\mathbf{U}) + nnz(\mathbf{R})}{nnz(\mathcal{X})} \tag{14}$$

Finally, *Time* denotes running time in seconds.

Data. Table 3 shows the data we used in our experiments.

Table 3. Summary of the tensor data used.

Name	I_1	I_2	I_3	**Nonzeros**
Facebook-wall [25]	63,891	63,890	1,504	738,485
Facebook-wall (synthetic) [30]	63,891	63,890	1,504	1,169,656
Hyperspectral Image [26]	538	323	148	25,715,854
Infectious [27]	407	410	1,392	17,298
Hypertext 2009 [28]	112	113	5,246	20,818
Haggle [29]	77	274	1,567	27,972
CAIDA [30]	189	189	1,000	20,511
CAIDA (synthetic) [30]	189	189	1,000	46,102

Machine. All the experiments are performed on a machine with a 10-core Intel 2.20 GHz CPU and 256 GB RAM.

Competing method. We compare our proposed method CTD with tensor-CUR [16], the state-of-the-art sampling-based tensor decomposition method. Both methods are implemented in MATLAB.

Performance of CTD-S

We measure the performance of CTD-S to answer Q1 and Q2. In summary, CTD-S is up to 11× more accurate for the same level of running time, 2.3× faster, and 24× more memory-efficient for the same level of error compared to tensor-CUR. Compared to the tensor-CUR, CTD-S is more accurate, and its running time and memory usage are relatively constant over various sample sizes. The detail of the experiment is as follows.

Both CTD-S and tensor-CUR takes a given tensor $\mathcal{X}$, mode α, and a sample size s as input because they are LR tensor decomposition methods. In each experiment, we give the same input and compare the performance. We set $\alpha = 1$ and perform experiments under various sample sizes s. We set the number of slabs to sample $r = s$ and the rank $k = 10$ in tensor-CUR, and set $\epsilon = 10^{-6}$ in CTD-S.

Fig 1 shows the *Time* vs. *Relative Error* and *Memory* vs. *Relative Error* of CTD-S compared to tensor-CUR under various sample sizes, which are the answers for Q1. The error of tensor-CUR is larger than that of CTD-S. This phenomenon coincides with the Lemma 2, which guarantees that CTD-S is more accurate than tensor-CUR theoretically. We compare running time and memory usage under the same level of error and not under the same sample size due to a large gap between the errors of CTD-S and that of tensor-CUR under the same sample size.

Fig 5 shows the *Relative Error*, *Time*, and *Memory* of CTD-S compared to those of tensor-CUR over increasing sample sizes s for the Haggle dataset that answers questions in Q2. The error of CTD-S decreases as s increases because it gains more data to sample important fibers which describe the original tensor well. The running time and memory usage of CTD-S are relatively constant compared to those of tensor-CUR. This is because CTD-S keeps only the linearly independent fibers, the number of which is bound by the rank of $\mathbf{X}_{(\alpha)}$. There are small fluctuations in the graphs since the sampling process of both CTD-S and tensor-CUR are based on randomness.

Fig 5. Error, running time, and memory usage of CTD-S compared to those of tensor-CUR over sample size s for Haggle dataset. CTD-S is more accurate over various sample sizes, and its running time and memory usage are relatively constant compared to the tensor-CUR.

Performance of CTD-D

We compare the performance of CTD-D with those of CTD-S to answer Q3. In summary, CTD-D is up to 82× faster for the same level of error compared to CTD-S. The detail is as follows.

To simulate a dynamic environment, we divide a given dataset into two parts along the time mode. We use the first 80% of the dataset as historical data and the later 20% as incoming data. We assume that historical data is already given and incoming data arrives sequentially at every time step, such that the whole data grows along the time mode. We measure the performance of CTD-D and CTD-S at each time step and calculate the average. We set the sample size d of CTD-D to be much smaller than that of CTD-S because CTD-D samples fibers only from the increment $\Delta\mathfrak{X}$ while CTD-S samples from the whole data $\mathfrak{X}$. We set d of CTD-D to be 0.01 times s of CTD-S, $\alpha = 1$, and $\epsilon = 10^{-6}$.

Fig 2 shows the *Time* vs. *Relative Error* and *Memory* vs. *Relative Error* relation of CTD-D compared to those of CTD-S. Note that CTD-D is much faster than CTD-S for all the datasets. The reason why CTD-D is especially faster for the Hyperspectral Image dataset is that the dataset has relatively many dependent fibers, which makes CTD-D skip updating $\mathbf{U}$, compared to the other datasets. CTD-D uses the same or slightly more memory than CTD-S does. This is because multiplication between sparse matrices used in updating $\mathfrak{C}$ does not always produce sparse output, thus the number of nonzero entries in $\mathfrak{C}$ increases slightly over time steps.

CTD At Work

In this section, we apply CTD-D to online DDoS attack detection in network traffic data and online troll detection in social network data. We show how CTD-D's interpretability can help successfully detect DDoS attacks and trolls.

Online DDoS attack detection

A DDoS attack makes an online service unavailable by sending a huge amount of traffic to the server from multiple sources. DDoS attacks are still major threats to many companies. In effect, 20% of financial companies get $1 million revenue loss per hour and 43% lose more than $250,000 hourly under DDoS attack, while 74% take more than 1 hour to shut down the attacks [24].

Our goal is to detect DDoS attacks in network traffic data efficiently in an online fashion. We propose a novel online DDoS attack detection method based on CTD-D's interpretability. We show that CTD-D is one of the feasible options for online DDoS attack detection and show how it detects attacks successfully. In contrast to the standard PARAFAC and Tucker decomposition methods, CTD-D can determine DDoS attacks from its decomposition result without expensive overhead. We aim to dynamically find a victim (destination host) and corresponding attackers (source hosts) of each DDOS attack in network traffic data that is when a victim receives a huge amount of traffic from a large number of attackers.

The online DDoS attack detection method based on CTD-D is as follows. First, we apply CTD-D on network traffic data which is a 3-mode tensor in the form of (source IP - destination IP - time). We assume an online environment where each slab of the network traffic data in the form of (source IP - destination IP) arrives sequentially at every time step. We use source IP mode as mode α. Second, we inspect the factor $\mathbf{R}$ of CTD-D, which consists of actual mode-α fibers from the original data. $\mathbf{R}$ is composed of important mode-α fibers which signify major activities such as DDoS attack or heavy traffic to the main server. Thanks to CTD, we can directly find out destination host and occurrence time of a major activity represented in a fiber in $\mathbf{R}$, by simply tracking the indices of fibers. We regard fibers with the same destination host index represent the same major activity, and consider the first fiber among those with the same destination host index to be the representative of each major activity. Then, we select fibers with the norm higher than the average among the first fibers and suggest them as candidates of DDoS attack. This is because DDoS attacks have much higher norms than normal traffic does.

We generate network traffic data by injecting DDoS attacks on the real-world CAIDA network traffic dataset [30]. We assume that randomly selected 20% of source hosts participate in each DDoS attack. Table 4 shows the result of DDoS attack detection method of CTD-D. CTD-D achieves high F1 score for various number n of injected DDoS attacks with notable precision. We set $d = 10$, and $\epsilon = 0.15$.

Table 4. The result of online DDoS attack detection method based on CTD-D. CTD-D achieves high F1 score for various n with notable precision, where n denotes the number of injected DDoS attacks.

n	Recall	Precision	F1 score
1	1.000	1.000	1.000
3	1.000	1.000	1.000
5	0.880	1.000	0.931
7	0.857	1.000	0.921

Online troll detection

Recent social network services (SNS) such as Facebook or Twitter has billions of users; their main concern is to detect trolls, or abnormal users, since trolls can severely undermine the service. Our goal is to detect trolls in social network tensor data in an online fashion. We define a troll as an abnormal user who posts on the other users' walls much more than normal users do. We show how CTD-D finds trolls successfully using its interpretability.

We use a process similar to the online DDoS attack detection method based on CTD-D described in the previous section to find trolls. We use the real-world Facebook-wall social network tensor, a 3-mode tensor containing triplets where each entry denotes the number of posts for the corresponding triplet. A triplet (User 1 - User 2 - time) means that User 2 posted on the User 1's wall. We assume an online environment where new data point in the form of (User 1 - User 2) arrives at every time step. We apply CTD-D with User 1 mode for α so that each fiber collected in the factor $\mathbf{R}$ represents User 2's behavior at some time. By tracking indices of fibers in the factor $\mathbf{R}$, we can reveal which fiber represents behaviors of which users at which time. We then decide trolls (User 2) by picking fibers which have norm larger than the average.

We test the ability of CTD to interpretability detect trolls by inserting synthetic trolls into the Facebook-wall dataset. Table 5 shows the result of online troll detection in Facebook-wall dataset based on CTD-D. It is notable that we can detect all the trolls inserted with very small sample size, $10^{-4}\%$ of the entire fibers, for a various number of trolls.

Table 5. The result of online troll detection in Facebook-wall dataset based on CTD-D. CTD-D detects all the trolls inserted ($recall = 1$) for various n, where n denotes the number of injected troll users. Note that we used only $10^{-4}\%$ of the entire fibers as a sample size.

n	Recall	Precision	F1 score
1	1.000	0.200	0.333
3	1.000	0.500	0.667
5	1.000	0.556	0.714
10	1.000	0.833	0.909

Conclusion

We propose CTD, a fast, accurate, and directly interpretable tensor decomposition method based on sampling. The static version CTD-S is up to 11× more accurate, 2.3× faster, and 24× more memory-efficient compared to the state-of-the-art method. The dynamic version CTD-D is up to 82× faster than CTD-S for an online environment. CTD-D is the first method providing interpretable *dynamic* tensor decomposition. Utilizing the interpretability of CTD, we were able to successfully detect online DDoS attacks and trolls from network data. The interpretability of CTD comes from the assumption that the original data fiber itself is sparse and interpretable, such as IP address or words in documents. Although not all real-world data have this property, such as values in gene expression data, a wide range of social and technological data in the online environment does have the sparse and interpretable properties. In such cases, CTD is capable of dynamically detecting important or abnormal data in an online environment.

Acknowledgments 444

This research was supported by Basic Science Research Program through the NRF of Korea (NRF-2015R1C1A2A01055739) and by the KEIT Korea under the "Global Advanced Technology Center" (10053204).

References

1. Khoa NLD, Zhang B, Wang Y, Liu W, Chen F, Mustapha S, et al.; PAKDD. On Damage Identification in Civil Structures Using Tensor Analysis. 2015;.
2. Prada MA, Toivola J, Kullaa J, HollméN J. Three-way Analysis of Structural Health Monitoring Data. Neurocomput. 2012;.
3. Wang Y, Chen R, Ghosh J, Denny JC, Kho A, Chen Y, et al. Rubik: Knowledge Guided Tensor Factorization and Completion for Health Data Analytics. KDD; 2015.
4. Cyganek B, Woźniak M. Tensor based representation and analysis of the electronic healthcare record data. BIBM; 2015.
5. Perros I, Chen R, Vuduc R, Sun J. Sparse Hierarchical Tucker Factorization and Its Application to Healthcare. ICDM; 2015.
6. Thuraisingham BM. Data Mining for Security Applications. WISI; 2006.
7. Phua C, Lee VCS, Smith-Miles K, Gayler RW. A Comprehensive Survey of Data Mining-based Fraud Detection Research; 2010.
8. Allanach J, Tu H, Singh S, Willett P, Pattipati K. Detecting, tracking, and counteracting terrorist networks via hidden Markov models. IEEE Aerospace; 2004.
9. Arulselvan A, Commander CW, Elefteriadou L, Pardalos PM. Detecting Critical Nodes in Sparse Graphs. Comput Oper Res. 2009;.
10. Cao Q, Sirivianos M, Yang X, Pregueiro T. Aiding the Detection of Fake Accounts in Large Scale Social Online Services. NSDI; 2012. p. 15–15.
11. Kontaxis G, Polakis I, Ioannidis S, Markatos EP. Detecting social network profile cloning. PERCOM; 2011.
12. Harshman RA. Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multi-modal factor analysis. 1970;.
13. Tucker LR. Some mathematical notes on three-mode factor analysis. Psychometrika. 1966;.
14. Drineas P, Mahoney MW. A randomized algorithm for a tensor-based generalization of the singular value decomposition. Linear algebra and its applications. 2007;.
15. Caiafa CF, Cichocki A. Generalizing the column–row matrix decomposition to multi-way arrays. Linear Algebra and its Applications. 2010;.
16. Mahoney MW, Maggioni M, Drineas P. Tensor-CUR decompositions for tensor-based data. SIAM J Matrix Anal Appl. 2008;.

17. Sun J, Tao D, Faloutsos C. Beyond streams and graphs: dynamic tensor analysis. KDD; 2006.

18. Sun J, Papadimitriou S, Yu PS. Window-based Tensor Analysis on High-dimensional and Multi-aspect Streams. ICDM; 2006.

19. Zhou S, Nguyen XV, Bailey J, Jia Y, Davidson I. Accelerating Online CP Decompositions for Higher Order Tensors. KDD; 2016.

20. Drineas P, Mahoney MW, Muthukrishnan S. Relative-error CUR matrix decompositions. SIAM J Matrix Anal Appl. 2008;.

21. Drineas P, Kannan R, Mahoney M. Fast Monte Carlo algorithms for matrices III: Computing a compressed approximate matrix decomposition. SIAM J Comput. 2006;.

22. Sun J, Xie Y, Zhang H, Faloutsos C. Less is More: Compact Matrix Decomposition for Large Sparse Graphs. SDM; 2007.

23. Tong H, Papadimitriou S, Sun J, Yu PS, Faloutsos C. Colibri: fast mining of large static and dynamic graphs. KDD; 2008.

24. Korolov M. DDoS costs, damages on the rise. CSO News, 2016.

25. http://socialnetworks.mpi-sws.org/data-wosn2009.html

26. http://www.imageval.com/scene-database-4-faces-3-meters/

27. http://konect.uni-koblenz.de/networks/sociopatterns-infectious

28. http://konect.uni-koblenz.de/networks/sociopatterns-hypertext

29. http://konect.uni-koblenz.de/networks/contact

30. https://github.com/leesael/CTD