SCED: A General Framework for Sparse Tensor Decompositions with Constraints and Elementwise Dynamic Learning

Anonymous

Abstract—**CANDECOMP/PARAFAC Decomposition (CPD) is one of the most popular and widely applied tensor decomposition methods. In recent years, sparse tensors that contain a huge portion of zeros but a limited number of non-zeros have attracted much interest. It can be problematic to directly apply existing decomposition algorithms to sparse tensors, since they are typically engineered for dense tensors and have lower efficiency for sparse ones. Furthermore, other issues related to sparsity can arise due to different data sources and application requirements. In particular, the role of zero entries may vary and it is often necessary to incorporate constraints such as non-negativity. The ability to learn on-the-fly is also a must for dynamic scenarios. State-of-the-art tensor decomposition algorithms only partially address the above issues. To fill this gap, we propose a general framework for finding CPD of sparse tensors. We show how to model the tensor decomposition problem by a generalized weighted CPD formulation and solve it efficiently. Our proposed method is also flexible in incorporating constraints and for incremental scenarios such as dynamic data streams. Via experiments on both synthetic and real-world datasets, we demonstrate significant improvements for our approach in terms of effectiveness, efficiency and scalability.**

1. Introduction

Multi-dimensional data is a daily feature of our lives, from video clips [1], to time-evolving graphs/networks such as social networks [2], to spatio-temporal data like fMRIs [3], [4]. The *Tensor*, a multi-way generalization of the matrix, is a natural representation for such data because of its ability to maintain the structural information. However, working with tensors is not easy due to the complex relationships among different dimensions. As a result, in order to simplify data, extract useful features and discover meaningful knowledge, *CANDECOMP/PARAFAC Decomposition* (CPD), a tool for tensors similar to PCA and SVD for matrices, has been extensively studied and widely applied in recent years [5], [6], [7].

As shown in Figure 1, the focus of this paper is to develop a framework (SCED) that addresses three challenges for tensor decomposition.: *Problem 1)* how to develop a unified formulation for finding the CPD of general sparse tensors together with an efficient solving algorithm; *Problem 2)* how to incorporate constrained decompositions, since they often provide more meaningful results; and *Problem 3)* how to make the decomposition algorithm incrementally handle dynamic updates to the tensor (the dynamic aspect we focus on here is different from existing online or incremental CPD works like [22], [23], [24], where the tensor is growing slice by slice. In this paper, a data stream consists of new individual cell entries that can dynamically be updated at any position in the tensor).

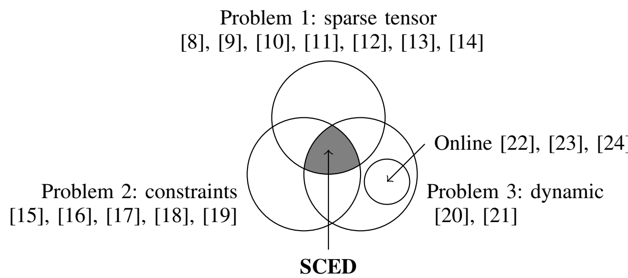


Figure 1. Research problem

An example motivating our research problem is context-aware recommender systems, where the ratings can be modeled by a three-way sparse tensor as $user \times item \times context$. Besides challenges imposed by the large scale and sparsity, there are several important issues to highlight.

First, the role of zero entries might be different depending on rating type. For example, zero entries in a system with explicit ratings are usually treated as missing values and ignored [11], [13], [14], while for implicit feedback like user click logs, one cannot simply discard these valuable zeros as they represent hidden preferences such as dislikes [12]. However, existing methods are usually specifically designed for one of above cases and lack the capability to generalize. Thus, an algorithm which works for one type of data may not be applicable and easily adaptable to other types.

Second, in practice, there exists a rich body of domain knowledge and this can be modeled as constraints that a CPD needs to satisfy. With the help of constraints, the produced decomposition will be more meaningful and interpretable. One example is non-negativity [15], [16], [17], [18] where each user's preference is modeled by a set of non-negative coefficients, which stand for his/her favor to each latent item/context groups. Therefore, it is important to efficiently incorporate this side information. Current constrained decomposition techniques have mostly been developed for dense tensors and face challenges in efficiency for sparse ones.

Third, it is common to see that after learning a decomposition model from historical data, a large amount of new data, $(user, item, context)$ tuples in this example, is generated. A static model can quickly become outdated in such a scenario and it is too expensive to recompute a new CPD due to the high time complexity. Therefore, a dynamic (incremental) learning model is desired, but to the best of our knowledge, there are no existing approaches that can be used in this scenario for tensors.

Overall, existing methods have partially or separately addressed the above issues, but how to handle them together in a unified manner is still an open question. To close this gap, we propose a new algorithm, SCED. Our contributions are as follows:

- We propose a new formulation and an efficient algorithm to find CPD of sparse tensors.
- We enhance our method with the capability to incorporate constraints, and the ability to dynamically track new CPDs on-the-fly.
- Via experiments on synthetic and real-world datasets, we demonstrate the effectiveness of our framework in terms of effectiveness, efficiency and scalability, compared to state-of-the-art baselines.

2. Related Work

Most existing works for sparse CPD specifically target on one of three special cases: 1) *True Observation* (TO) that treats zeros the same as non-zero observations; 2) *Missing Value* (MV) that ignores all zeros; and 3) *Implicit Information* (II) where zeros slightly contribute to the model, but less useful than non-zeros. For TO case, Bader and Kolda [8] proposed an algorithm based on *Alternating Least Squares* (ALS) by tailoring it to sparse tensors, with support from special data structures and customized tensor-related operations. With the advance in distributed computing, techniques like MapReduce are also used to further speed up ALS for large scale sparse tensors [9], [10]. In terms of MV situation, Acar *et al.*'s weighted CPD is a well suit framework and an optimization based algorithm, WOPT [11], is proposed to decompose sparse tensors with missing values. Apart from this, Bayesian methods are also explored by Rai *et al.* [13] and Zhao *et al.* [14]. Compared to TO and MV, II is a less studied case by tensor researchers, the only work can be found is [12]. This is an extension of *Matrix Factorization* (MF) works for implicit feedbacks [20], [25], [26] to tensors, where a small uniform weight is assigned to zero entries.

In terms of constrained CPD, non-negativity and sparseness are two popular constraints mainly explored by researchers. Early attempt [15] handled non-negativity based on Lee and Seung's *Multiplicative Update* (MU) rule [27]. Additionally, non-negative CPD algorithms based on ANLS [16], HALS [17] and Newton method [18] are also proposed. Similar to MF, to promote sparse solutions, l_1-norm is usually used as regularization to CPD and an implementation of this idea can be found in [19].

Regarding to learn CPD with dynamic data stream, there are limited number of researches on online CPD [22], [23],

TABLE 1. Comparison to related works

	General	Efficient	Scalable	Constraints	Dynamic
ALS [8]	TO	✓	✓		
MU [15]	TO	✓	✓	✓	
WOPT [11]	MV				
iTALS [12]	II				✓*
SCED	TO, MV, II	✓	✓	✓	✓

* not reported in original paper, but applicable with modifications

TABLE 2. Notations and basic operations

a, $\mathbf{a}$, $\mathbf{A}$, $\mathfrak{X}$	scalar, vector, matrix, tensor
$\mathbf{a}_{i\cdot}$, $\mathbf{a}_{\cdot j}$	the i-th row and j-th column vectors of $\mathbf{A}$
a_{ij}	the (i,j)-th element of $\mathbf{A}$
$\mathbf{A}^\top$, $\mathbf{A}^{-1}$, $\|\|\mathbf{A}\|\|$	transpose, inverse and Frobenius norm of $\mathbf{A}$
$\mathbf{A}^{(n)}$	the n-the matrix
$\odot$, $\circledast$	Khatri-Rao product, elementwise product
$\bigodot^{-n}\mathbf{A}^{(n)}$	$\mathbf{A}^{(N)}\cdots\odot\mathbf{A}^{(n+1)}\odot\mathbf{A}^{(n-1)}\odot\cdots\mathbf{A}^{(1)}$
$\circledast^{-n}\mathbf{A}^{(n)}$	$\mathbf{A}^{(N)}\cdots\circledast\mathbf{A}^{(n+1)}\circledast\mathbf{A}^{(n-1)}\circledast\cdots\mathbf{A}^{(1)}$
$\mathbf{X}_{(n)}$	mode-n unfolding of $\mathfrak{X}$
Ω	indices set for all entries in a tensor
Ω^+, Ω^-	indices sets for non-zero and zero entries

[24] that new data is appending slice by slice, which is not suitable for the dynamic scenario we aim in this paper. The most related papers in the literature are for MF [20], [21], where elementwise dynamic changes are handled by only refreshing corresponding rows related to the new entry.

Overall, only partial solutions existed for our research question and it is necessary to develop a general framework that can address all aforementioned critical issues altogether. Lastly, a comparison between our approach and the most relevant and representative works is shown in Table 1.

3. Preliminaries and Background

We summarize the notations used through this paper in Table 2. Given an N^{th}-order tensor $\mathfrak{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, CPD approximates it by N *loading* matrices $\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)}$, and the $(i_1, \dots, i_N)$-th entry of $\mathfrak{X}$ is estimated as

$$\hat{x}_{i_1,\dots,i_N} = \sum_{r=1}^{R} \prod_{n=1}^{N} a^{(n)}_{i_n r}, \tag{1}$$

where R is the decomposition rank. For conventional CPD (the TO case), all entries are fitted by such latent factor model. Specifically, the loss function can be written as

$$\begin{aligned}\mathcal{L}_{cpd} &= \frac{1}{2}\sum\nolimits_{\Omega}(x_{i_1,\dots,i_N} - \hat{x}_{i_1,\dots,i_N})^2 \\ &= \frac{1}{2}||\mathbf{X}_{(n)} - \mathbf{A}^{(n)}\mathbf{B}^{(n)\top}||^2,\end{aligned} \tag{2}$$

where Ω is the set contains all possible indices combinations such that $\{(i_1, \dots, i_N) \in \Omega \mid \forall i_n \in [1, I_n], \forall n \in [1, N]\}$ and $\mathbf{B}^{(n)} = \bigodot^{-n}\mathbf{A}^{(n)}$.

Minimizing (2) is non-trivial since it is not convex w.r.t. all loading matrices. As a result, the working horse algorithm [5], ALS, divides (2) into a series of small convex sub-problems. Each sub-problem only optimizes one loading matrix at a time by fixing all others. By doing so, we can iteratively update each $\mathbf{A}^{(n)}, n \in [1, N]$ as

$$\mathbf{A}^{(n)} \leftarrow \mathbf{X}_{(n)}\mathbf{B}^{(n)}(\mathbf{B}^{(n)}{}^\top \mathbf{B}^{(n)})^{-1}.$$

Directly applying ALS to sparse tensors overkills due to its poor efficiency. To address this, Bader and Kolda [8] speed up ALS to $\mathcal{O}(|\Omega^+|R)$ via customizing the unfolding and Khatri-Rao operations for sparse tensors.

In the MV case that zeros represent missing values, such zeros should be omitted since they carry no information and fitting them will only lead the model to a wrong direction. As a result, in order to handle missing values, weighted CPD is introduced by Acar *et al.* [11] as

$$\mathcal{L}_{wcpd} = \frac{1}{2}\sum\nolimits_{\Omega} w_{i_1,\ldots,i_N}(x_{i_1,\ldots,i_N} - \hat{x}_{i_1,\ldots,i_N})^2, \quad (3)$$

$$w_{i_1,\ldots,i_N} = \begin{cases} 0 & \text{if } x_{i_1,\ldots,i_N} \text{ is missing,} \\ 1 & \text{otherwise,} \end{cases} \quad (4)$$

where $w_{i_1,\ldots,i_N}$ is the weight assigned to the $(i_1, \ldots, i_N)$-th entry's approximation.

To minimize the loss function, Acar *et al.* treats this as an optimization problem and all parameters in loading matrices are stacked as a parameter vector, which can be simultaneously optimized by solvers such as Nonlinear Conjugate Gradient. Similar to ALS, this Weighted OPTimization (WOPT) strategy has $\mathcal{O}(|\Omega^+|R)$ time complexity.

The II case is well studied in recommender systems with implicit feedbacks by MF [20], [25], [26]. Under such circumstances, the zeros cannot be simply ignored as missing values, nor be equally treated as observations, since they somehow measure implicit information such as dislike. One popular approach for this type of data is to assign different weights to zero and non-zero entries, so that the contribution of implicit information is leveraged.

Inspired by this, Hidasi and Tikk [12] proposed iTALS algorithm, which extends the weighted CPD framework to the II case by modifying the weighting schema (4) as

$$w_{i_1,\ldots,i_N} = \begin{cases} 1 & \text{if } x_{i_1,\ldots,i_N} = 0, \\ \alpha \cdot \#(i_1, \ldots, i_N) > 1 & \text{otherwise,} \end{cases}$$

where α is a parameter to control the difference of weights between non-zeros and zeros, and $\#(i_1, \ldots, i_N)$ is the number of event tuples corresponding to entry $(i_1, \ldots, i_N)$.

Hidasi and Tikk optimize this implicit CPD by row-wise ALS, which shares the same principle as typical ALS but just optimizes one row at a time as

$$\mathbf{a}^{(n)}_{i_n\cdot} \leftarrow \mathbf{x}_{(n)_{i_n\cdot}}\mathbf{W}^{(i_n)}\mathbf{B}^{(n)}(\mathbf{B}^{(n)}{}^\top \mathbf{W}^{(i_n)}\mathbf{B}^{(n)})^{-1},$$

where $\mathbf{x}_{(n)_{i_n\cdot}}$ is the i_n-th row of the mode-n unfolding $\mathbf{X}_{(n)}$, $\mathbf{W}^{(i_n)}$ is a matrix with all weights related to $\mathbf{x}_{(n)_{i_n\cdot}}$ on its diagonal. The major disadvantage of iTALS is its inefficiency, as a $R \times R$ matrix, $\mathbf{B}^{(n)}{}^\top \mathbf{W}^{(i_n)}\mathbf{B}^{(n)}$, has to be calculated for each individual rows at a cost of $\mathcal{O}(|\Omega^+_{i_n}|R^2)$, which results in an overall complexity as $\mathcal{O}(|\Omega^+|R^2)$.

4. Our Approach

In this section, we introduce our approach for finding the CPD for sparse tensors in general. We first show how to model all special cases, TO, MV and II, under the weighted CPD framework by modifying the weighting schema. Then a highly efficient algorithm is proposed, which is also able to handle constraints such as non-negativity and sparseness. Lastly, we show that our algorithm is applicable to dynamic environment where new entries arrive at high velocity.

4.1. A General Weighting Schema

Under the weighted CPD framework, TO case can be easily modeled by assigning the weight of zero entries to 1. Additionally, the weight of zeros in II case is always larger than 0, but smaller than the weight of non-zeros. Based on these, we proposed the following weighting schema that models all these three cases:

$$w_{i_1,\ldots,i_N} = \begin{cases} \alpha \in [0,1] & \text{if } x_{i_1,\ldots,i_N} = 0, \\ 1 & \text{otherwise,} \end{cases} \quad (5)$$

where α is the weight of zero entries. When $\alpha = 1$, this is equivalent to TO case; it reduces to MV case by letting $\alpha = 0$; and II case is also included in our proposed schema by choosing $\alpha \in (0, 1)$.

4.2. Derivation of SCED

Our algorithm is an generalization of elementwise ALS (eALS) [21] to sparse tensors. Its principal is similar to standard ALS. The key difference is that in eALS, only one parameter (one cell in a loading matrix) is updated at a time, while in ALS, each time one loading matrix is estimated as a whole. The advantage of using a finer-grain update strategy is that it provides the freedom to choose the desired updating sequence without scarifying effectiveness and efficiency. For example, one can update a fraction of cells in one loading matrix at first, then jump to the estimation of another part of parameters in other loading matrices. This is essential for dynamic updating, which will be discussed in §4.4.

Specifically, let $j_n = (i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N)$, $b^{(n)}_{j_n r} = \prod_{\tilde{n} \neq n}^{N} a^{(\tilde{n})}_{i_{\tilde{n}} r}$, we can rewrite (1) and (3) as

$$\begin{aligned} \hat{x}_{i_n j_n} &= \sum_{r=1}^{R} a^{(n)}_{i_n r} b^{(n)}_{j_n r} = \sum_{\tilde{r} \neq r}^{R} a^{(n)}_{i_n \tilde{r}} b^{(n)}_{j_n \tilde{r}} + a^{(n)}_{i_n r} b^{(n)}_{j_n r} \\ &= \hat{x}^r_{i_n j_n} + a^{(n)}_{i_n r} b^{(n)}_{j_n r}, \\ \mathcal{L}_{wcpd} &= \frac{1}{2}\sum\nolimits_{\Omega} w_{i_n j_n}(x_{i_n j_n} - \hat{x}_{i_n j_n})^2 \\ &= \frac{1}{2}\sum\nolimits_{\Omega} w_{i_n j_n}(x_{i_n j_n} - \hat{x}^r_{i_n j_n} - a^{(n)}_{i_n r} b^{(n)}_{j_n r})^2. \end{aligned}$$

The partial derivative of $\mathcal{L}_{wcpd}$ w.r.t. the (i_n, r)-th parameter of loading matrix $\mathbf{A}^{(n)}$ is

$$\frac{\partial \mathcal{L}_{wcpd}}{\partial a^{(n)}_{i_n r}} = -\sum\nolimits_{\Omega_{i_n}} w_{i_n j_n}(x_{i_n j_n} - \hat{x}^r_{i_n j_n} - a^{(n)}_{i_n r} b^{(n)}_{j_n r}) b^{(n)}_{j_n r}$$

$$a_{i_n r}^{(n)} \leftarrow \frac{\sum_{\Omega_{i_n}^{+}} (x_{i_n j_n} - (1-\alpha)\hat{x}_{i_n j_n}^{r}) b_{j_n r}^{(n)} - \alpha(\mathbf{a}_{i_n \cdot}^{(n)} \mathbf{q}_{\cdot r}^{(n)} - a_{i_n r}^{(n)} q_{rr}^{(n)})}{(1-\alpha)\sum_{\Omega_{i_n}^{+}} (b_{j_n r}^{(n)})^2 + \alpha q_{rr}^{(n)}} \tag{8}$$

By setting the derivative to 0, then we reach the closed form solution for $a_{i_n r}^{(n)}$ as

$$a_{i_n r}^{(n)} \leftarrow \frac{\sum_{\Omega_{i_n}} w_{i_n j_n} (x_{i_n j_n} - \hat{x}_{i_n j_n}^{r}) b_{j_n r}^{(n)}}{\sum_{\Omega_{i_n}} w_{i_n j_n} (b_{j_n r}^{(n)})^2}. \tag{6}$$

By applying the weighting schema (5) to (6) we have

$$a_{i_n r}^{(n)} \leftarrow \frac{\sum_{\Omega_{i_n}^{+}} (x_{i_n j_n} - \hat{x}_{i_n j_n}^{r}) b_{j_n r}^{(n)} - \alpha \sum_{\Omega_{i_n}^{-}} \hat{x}_{i_n j_n}^{r} b_{j_n r}^{(n)}}{\sum_{\Omega_{i_n}^{+}} (b_{j_n r}^{(n)})^2 + \alpha \sum_{\Omega_{i_n}^{-}} (b_{j_n r}^{(n)})^2}.$$

So far there are four summations to do for updating $a_{i_n r}^{(n)}$. The left two are related to non-zero entries only, which can be calculated efficiently as $|\Omega_{i_n}^{+}| \ll |\Omega_{i_n}|$. While the other two have to iterate over $\Omega_{i_n}^{-}$, which contains indices for all zeros in the i_n-th row of $\mathbf{X}_{(n)}$. In the following we show that such access to all zero entries is unnecessary and avoidable.

Since $\Omega_{i_n}^{+} \cup \Omega_{i_n}^{-} = \Omega_{i_n}$ we can transform the zero entries related summations as

$$\sum_{\Omega_{i_n}^{-}} \hat{x}_{i_n j_n}^{r} b_{j_n r}^{(n)} = \sum_{\Omega_{i_n}} \hat{x}_{i_n j_n}^{r} b_{j_n r}^{(n)} - \sum_{\Omega_{i_n}^{+}} \hat{x}_{i_n j_n}^{r} b_{j_n r}^{(n)},$$
$$\sum_{\Omega_{i_n}^{-}} (b_{j_n r}^{(n)})^2 = \sum_{\Omega_{i_n}} (b_{j_n r}^{(n)})^2 - \sum_{\Omega_{i_n}^{+}} (b_{j_n r}^{(n)})^2.$$

Again, the non-zero related summations can be readily obtained, and the major concern is how to efficiently get the terms related to Ω_{i_n}. Let $\mathbf{Q}^{(n)} = \circledast^{-n} {\mathbf{A}^{(n)}}^{\top} \mathbf{A}^{(n)}$ [5], one can easily verify that $\sum_{\Omega_{i_n}} (b_{j_n r}^{(n)})^2 = q_{rr}^{(n)}$. Additionally, recall that $\hat{x}_{i_n j_n}^{r} = x_{i_n j_n} - a_{i_n r}^{(n)} b_{j_n r}^{(n)}$, $\sum_{\Omega_{i_n}} \hat{x}_{i_n j_n}^{r} b_{j_n r}^{(n)}$ can be rewritten as

$$\begin{aligned}
&\sum_{\Omega_{i_n}} \Big(\sum_{r=1}^{R} a_{i_n r}^{(n)} b_{j_n r}^{(n)} - a_{i_n r}^{(n)} b_{j_n r}^{(n)}\Big) b_{j_n r}^{(n)} \\
&= \sum_{r=1}^{R} a_{i_n r}^{(n)} \sum_{\Omega_{i_n}} (b_{j_n r}^{(n)})^2 - a_{i_n r}^{(n)} \sum_{\Omega_{i_n}} (b_{j_n r}^{(n)})^2 \\
&= \mathbf{a}_{i_n \cdot}^{(n)} \mathbf{q}_{\cdot r}^{(n)} - a_{i_n r}^{(n)} q_{rr}^{(n)},
\end{aligned} \tag{7}$$

which can be efficiently computed in $\mathcal{O}(R)$ time.

Therefore, if $\mathbf{Q}^{(n)}$ is known, the complexity to update $a_{i_n r}^{(n)}$ is only related to $|\Omega_{i_n}^{+}|$. In addition, a list of auxiliary matrices $\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}$ where $\mathbf{U}^{(n)} = {\mathbf{A}^{(n)}}^{\top} \mathbf{A}^{(n)}, n \in [1, N]$ can be calculated and cached in advance, in order to speed up the computing of $\mathbf{Q}^{(n)}$, which takes $\mathcal{O}(NR^2)$ time. And only the n-th auxiliary matrix $\mathbf{U}^{(n)}$ need to be re-computed after updating $\mathbf{A}^{(n)}$.

Overall, by putting everything together, we reach the final update rule as (8), and the proposed SCED algorithm is summarized in Algorithm 1.

Algorithm 1: SCED Algorithm

Input: Input tensor $\mathfrak{X}$, decomposition rank R, weight for zeros α
Output: Loading matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$

Randomly initialize $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$
for $(i_1, \ldots, i_N) \in \Omega^{+}$ **do** $\hat{x}_{i_1,\ldots,i_N} \leftarrow$ Eq. (1)
for $n \leftarrow 1$ **to** N **do** $\mathbf{U}^{(n)} = {\mathbf{A}^{(n)}}^{\top} \mathbf{A}^{(n)}$
while *stopping criteria is not met* **do**
  for $n \leftarrow 1$ **to** N **do**
    $\mathbf{Q}^{(n)} \leftarrow \circledast^{-n} \mathbf{U}^{(n)}$
    for $i_n \leftarrow 1$ **to** I_n **do**
      for $(i_n, j_n) \in \Omega_{i_n}^{+}$ **do** $b_{j_n \cdot}^{(n)} = \prod_{\tilde{n} \neq n}^{N} a_{i_{\tilde{n}} \cdot}^{(\tilde{n})}$
      for $r \leftarrow 1$ **to** R **do**
        $\hat{\mathbf{x}}_{i_n \cdot}^{r} = \hat{\mathbf{x}}_{i_n \cdot} - a_{i_n r}^{(n)} {\mathbf{b}_{\cdot r}^{(n)}}^{\top}$
        $a_{i_n r}^{(n)} \leftarrow$ Eq. (8)
        $\hat{\mathbf{x}}_{i_n \cdot} = \hat{\mathbf{x}}_{i_n \cdot}^{r} + a_{i_n r}^{(n)} {\mathbf{b}_{\cdot r}^{(n)}}^{\top}$
      end
    end
    $\mathbf{U}^{(n)} = {\mathbf{A}^{(n)}}^{\top} \mathbf{A}^{(n)}$
  end
end

4.3. Incorporating Constraints

Constraints are commonly used in real-world decompositions. Here we mainly focus on two types of constraints: non-negativity and regularizations, due to their popularity.

4.3.1. Non-negativity. Non-negativity is a widely used constraint in CPD to enforce part-based solutions that are usually more interpretable. We handle this constraint by applying a simple "half-wave rectifying" [17] nonlinear projection in addition to each updating. Specifically, after getting an updated value by (8), we keep it if it is greater than 0; otherwise, the updated value is replaced by 0 (or a small number such 1×10^{-9} for numerical stableness). The correctness of this is supported by the following theorem:

Theorem 1. *The minimization problem*

$$\min_{a_{i_n r}^{(n)} \geq 0} \mathcal{L}_{wcpd}$$

has the unique solution as

$$a_{i_n r}^{(n)} = \left[\frac{\sum_{\Omega_{i_n}} w_{i_n j_n} (x_{i_n j_n} - \hat{x}_{i_n j_n}^{r}) b_{j_n r}^{(n)}}{\sum_{\Omega_{i_n}} w_{i_n j_n} (b_{j_n r}^{(n)})^2} \right]_{+}$$

where $[z]_{+} = \mathtt{max}(0, z)$.

Proof. This can be proved in similar way as Kim *et al.* [28]. We can organize the derivative of $\mathcal{L}_{wcpd}$ w.r.t. $a^{(n)}_{i_n r}$ as

$$\frac{\partial \mathcal{L}_{wcpd}}{\partial a^{(n)}_{i_n r}} = a^{(n)}_{i_n r} \cdot slope + intercept,$$

where $intercept = -\sum_{\Omega_{i_n}} w_{i_n j_n}(x_{i_n j_n} - \hat{x}^r_{i_n j_n}) b^{(n)}_{j_n r}$ and $slope = \sum_{\Omega_{i_n}} w_{i_n j_n}(b^{(n)}_{j_n r})^2$. If $intercept \leq 0$, it is clear that $\mathcal{L}_{wcpd}$ reaches its minimum at $a^{(n)}_{i_n r} = -\frac{intercept}{slope}$, where the derivative is 0; if $intercept > 0$, the loss, $\mathcal{L}_{wcpd}$, increases as $a^{(n)}_{i_n r}$ become larger than 0. Thus, the minimum is attained at $a^{(n)}_{i_n r} = 0$. As a result, combining both cases, the solution can be expressed as $a^{(n)}_{i_n r} = [-\frac{intercept}{slope}]_+$ $\square$

4.3.2. Regularization. A common strategy to take constraints into considerations is treating them as regularizations to the original optimization problem as

$$\min_{\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}} \mathcal{L}_{wcpd} + \sum_{n=1}^{N} \lambda_n \phi_n(\mathbf{A}^{(n)}), \tag{9}$$

where λ_n is a non-negative regularization parameter and ϕ_n is the regularizing function applied to the n-th loading matrix. Depending on constraints, different ϕ can be used. For example, if $\phi_n(\mathbf{A}^{(n)}) = \frac{1}{2}||\mathbf{A}^{(n)}||^2$, Tikhonov regularization is used to prevent overfitting; and $\phi_n(\mathbf{A}^{(n)}) = \sum_{i_n=1}^{I_n} ||\mathbf{a}^{(n)}_{i_n \cdot}||_1$ is another widely used regularization to promote sparseness in solution.

Above regularized optimization problem (9) can be easily solved by the proposed SCED algorithm with similar derivation in §4.2. Due to the page limit, here we directly give the closed form solutions based on (6), and of course similar efficient version as (8) can be derived

$$a^{(n)}_{i_n r} \leftarrow \begin{cases} \frac{\sum_{\Omega_{i_n}} w_{i_n j_n}(x_{i_n j_n} - \hat{x}^r_{i_n j_n}) b^{(n)}_{j_n r}}{\sum_{\Omega_{i_n}} w_{i_n j_n}(b^{(n)}_{j_n r})^2 + \lambda_n} & l_2\text{-norm}, \\ \frac{\sum_{\Omega_{i_n}} w_{i_n j_n}(x_{i_n j_n} - \hat{x}^r_{i_n j_n}) b^{(n)}_{j_n r} + \lambda_n \cdot \mathtt{sign}(a^{(n)}_{i_n r})}{\sum_{\Omega_{i_n}} w_{i_n j_n}(b^{(n)}_{j_n r})^2} & l_1\text{-norm}. \end{cases}$$

4.4. Dynamic Learning

In real-world applications, it is not uncommon to see that after decomposing a tensor, new data will keep arriving at a high speed. A method that can efficiently track the new decompositions in such dynamic scenario is desired, since the static model may not perform well because it is not up-to-date.

This problem has been extensively studied for matrix cases and a common assumption is that the new data will only has considerable impact to the local features, while the global model will not be affected significantly. For example, giving a matrix $\mathbf{X} \in \mathbb{R}^{M \times N}$ and its decomposition $\mathbf{W} \in \mathbb{R}^{M \times R}$ and $\mathbf{H} \in \mathbb{R}^{N \times R}$, in order to learn a new interaction x_{mn}, only the m-th and n-th rows of $\mathbf{W}$ and $\mathbf{H}$ will be updated, while other parameters are remaining unchanged.

Algorithm 2: SCED for Dynamic Learning

Input: Existing loading matrices $\mathring{\mathbf{A}}^{(1)}, \ldots, \mathring{\mathbf{A}}^{(N)}$, new interaction $x_{i_1,\ldots,i_N}$
Output: Updated matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$

for $n \leftarrow 1$ **to** N **do**
  $\mathbf{A}^{(n)} \leftarrow \mathring{\mathbf{A}}^{(n)}$
  if $\mathbf{a}^{(n)}_{i_n \cdot}$ *not exists* **then** random initialize $\mathbf{a}^{(n)}_{i_n \cdot}$
end
$\hat{x}_{i_1,\ldots,i_N} \leftarrow$ Eq. (1)
while *stopping criteria is not met* **do**
  for $n \leftarrow 1$ **to** N **do**
    $\mathring{\mathbf{U}} \leftarrow \mathbf{a}^{(n)\top}_{i_n \cdot} \mathbf{a}^{(n)}_{i_n \cdot}$
    update $\mathbf{a}^{(n)}_{i_n \cdot}$ */* line 8-13 of Algorithm 1 */*
    $\mathbf{U} \leftarrow \mathbf{a}^{(n)\top}_{i_n \cdot} \mathbf{a}^{(n)}_{i_n \cdot}$
    $\mathbf{U}^{(n)} = \mathbf{U}^{(n)} - \mathring{\mathbf{U}} + \mathbf{U}$ */* update cache */*
  end
end

TABLE 3. TIME COMPLEXITY

	SCED	Baseline
TO ($\alpha = 1$) & MV ($\alpha = 0$)	$\mathcal{O}(\lvert\Omega^+\rvert R)$	$\mathcal{O}(\lvert\Omega^+\rvert R)$
II ($\alpha \in (0, 1)$)	$\mathcal{O}(\lvert\Omega^+\rvert R)$	$\mathcal{O}(\lvert\Omega^+\rvert R^2)$

Since CPD is a generalization of MF for multi-way data, similar vector retaining strategy can be used for dynamically learning new incoming interactions on tensorial data and we summarize the dynamic version of SCED in Algorithms 2.

4.5. Complexity

Here we briefly give a time complexity analysis and the comparison with existing methods can be found in Table 3. ALS, WOPT and iTALS are chosen baselines that specifically target on TO, MV and II cases, respectively.

For SCED, as shown in Algorithm 1, to update one loading matrix, $\mathbf{Q}^{(n)}$ can be calculated in $(N-1)R^2$ operations (line 6) and for each row in $\mathbf{A}^{(n)}$, a $|\Omega^+_{i_n}| \times R$ matrix $\mathbf{B}^{(n)}$ is generated (line 8). Then each element $a^{(n)}_{i_n r}$ is updated at a cost of $\mathcal{O}(|\Omega^+_{i_n}| + R)$ (line 10-12). In total, the time cost for updating $\mathbf{A}^{(n)}$ is $\mathcal{O}(|\Omega^+|R + I_n R^2)$. This procedure is repeated for all loading matrices and takes $\mathcal{O}(|\Omega^+|NR + \sum_{n=1}^{N} I_n R^2)$ operations for one iteration, which is dominant by $\mathcal{O}(|\Omega^+|R)$. In summary, for TO and MV cases, the proposed method shares same efficiency as state-of-the-arts, while our method is R times faster than iTALS for II case.

Similar to the static case, the complexity of dynamic SCED is dominant by $\mathcal{O}(|\Omega^+_{i_n}|R)$. While in general, the dynamic case is around I times faster than batch case, where $I = \mathtt{mean}(I_1, \ldots, I_N)$, since only one row is updated for each loading matrix.

5. Empirical Analysis

In this section, we evaluate the performance of our SCED algorithm with state-of-the-arts in terms of effectiveness and efficiency. The performance is evaluated on both synthetic and real-world datasets. For each dataset, both static and dynamic settings are tested. After that, based on the investigation on large-scale synthetic tensors, we further analyze the scalability of our approach.

5.1. Experiment Specifications

5.1.1. Datasets. The first half of the experiments are conducted on three $200 \times 200 \times 200$ synthetic datasets: SYN-TO, SYN-MV and SYN-II. All of them have rank as 20 and density as 1%. The key difference among them is the role of zero entries. SYN-TO is generated by sparse loading matrices such that its zeros are true observations. In contrast, the zeros in SYN-MV are missing values that are randomly sampled from a dense tensor, which is generated by random loading matrices. Since it is non-trivial to simulate the implicit feedbacks, while the weight of zeros in such situation lies between the weights of TO and MV cases, we create SYN-II by mixing the data generation protocols of SYN-TO and SYN-MV. Specifically, less sparse loading matrices are generated to form a tensor with around 50% non-zeros, from which 1% values are randomly sampled as the SYN-II.

In addition, to better evaluate the performance of our algorithm in real-world applications, three datasets of varying characteristics have been used: MovieLens[1], LastFM[2] and MathOverflow[3]. Their detail can be found in Table 4.

TABLE 4. DETAIL OF REAL-WORLD DATASETS

datasets	Size	nnz^*	Density	Source
MovieLens	$6040 \times 3952 \times 1040$	1×10^6	4.03×10^{-5}	[29]
LastFM	$991 \times 1000 \times 168$	2.9×10^6	1.73×10^{-2}	[30]
MathOverflow	$24818 \times 24818 \times 2351$	4×10^5	2.75×10^{-7}	[31]

* number of non-zeros

5.1.2. Baselines. Four baselines have been selected as the competitors to evaluate the performance in our experiment

(i) ALS: an implementation for sparse tensors from Tensor Toolbox [32].

(ii) MU: a multiplicative update rule based algorithm for non-negative CPD from Tensor Toolbox [32].

(iii) WOPT [11]: an algorithm for decomposing incomplete tensors based on weighted optimization.

(iv) iTALS [12]: an approach that decomposes sparse tensors that represent implicit feedbacks.

It should be noted that all these baselines are batch methods and there is no existing work that can be directly used for dynamic updating on sparse tensors. However, since iTALS has a row-wise update rule, we modify it under the same vector-retaining model as our method, as a baseline that is able to perform dynamic learning.

1. https://movielens.org
2. https://www.last.fm
3. https://mathoverflow.net

5.1.3. Evaluation Metrics. The empirical performance is measured from both effectiveness and efficiency aspects.

In terms of effectiveness, for synthetic datasets, because the ground truth loading matrices are known already, *fitness* is used and defined as

$$fitness \triangleq \left(1 - \frac{\|\hat{\mathcal{X}} - \mathcal{X}\|}{\|\mathcal{X}\|}\right),$$

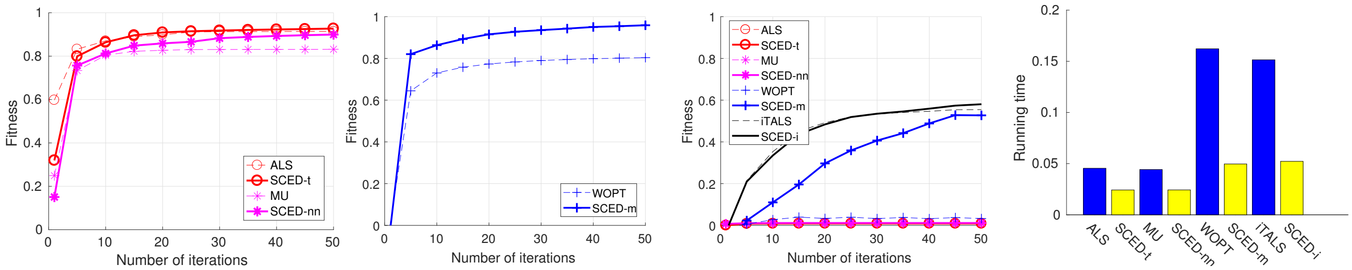
where $\mathcal{X}$ is the tensor formed by ground truth, $\hat{\mathcal{X}}$ is the estimation and $\|\bullet\|$ denotes the Frobenius norm. The closer the fitness to 1, the better decomposition we got. However, for each real-world dataset, a test set (10%) is sampled and Root Mean Square Error (RMSE) is used for measuring the decomposition quality, since the ground truth is not given. The lower RMSE, the better result.

In addition, with respect to efficiency, for static decomposition, the average *running time* for one iteration, measured in seconds, is reported to validate the time efficiency of an algorithm. On the other hand, the running time for processing one new entry are recorded to compare the efficiency under the dynamic setting.

5.1.4. Experimental Setup. For both synthetic and real-world datasets, there are mainly two parts of experiments have been conducted for performance evaluation: static and dynamic settings.

Static setting: given a data tensor, it is decomposed by each algorithm with the same random initialization. It should be noted that for real-world tensors, 10% of observations are randomly sampled and hold out as test set for RMSE calculation. In terms of experimental parameters, 20 has been used as the decomposition rank over all experiments, since we are not aiming at finding the best decomposition, but more interested in the relative performance between our proposal and baselines. For synthetic datasets, the maximum number of iterations is set to 50. While for real-world datasets, this has been set to 10 due to low efficiency of iTALS and WOPT.

Dynamic setting: 10% of observations are randomly sampled as dynamic set, such that each dataset is divided into different parts: 90% training, 10% dynamic for synthetic datasets, and 80% training, 10% dynamic and 10% testing for real-world datasets. The training set is decomposed by the batch algorithm with different random initializations, and the best result is chosen as the base for dynamic learning. In the dynamic updating phase, at each time stamp, an entry from dynamic set is randomly selected and processed by both our SCED algorithm and its competitors. Specifically, for batch baselines (ALS and WOPT), previous decomposition is used as a hot start and only one iteration is performed to process the new data. After that, effectiveness (fitness for synthetic, RMSE for real-world datasets) and efficiency (running time in seconds) metrics are recorded for performance demonstration.



(a) TO (b) MV (c) II (d) Efficiency

Figure 2. Performance of static decomposition on synthetic datasets

Another difference between synthetic and real-world experiments is the choice of baselines. For synthetic datasets, since the role of zeros are known, only algorithms that specifically target on it are used for comparison. For example, ALS and MU are selected as baselines for SYN-TO, compared to our SCED variants, SCED-t and SCED-nn; while the comparison is only conducted between WOPT and SCED-m for SYN-MV. Conversely, for SYN-II, all algorithms are used for experiment, in order to show the merits of assigning small weight to zero entries under such circumstance. Similarly, for real-world datasets, all baselines have been used for comparison in the static experiment. While under the dynamic setting, only SCED-i and iTALS are used for dynamic learning, because of their relative good performance in static case, and the low efficiency of other batch baselines.

In terms of algorithm-specified parameters, apart from the aforementioned parameters (rank and maximum number of iterations), there is no parameter need to be tuned for ALS and MU. For WOPT, default parameters have been used for its internal line search procedure. Regarded to SCED-i and iTALS, the weight of zeros (α) is set as the density of each dataset, in order to balance the contribution of non-zero and zero entries. This is just a heuristic way and of course it can be fine tuned by cross validation. While since the goal is to see the relative performance among different algorithms, we chose to settle on this strategy for our experiment. Lastly, there is no regularization has been used in SCED, for a fair comparison, i.e., $\lambda_n = 0, n \in [1, N]$.

For each dataset, all experiments are replicated 10 times on a desktop with Intel i7 processors, 16 GB RAM and Matlab 2016b. The reported results are averaged over these 10 runs.

5.2. On Syntheic datasets

5.2.1. Static Results. The performance of decomposing synthetic tensors under the static setting is presented in Figure 2. Since the efficiency is only related to the number of non-zeros and has no linkage to the types of data, we summarize all efficiency result into Figure 2d.

As can be seen from the figures, in most of cases, our proposal, SCED (denoted by solid lines), shows better decomposition quality, compared to baselines (denoted by dashed lines). Specifically, for SYN-TO and SYN-MV datasets, even though baselines yield acceptable results, significant improvements can be found in our algorithm. For SYN-II dataset, all methods that treat zeros as equally important as observations work very poor. In contrast, slightly better performance can be found in WOPT, which ignores all distractions from zero entries. Even this, it is still beaten by our approach by a very large margin, where SCED-m achieves around 0.5 in fitness score, while the fitness of WOPT is lower than 0.1. Both SCED-i and iTALS show similar performance to each other. However, it is clear that iTALS is much slower than our proposal.

In terms of efficiency, ALS and MU share similar performance to each other, while they are nearly twice slower than our algorithm. Significant time consumption can be observed in WOPT, due to its internal line search procedure, which makes it less appealing compared to our algorithm.

5.2.2. Dynamic Results. In this part of experiment, the training set is decomposed by the batch algorithm for initialization: ALS for SYN-TO, WOPT for SYN-MV and batch SCED-i for SYN-II datasets, respectively. With the best seed, the dynamic set is learned and the results are reported in Figure 3.

It is clear that for SYN-TO dataset, our proposed method achieves perfect overlapping with ALS, which is a batch algorithm that does optimization on all observations to time. This verifies the usefulness of the proposed dynamic updating schema. In terms of efficiency, on average, our method speeds up the processing time for one new entry by around 170 times (~ 0.09 second in ALS v.s. $\sim 5 \times 10^{-4}$ second in SCED-t).

Even greater speed-up can be observed for SYN-MV, where our algorithm ($\sim 5.5 \times 10^{-4}$ second) is more than 1400 times faster than the batch baseline, WOPT (~ 0.8 second). More importantly, better decomposition quality is also shown in our algorithm, which confirmed the superior performance of our algorithm, compared to WOPT, in both static and dynamic settings.

With respect to SYN-II dataset, both SCED-i and iTALS are capable to dynamically track the new CPD when new entries fed in. Similar to SYN-TO, perfect overlapping can be found while our method is three times faster than iTALS.

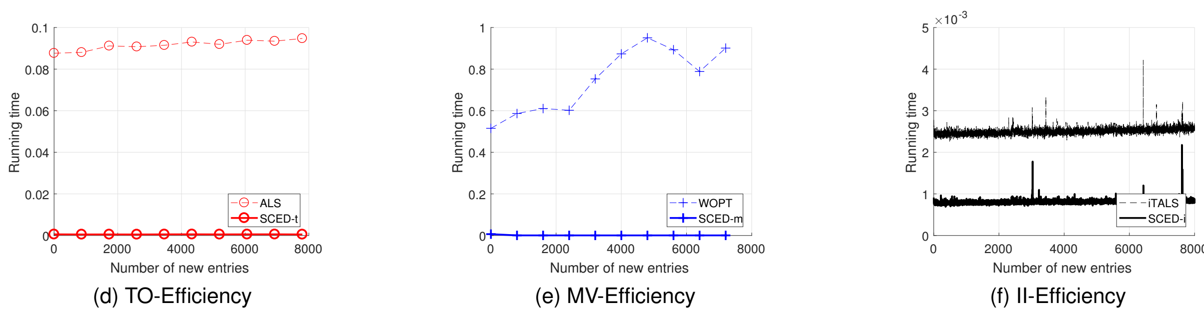


Figure 3. Dynamic decomposition on synthetic datasets

5.3. On Real-world datasets

5.3.1. Static Results. The performance comparison on real-world datasets can be found in Figure 4. Among all datasets, the smallest RMSEs are always obtained by SCED-i and iTALS, which means that giving small weight to zero entries is more likely to get better understanding of data, compared to the other two extreme cases that either totally ignoring them or treating them equally as observations. TO-based methods ($\alpha = 1$) produce poor results in general. However, within this group, we still see better results from our algorithm. For example, one can find that SCED-nn shows much better convergence, compared to its competitor for the same task, MU. Similar to the results on synthetic datasets, significant improvement can be seen between SCED-m and WOPT in MovieLens and LastFM. On the other hand, we notice the considerable performance drop of them on MathOverflow. One reason behind this might be that their optimizations focus on non-zeros only, while MathOverflow is much more sparse than other two datasets. As a result, the CPDs produced by them will greatly bias to known entries, which can be considered as overfitting. One possible solution to address this issue is to add regularizations into the objectives, which can be easily handled by our proposal (e.g., set λ to 0.1 with ϕ as l_2-norm), while there is no clear way to adopt WOPT for this case.

5.3.2. Dynamic Results. As mentioned in experiment setup, only SCED-i and iTALS has been chosen for dynamic updating for real-world datasets, due to their good static performance and the inefficiency of other batch methods for large-scale datasets. We report the performance comparison in Figure 5 and Table 5. Additionally, to validate whether such dynamic learning truly effective, we also decompose each tensor with batch SCED-i algorithm before (80% training) and after (80% training + 10% dynamic) the dynamic learning phase, as reference points. Specifically, the maximum number of iterations for batch method is set to 200 and the decomposition generated before feeding the dynamic set is used as the seed for hot start.

TABLE 5. AVERAGE RUNNING TIME TO PROCESS ONE NEW ENTRY ON REAL-WORLD DATASETS. VALUES IN PARENTHESES ARE NUMBER OF ENTRIES CAN BE PROCESSED PER MINUTES

dataset	iTALS	SCED-i
MovieLens	0.0255 (2356)	0.0094 (6359)
LastFM	0.0484 (1241)	0.0185 (3244)
MathOverflow	0.0091 (6581)	0.0081 (7408)

As can be seen from Figure 5, both SCED-i and iTALS can effectively tracking the decompositions when new data arrives, while our algorithm is around 3 times faster than iTALS. In terms of efficiency of batch method, it takes 320, 475 and 600 seconds to decompose the training sets of MovieLens, LastFM and MathOverflow, respectively. And roughly speaking, extra 30 seconds is needed for processing the additional dynamic sets. Such high expense makes batch method infeasible to be applied to a highly dynamic system that new data is always arriving at high velocity. Unlike batch algorithm that can only process one state of data at high time cost, dynamic algorithms can keep tracking decompositions of all intermediate states at significantly lower cost. This means the proposed algorithm can efficiently and easily refresh the decomposition model to date, therefore, to provide better service than batch techniques, where the most up-to-date decomposition is not always available.

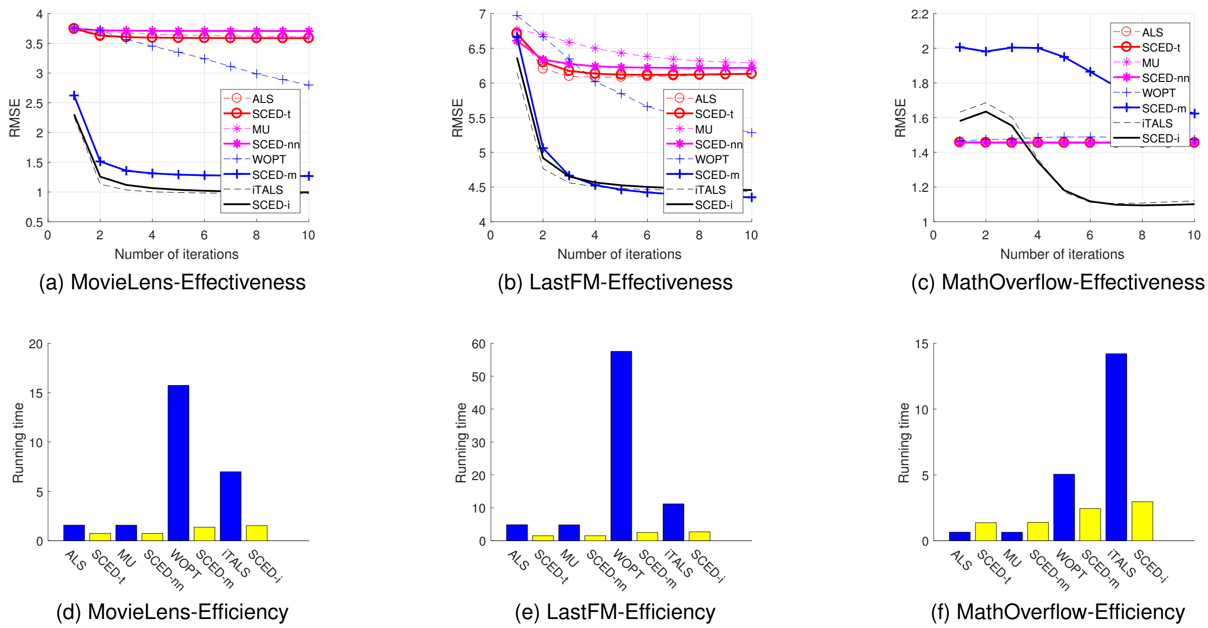


Figure 4. Static decomposition on real-world datasets

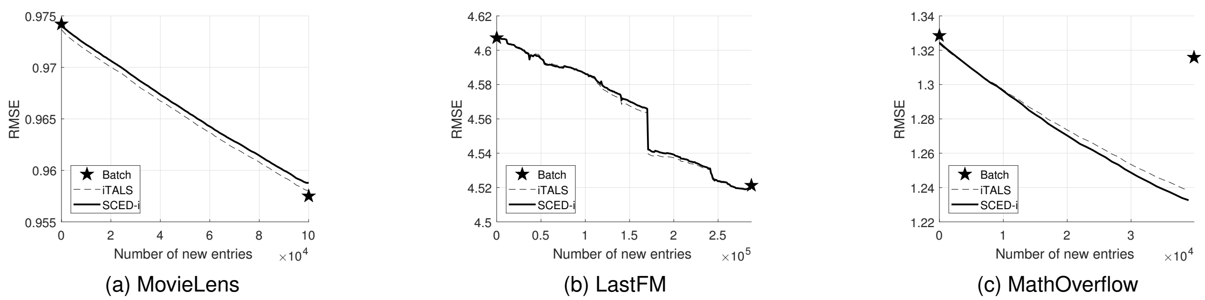


Figure 5. Dynamic decomposition on real-world datasets

5.4. Scalability

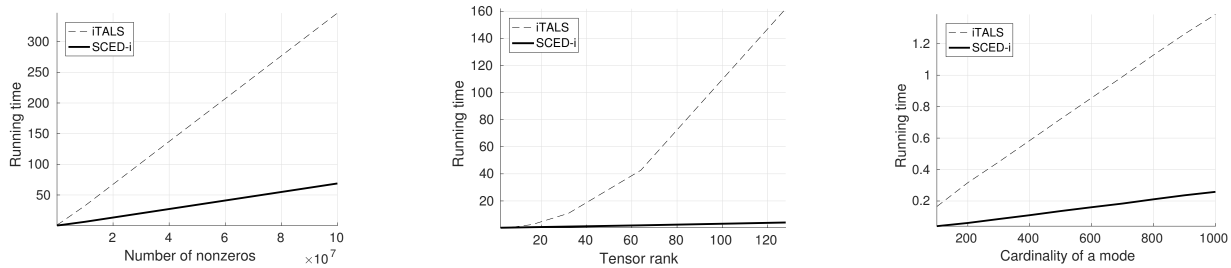
As shown in §4.5, theoretically, SCED is R times faster than iTALS. Previous experiments partially confirm this analysis. However, the observed speed-up is only around 3 to 5 times. This is because that R has been set to 20, which is too small to see the trend. As a result, to evaluate the performance of SCED and iTALS on large scale datasets, scalability test is performed w.r.t. three key features: number of non-zeros, decomposition rank and tensor size.

Specifically, random tensors of size $1000 \times 1000 \times 1000$ with $R = 20$ are decomposed and the result can be found in Figure 6a. The number of non-zeros in them varies from 1×10^4 to 1×10^8. Similar evaluation has been done for analyzing the scalability w.r.t. R (Figure 6b, R varies from 2 to 128, by fixing $nnz = 1 \times 10^4$ and size as $1000 \times 1000 \times 1000$) and the tensor size (Figure 6c, cardinality of a mode varies from 100 to 1000, with $R = 20$ and $nnz = 1 \times 10^4$). The reported results are average running time for one iteration under the batch setting, while it is clear that similar trends can be seen for the dynamic case.

5.5. Highlights of Results

To make a clear summary of the experimental performance, we highlight some key findings as follows.

- The proposed SCED algorithm uniformly produces best or close-to-best quality decompositions on different types of data, no matter in static or dynamic settings. Two major improvements can be found in SCED-m v.s. WOPT, and SCED-nn v.s. MU.
- The efficiency of SCED is similar to ALS and MU. While our method is significantly faster than WOPT and iTALS, spanning all types of data and settings.
- The proposed dynamic learning method is highly effective and usually demonstrates comparable or even better result to batch methods. While our method reduce time cost by factor of hundreds to thousands.
- Compared to iTALS, our method is more suitable to be applied to large scale data since better scalability is shown w.r.t. number of non-zeros, decomposition rank and tensor size.



(a) $nnz \in [1e4, 1e8]$ (b) $R \in [2, 128]$ (c) $I \in [100, 1000]$

Figure 6. Scalability

6. Conclusions and Future Work

To conclude, in this paper, we address the problem of finding the CPD of sparse tensors. An efficient algorithm, SCED, is proposed, which has linear time complexity w.r.t. the number of non-zeros and decomposition rank. In addition, our framework is also flexible to handle constraints such as non-negativity and regularizations like l_1 and l_2 norms. Last but not the least, a dynamic learning algorithm is also proposed to tackle with dynamic tensors that have new data coming at element-level. As evaluated on both synthetic and real-world datasets, under both static and dynamic settings, our algorithm demonstrate outstanding performance in terms of effectiveness, efficiency and scalability, compared to state-of-the-art algorithms.

Regarding to future work, apparently there is still room for improving our method. One possible direction is to link our work with existing online work, as slice-by-slice online tensors can be considered as a special dynamic case. Another potential area is to further extend our method for more dynamic cases, since currently we only looked at the addition case, while situations such as deletion and modification of existing cells are also of interest and perhaps can be addressed in similar manner as the current proposal.

References

[1] C. Mu, et al., "Square deal: Lower bounds and improved relaxations for tensor recovery." in *ICML*, 2014.

[2] A. Anandkumar, et al., "A tensor approach to learning mixed membership community models." *JMLR*, 2014.

[3] I. Davidson, et al., "Network discovery via constrained tensor analysis of fmri data," in *SIGKDD*, 2013.

[4] G. Ma, et al., "Spatio-temporal tensor analysis for whole-brain fmri classification," in *ICDM*, 2016.

[5] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, 2009.

[6] A. Cichocki, et al., "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Process. Mag.*, 2015.

[7] N. D. Sidiropoulos, et al, "Tensor decomposition for signal processing and machine learning," *arXiv preprint arXiv:1607.01668*, 2016.

[8] B. W. Bader and T. G. Kolda, "Efficient matlab computations with sparse and factored tensors," *SIAM J. Sci. Comput.*, 2007.

[9] U. Kang, et al, "Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries," in *SIGKDD*, 2012.

[10] J. H. Choi and S. Vishwanathan, "Dfacto: Distributed factorization of tensors," in *NIPS*, 2014.

[11] E. Acar, et al., "Scalable tensor factorizations for incomplete data," *Chemometr. Intell. Lab.*, 2011.

[12] B. Hidasi, et al, "Fast als-based tensor factorization for context-aware recommendation from implicit feedback," in *ECML-PKDD*, 2012.

[13] P. Rai, et al, "Scalable bayesian low-rank decomposition of incomplete multiway tensors." in *ICML*, 2014.

[14] Q. Zhao, L. Zhang, and A. Cichocki, "Bayesian cp factorization of incomplete tensors with automatic rank determination," *TPAMI*, 2015.

[15] M. Welling and M. Weber, "Positive tensor factorization," *Pattern Recognition Letters*, 2001.

[16] C.-J. Lin, "Projected gradient methods for nonnegative matrix factorization," *Neural computation*, 2007.

[17] A. Cichocki, et al, "Fast local algorithms for large scale nonnegative matrix and tensor factorizations," *IEICE T FUND ELECTR*, 2009.

[18] S. Hansen, et al, "Newton-based optimization for kullback–leibler nonnegative tensor factorizations," *OPTIM METHOD SOFTW*, 2015.

[19] J. Liu, et al, "Sparse non-negative tensor factorization using columnwise coordinate descent," *PATTERN RECOGN*, 2012.

[20] R. Devooght, et al, "Dynamic matrix factorization with priors on unknown values," in *SIGKDD*, 2015.

[21] X. He, et al, "Fast matrix factorization for online recommendation with implicit feedback," in *SIGIR*, 2016.

[22] S. Zhou, et al, "Accelerating online cp decompositions for higher order tensors," in *SIGKDD*, 2016.

[23] D. Nion, et al, "Adaptive algorithms to track the parafac decomposition of a third-order tensor," *IEEE Trans. Signal Process.*, 2009.

[24] J. Sun, et al, "Incremental tensor analysis: Theory and applications," *TKDD*, 2008.

[25] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *ICDM*, 2008.

[26] I. Pilászy, et al, "Fast als-based matrix factorization for explicit and implicit feedback datasets," in *RecSys*, 2010.

[27] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *NIPS*, 2001.

[28] J. Kim, et al, "Algorithms for nonnegative matrix and tensor factorizations: A unified view based on block coordinate descent framework," *J. Global Optim.*, 2014.

[29] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *TiiS*, 2016.

[30] O. Celma, *Music Recommendation and Discovery in the Long Tail*, 2010.

[31] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks," in *WSDM*, 2017.

[32] B. W. Bader, T. G. Kolda *et al.*, "Matlab tensor toolbox version 2.6," Available online, February 2015.