Distributed Dense Tucker Decomposition Based on Hierarchical SVD

Zisen Fang[1,2], Fumin Qi[1(✉)], Yichuan Dong[1], Yong Zhang[2], and Shengzhong Feng[1]

[1] National Supercomputing Center in Shenzhen, Shenzhen, China
{fangzs, qifm, dongyc, fengsz}@nsccsz.cn
[2] Shenzhen Institutes of Advanced Technology, Chinese Academy of Science, Beijing, China
zhangyong@siat.ac.cn

Abstract. As an important tool of multi-way/tensor data analysis tool, Tucker decomposition has been applied widely in various fields. But traditional sequential Tucker algorithms have been outdated because tensor data is growing rapidly in term of size. To address this problem, in this paper, we focus on parallel Tucker decomposition of dense tensors on distributed-memory systems. The proposed method uses Hierarchical SVD to accelerate the SVD step in traditional sequential algorithms, which usually takes up most computation time. The data distribution strategy is designed to follow the implementation of Hierarchical SVD. We also find that compared with the state-of-the-art method, the proposed method has lower communication cost in large-scale parallel cases under the assumption of the α–β model.

Keywords: Dense tensor · Tucker decomposition · Distributed memory

1 Introduction

In computer science, tensors are N-way arrays. In early days, tensor data was dealt with by vectorization, which broke construction information of original data. With the development of higher order data processing, tensor analysis has drawn lots of attention and shown advantage to vectorization methods. As the most important tool in tensor analysis, tensor decomposition methods are widely applied in multi fields such as computer vision [1, 2], machine learning [3, 4], neuroscience [5], data mining [6], social network computing [7] and so on. There are several tensor decomposition methods, among which Tucker decomposition [8] is one of most famous ones.

Tucker decomposition is traditionally solved by HOOI [9] with quite expensive cost due to iteration progress and TTM (Tensor Times Matrix) chain computation. Since Higher Order Singular Values Decomposition (HOSVD) [10] plays the role of initial step in HOOI, its truncated version Truncated HOSVD (T-HOSVD) can be used to approximate the output of HOOI, with only a little accuracy lost but great reduction of computation cost. An even better approximation is Sequential Truncated HOSVD (ST-HOSVD) [11], which applies greedy strategy to the truncation progress and achieves improvements in both accuracy and speed.

L. Ning et al. (Eds.): PAAP 2020, CCIS 1362, pp. 355–364, 2021.
https://doi.org/10.1007/978-981-16-0010-4_31

Although ST-HOSVD has reduced lots of computation cost compared to traditional methods like HOOI, it's still far from enough. With the development of our society and techniques, data sizes are growing rapidly. Traditional sequential solution within one computation node fails to handle Tucker decomposition of large tensor data today, due to not only extremely long computation time but also memory limit of one single node. To solve Tucker decomposition efficiently, recent years have seen multiple parallel algorithms on this issue.

Based on Hadoop [12], HaTen2 [13] can handle sparse tensor data of billion scale. This method divides complex tensor multiplications into large amount of unrelated vector dot products. To cut the dependencies of original sequential steps, HaTen2 takes additional operators which increases computation cost but improves parallelism a lot. For sparse tensors, this method reduces execution time on clusters with few additional tasks, while it costs much more for nonsparse cases. To deal with dense data, [14] proposes the first distributed dense Tucker decomposition algorithm on CPU clusters. It distributes tensor data across all modes. For example, let $\mathcal{X}$ be a tensor of size $I_1 \times I_2 \times I_3$, and $p = p_1 \times p_2 \times p_3$ be the number of available processors. $\mathcal{X}$ is divided into tensor blocks of size $\frac{I_1}{p_1} \times \frac{I_2}{p_2} \times \frac{I_3}{p_3}$, and each processor owns one block. This distribution strategy has been followed by several works as described below. [15] distributes tensor data using the same strategy and optimizes the TTM chain computation for parallel HOOI. It constructs TTM-trees to present TTM schemes and searches the best one with least computational load. GPUTensor [16] applies the same distribution strategy of [14] to Tucker decomposition on GPU platforms.

This paper focus on Tucker decomposition of dense tensors, in which case, [14] offers the current state-of-the-art method. Although there are some other methods extends it, they all follow its data distribution strategy. But actually, it has not optimized communication cost among different processors yet, which has a great influence on computation efficiency for parallel algorithms. As the data size increases day by day, large scale parallel techniques are in great need for data analysis. So in this paper, we present a new parallel Tucker decomposition method with optimized communication cost. Different from the methods mentioned above, we merge the gram matrix computation and eigen decomposition into one SVD step, which is implemented in parallel based on Hierarchical SVD [17] and leads to much less communication cost.

The rest of this paper is organized as follows. Section 2 is the preliminaries including basic concepts of tensor operations and a simple instruction of ST-HOSVD. Section 3 explains the details of the proposed method, focusing on implementation of parallel SVD and communication cost analysis. Section 4 is the experiments.

2 Preliminaries

2.1 Tensor Notations

In this paper, we denote a tensor by calligraphic letters, like $\mathcal{X}$. The dimension of a tensor is called order (also called mode). The space of a real tensor of order N and size $I_1 \times \ldots \times I_N$ is denoted as $R^{I_1 \times I_2 \times \ldots \times I_N}$.

Definition 1 (Matricization). The mode-n matricization of tensor $\mathcal{X} \in R^{I_n \times I_2 \times \ldots I_N}$ is the matrix, denoted as $\mathcal{X}_{(n)} \in R^{I_n \times \prod_{k \neq n} I_k}$, whose columns are composed of all the vectors obtained from $\mathcal{X}$ by fixing all indices but n-th.

Definition 2 (Folding Operator). Suppose $\mathcal{X}$ be a tensor. The mode-n folding operator of a matrix $M = \mathcal{X}_{(n)}$, denoted as $fold_n(M)$, is the inverse operator of the unfolding operator.

Definition 3 (Mode-n Product, TTM operator, Tensor Times Matrix). Mode-n product of tensor $\mathcal{X} \in R^{I_n \times I_2 \times \ldots I_N}$ and matrix $A \in R^{J \times I_n}$ is denoted by $\mathcal{X} \times_n A$, whose size is $I_1 \times \ldots \times I_{n-1} \times J \times I_{n+1} \times \ldots \times I_N$, and defined by $\mathcal{X} \times_n A = fold_n(M)$.

Definition 4 (Gram Matrix). The Gram Matrix of a matrix M is MM^T.

2.2 St-HOSVD

Tucker factorization is a method that decomposes a given tensor $X \in R^{I_1 \times I_2 \times \ldots \times I_N}$ into N factor matrices $U_1, U_2, \ldots, U_N$ and a core tensor $\mathcal{C}$ such that

$$\mathcal{X} \approx \mathcal{C} \times_1 U_1 \ldots \times_N U_N \tag{1}$$

In this paper, we consider the optimization of ST-HOSVD [10] to solve Tucker decomposition. The algorithm of ST-HOSVD is described as Algorithm 1 below.

Algorithm 1. ST-HOSVD[10]

Input: The tensor $\mathcal{X} \in R^{I_1 \times I_2 \times \ldots \times I_N}$, ranks $(r_1, r_2, \ldots r_N)$
Output: The core tensor $\mathcal{C}$, and factor matrices $U_1, U_2, \ldots, U_N$
$\mathcal{C} \leftarrow \mathcal{X}$
For $k = 1 \ldots N$
    Calculate Gram matrix $: G = \mathcal{C}_{(k)} \cdot \mathcal{C}_{(k)}^T$
    Take the leading r_k eigen vectors of G and form the factor matrix U_k
    $\mathcal{C} \leftarrow \mathcal{C} \times_k U_k$
EndFor

3 The Proposed Method

In this section, we will describe the proposed method in detail. In ST-HOSVD (Algorithm 1), to obtain the factor matrix of every mode, one needs to calculate the gram matrix of $\mathcal{X}_{(n)}$ and its eigen decomposition. But actually, one SVD step is sufficient to obtain the factor matrix. What's more, in parallel cases, using truncated SVD can reduce communication cost, as is explained later. This is also a reason of why we prefer SVD here.

The implementation of parallel SVD in the proposed method can be divided into 3 steps:

- Distribute data to each processor.
- Compute SVD of the local data on each processor.
- Root processor gathers SVD results and merge them to get the factor matrix.

Step 2 and 3 are the key of the proposed method, and data distribution in step 1 should follow them. So Sect. 3.1 explains step 2 and 3 firstly, followed by Sect. 3.2 with accuracy analysis. Section 3.3 describes data distribution and analysis communication cost after that.

3.1 Hierarchical SVD

The matricization of a tensor is usually a highly rectangle matrix $M \in R^{I \times D}$ with $I \ll D$. To compute the SVD of M efficiently, a natural way is to divide M into smaller matrices such that $M = [M_1|M_2|\ldots|M_p]$ with $M_i \in R^{I \times D_i}$, and compute SVD for each M_i and then merge them. Actually, this is the exact idea of Hierarchical SVD [17], which computes U and S parallelly while discarding V for the SVD of $M : U \cdot S \cdot V^T$. And we can see in Algorithm 1, in ST-HOSVD, to obtain the factor matrix of Tucker decomposition for mode n, one only needs the left singular vectors U from the SVD of $\mathcal{X}_{(n)} = U \cdot S \cdot V^T$.

The Hierarchical SVD algorithm calculates the SVD of each M_i by $U_i S_i V_i^T$ and discards the right singular matrices V_i^T. Then it calculates the SVD of $[U_1 S_1|U_2 S_2|\ldots|U_p S_p]$ and output the left singular matrix and singular values. Figure 1 illustrates the progress of Hierarchical SVD.

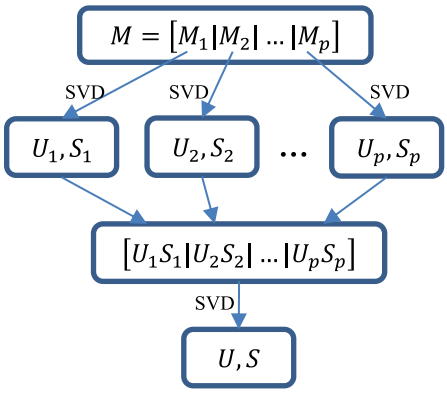

Fig. 1. The illustration of Hierarchical SVD [17].

In the real world, most tensor data are of low-rank, which means truncated SVD with only a few leading singular values may lose little information of the original data.

In this case, using SVD instead of gram matrix can reduce communication cost. The reason runs as follows. Suppose every processor has a local matrix $M_i \in R^{I \times D_i}$. The local gram matrix is of size $I \times I$, which is the transferring data size per processor. But if we use SVD and the truncated rank is d, then each processor only needs to transfer $U_i S_i \in R^{I \times d}$. As mentioned above, d is usually much smaller than I in real world data, so the transferring data size $I \times d$ is also much smaller than that of local gram matrix computation.

In extremely large-scale parallel cases with very large p, one can calculate SVD with more levels to reduce communication cost further. See Fig. 2 as an example. But one should also note that more levels come with larger error for truncated SVD. So we suggest this strategy only serve to trade off accuracy for computation time, or cases of non-truncated SVD.

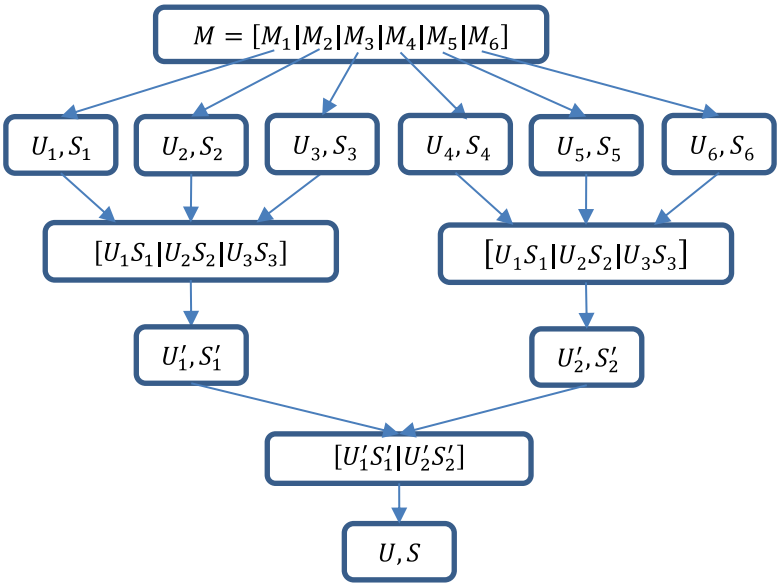

Fig. 2. Example of two-level Hierarchical SVD.

3.2 Discussion on Error of Hierarchical SVD

Now let's come to the accuracy of Hierarchical SVD. For normal SVD, the error analysis is very clear. Let A be a matrix and s_i, u_i, v_i $(i = 1, \ldots n)$ be its singular values in descent order and the corresponding left singular vectors and right singular vectors. Let $(A)_d$ represents the recovered matrix of reduced SVD of A with rank $d \leq n$, that is,

$$(A)_d = \sum_{i=1}^{d} s_i \cdot u_i \cdot v_i^T \tag{2}$$

Then $(A)_d$ is the best approximation of A such that the rank is at most d. And the approximation error is $\|A-(A)_d\|_F=\sqrt{\sum_{i=d+1}^{n}s_i^2}$.

In [17], there is also a theorem to guarantee the error bound of Hierarchical SVD with multiple layers. We presente it here as below.

Theorem 1 (from [17]). Let A be a matrix. Assume that U and S are the outputs of q-level Hierarchical SVD algorithm with input A. Then there exists a unitary matrix V such that $US=AV+\Psi$, where

$$\|\Psi\|_F\le\left[\left(1+\sqrt{2}\right)^{q+1}-1\right]\|A-(A)_d\|_F \tag{3}$$

From Theorem 1, $USV^T=A+\Psi V^T$, and $\|\Psi V^T\|_F=\|\Psi\|_F$, then we can know that U from Hierarchical SVD is the normal left matrix of $A+\Psi V^T$, which is close to A. This means that Hierarchical SVD may increase error of reduced SVD by at most $\left[\left(1+\sqrt{2}\right)^{q+1}-1\right]$ times.

For 1-layer Hierarchical SVD, there is a tighter bound guaranteed by another theorem from [17] as below. Here, we denote $\overline{A}:=US$, where the SVD of A is $A=USV^T$.

Theorem 2. Let $A=\left[A_1|A_2|\ldots|A_p\right]$, $B=\left[\overline{(A_1)_d}|\overline{(A_2)_d}|\ldots|\overline{(A_p)_d}\right]$, and $B'=B+\Psi$, then there exists a unitary matrix W such that

$$\left\|A-\overline{(B')_d}W\right\|_F\le 3\sqrt{2}\left\|A-(A)_d\right\|_F+\left(1+\sqrt{2}\right)\|\Psi\|_F \tag{4}$$

In Theorem 2, it is routine to see that $\overline{(B)_d}=US$, where U and S are the outputs of 1-level Hierarchical SVD algorithm with input A. Setting Ψ to zero matrix will lead to

$$\left\|A-\overline{(B)_d}W\right\|_F\le 3\sqrt{2}\left\|A-(A)_d\right\|_F \tag{5}$$

So for 1-layer Hierarchical SVD, Hierarchical SVD may increase error of reduced SVD by at most $3\sqrt{2}$ times, which is small than the bound in Theorem 1 $\left[\left(1+\sqrt{2}\right)^2-1\right]=2+2\sqrt{2}$.

3.3 Data Distribution and Communication Cost

Consider a tensor $\mathcal{X}\in R^{I_n\times I_2\times\ldots I_N}$. Denote $I=I_1\times I_2\times\ldots\times I_N$ be the number of entries of the tensor, and $J_k=\frac{I}{I_k}$ be the products of all I_i except for I_k.

In STHOSVD, data distribution occurs when implementing Hierarchical SVD across processors. For every mode k, the proposed method applies Hierarchical SVD on $\mathcal{X}_{(k)}$, the matricization of the tensor. We distribute data in a natural way. Divide $\mathcal{X}_{(k)}=\left[M_1|M_2|\ldots|M_p\right]$, where $M_i\in R^{I_k\times D_i}$ and $\sum_{i=1}^{p}D_i=J_k$, and every processor

owns one matrix block M_i. To achieve maximum performance, the matrix blocks should have sizes of almost the same.

Assume that $M_1, M_2 \ldots M_p$ happen to have the same size of $I_k \times \frac{J_k}{p}$. Using the α–β model (the latency cost vs the per-word transfer cost), the communication cost of distributing $M_1, M_2 \ldots M_p$ from root processor to other processors is $(p-1)\alpha + I_k \frac{J_k}{p}\beta = (p-1)\alpha + \frac{I}{p}\beta$. After every processor finishes calculating the SVD of the assigned matrix block M_i, the root processor will gather $U_i S_i$ from all processors, and the communication cost is $\alpha \log_2(p) + (p-1) I_k r_k \beta$, where r_k is the reduced rank of mode k. So the total communication cost when calculating factor matrix of mode k is $(p-1)\alpha + \frac{I}{p}\beta + \alpha \log_2(p) + \frac{(p-1)}{p} I_k r_k \beta = (p - 1 + \log_2(p))\alpha + \left[\frac{I}{p} + \frac{(p-1)}{p} I_k r_k\right]\beta$.

Consider the state-of-the-art method from [14], with p processors organized in a grid of size $p_1 \times p_2 \times \ldots \times p_N$. Calculation of factor matrix of mode k includes an MPI send-receive operator and reduce operator for gram matrix, see [14] for details. The communication cost is $(p_k - 1)\left(\alpha + \frac{I}{p}\beta\right) + \alpha \log_2(p_k) + \frac{(p_k-1)}{p} I_k^2 \beta = (p_k - 1 + \log_2(p_k))\alpha + \left[(p_k - 1)\frac{I}{p} + \frac{(p_k-1)}{p} I_k r_k\right]\beta$. For most cases, $\frac{I}{p}$ is the dominant item. In large-scale parallel cases, p is very large so that every p_k is large too. Comparing the proposed method with the state-of-the-art method from [14], it is easy to see that the former one has lower communication cost.

TTM operator is simple here. We need only to send the factor matrix $\boldsymbol{U_k}$ from root to all processors and the latter ones update the local assigned matrix block by $M_i \leftarrow M_i \times_k U_k$. Here we ignore the communication cost cause it needs little data to transfer, and the case of the state-of-the-art method from [14] is the same.

The proposed distributed ST-HOSVD algorithm is summarized in Algorithm 2.

Algorithm 2. The proposed Distributed ST-HOSVD

Input: The tensor $\mathcal{X} \in R^{I_1 \times I_2 \times \ldots \times I_N}$, ranks $(r_1, r_2, \ldots r_N)$
Output: The core tensor $\mathcal{C}$, and factor matrices $\boldsymbol{U_1}, \boldsymbol{U_2}, \ldots, \boldsymbol{U_N}$
FOR $k = N, N-1, \ldots, 1$
  Distribute data to all processors
  Hierarchical SVD step:
    Every processor calculates the reduced SVD (rank r_k) of the assigned matrix block by $M_i \leftarrow U_i S_i V_i^T$
    Root processor gathers $U_i S_i$ from all processors
    Root processor calculates the reduced SVD (rank r_k) of $\left[U_1 S_1 | U_2 S_2 | \ldots | U_p S_p\right] \leftarrow USV^T$
  Update factor matrix $\boldsymbol{U_k} \leftarrow U$
  Broadcast $\boldsymbol{U_k}$ to each processor from root.
  Every processor updates the assigned matrix block by $M_i \leftarrow M_i \times_k \boldsymbol{U}_k$
ENDFOR
Gather all M_i to root processor and fold them to form the tensor core tensor $\mathcal{C}$.

4 Experiments

We conduct two kinds of experiments in this section. The first one focuses on the merging results of Hierarchical SVD, and the second one focuses on the comparison of the proposed method and the state-of-the-art method.

4.1 Experiments of SVD

In this part, we generate a random matrix A of size 128×1024. The matrix A is divided into different number of blocks by columns ($A = [A_1|A_2|\ldots|A_p]$), and the 1-layer Hierarchical SVD with reduced rank 64 is implemented on it. Reduced SVD is used here as a comparison. In Hierarchical SVD, the right singular matrix V is discarded, and the left singular matrix U is the key output. To evaluate the quality of U, we estimate the projection error of A by

$$e = \frac{\|A - UU'A\|_F}{\|A\|_F} \tag{6}$$

For the normal reduced SVD, the error is 0.2962. For Hierarchical SVD, the results runs is presented in Table 1. We can see that the projection errors of Hierarchical SVD are very close to that of the normal reduced SVD. These results show Hierssarchical SVD has only little accuracy lost compared with normal reduced SVD.

Table 1. Projection Errors with different p.

p	2	4	8	16
e	0.2986	0.2980	0.2969	0.2962

4.2 Experiments of Tucker Decomposition

In this part, we present some experimental results to verify the performance of the proposed method. The experiments are implemented on a cluster with 128 nodes. Each node is equipped with 128 GB memory, a 32-core CPU of 2.0 GHz and a GPU of 13.3 TFLOPs peak single precision (FP32) Performance. We generate five tensor of size $1024 \times 1024 \times 1024$ randomly, and compare the performances of the proposed method with the state-of-the-art method from [14]. The two methods are implemented using different number of nodes, namely 16, 32, 64, 128, to test their scalabilities. For each method, we apply it to the five tensors and set the reduced rank to be 512 for all modes. For the proposed method, the number of layers is set to 1 in Hierarchical SVD. The average executing time of each method is taken as its performance. The experimental result is reported in Table 2 and Fig. 3. It is clear that the proposed method has better performance with larger scale parallelism.

Table 2. The average executing time versus clusters of different number of nodes.

#nodes	Time (s)	
	Proposed	[14]
16	779.86	214.33
32	413.84	180.18
64	237.22	285.16
128	151.58	537.62

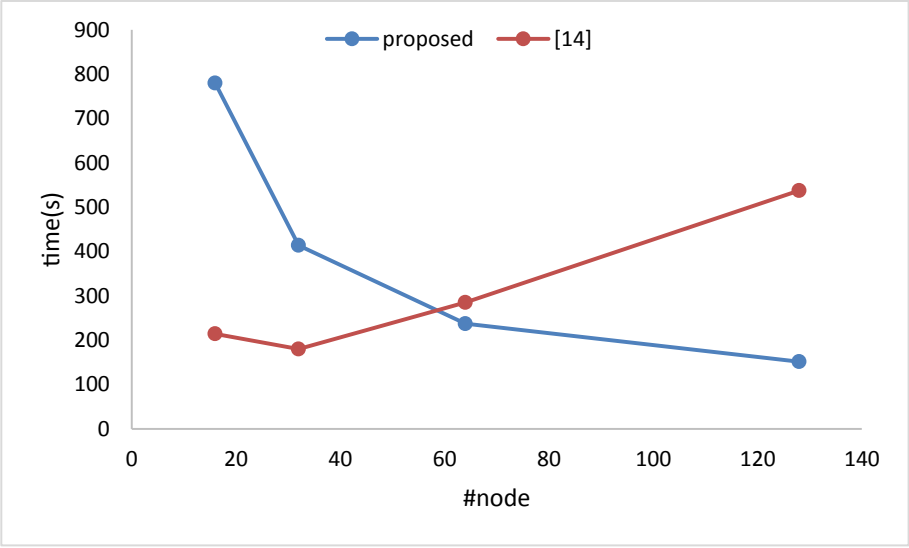

Fig. 3. The average executing time versus clusters of different number of nodes.

5 Conclusion

In this paper, we focus on the problem of high-performance Tucker decomposition on distributed-memory systems. We have proposed a method based on Hierarchical SVD for the problem, which has lower communication cost in large-scale cases compared with the state-of-the-art method. The experiments highlight the proposed method in term of executing time although there is a little accuracy lost.

In the future, we are going to explore how to make full use of the idea of Hierarchical SVD to improve parallel efficiency and reduce the error bound.

Acknowledgements. The work was supported by the National Key Research and Development Project of China (Grant No. 2019YFB2102500).

References

1. Barmpoutis, A.: Tensor body: real-time reconstruction of the human body and avatar synthesis from RGB-D. IEEE Trans. Cybern. **43**(5), 1347–1356 (2013)
2. Tao, D., Li, X., Wu, X., Maybank, S.J.: General tensor discriminant analysis and Gabor features for gait recognition. IEEE Trans. Pattern Anal. Mach. Intell. **29**(10), 1700–1715 (2007)
3. Li, X., Lin, S., Yan, S., Xu, D.: Discriminant locally linear embedding with high-order tensor data . IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **38**(2), 342–352 (2008)
4. Signoretto, M., Tran Dinh, Q., De Lathauwer, L., Suykens, J.A.K.: Learning with tensors: a framework based on convex optimization and spectral regularization. Mach. Learn. **94**(3), 303–351 (2013). https://doi.org/10.1007/s10994-013-5366-3
5. Mørup, M., Hansen, L.K., Herrmann, C.S., Parnas, J., Arnfred, S.M.: Parallel factor analysis as an exploratory tool for wavelet transformed event-related EEG. NeuroImage **29**(3), 938–947 (2006)
6. Mørup, M.: Applications of tensor (multiway array) factorizations and decompositions in data mining. Wiley Interdisc. Rev. Data Mining Knowl. Disc. **1**(1), 24–40 (2011)
7. Sun, J., Papadimitriou, S., Lin, C.-Y., Cao, N., Liu, S., Qian, W.: Multivis: content-based social network exploration through multi-way visual analysis. In: Proceedings of the 2009 SIAM International Conference on Data Mining, pp. 1064–1075. SIAM (2009)
8. Tucker, L.R.: Some mathematical notes on three-mode factor analysis. Psychometrika **31**(3), 279–311 (1966)
9. De Lathauwer, L., De Moor, B., Vandewalle, J.: On the best rank-1 and rank-(R1, R2,…, RN) approximation of high-order tensors. SIAM J. Matrix Anal. Appl. **21**(4), 1324–1342 (2000)
10. Lathauwer, L.D., Moor, B.D., Vandewalle, J.: A multilinear singular value decomposition. SIAM J. Matrix Anal. Appl. **21**, 1253–1278 (2000)
11. Vannieuwenhoven, N., Vandebril, R., Meerbergen, K.: A new truncation strategy for the higher-order singular value decomposition. SIAM J. Sci. Comput. **34**(2), A1027–A1052 (2012)
12. Hadoop information. https://hadoop.apache.org/
13. Jeon, I., Papalexakis, E.E., Kang, U., Faloutsos, C.: HaTen2: billion-scale tensor decompositions. In: 2015 IEEE 31st International Conference on Data Engineering, Seoul, 2015, pp. 1047–1058. https://doi.org/10.1109/ICDE.2015.7113355
14. Austin, W., Ballard, G., Kolda, T.G.: Parallel tensor compression for large-scale scientific data. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Chicago, IL, pp. 912–922 (2016). https://doi.org/10.1109/IPDPS.2016.67
15. Chakaravarthy, V.T., Choi, J.W., Joseph, D.J., Murali, P., Sreedhar, D.: On optimizing distributed tucker decomposition for sparse tensors. In: ICS 2018: Proceedings of the 2018 International Conference on Supercomputing, pp. 374–384 (2018)
16. Zou, B., Li, C., Tan, L., Chen, H.: GPUTensor: efficient tensor factorization for context-aware recommendations. Inf. Sci. **299**(4), 159–177 (2015)
17. Iwen, M.A., Ong, B.W.: A distributed and incremental SVD algorithm for agglomerative data analysis on large networks. SIAM J. Matrix Anal. Appl. **37**(4), 1699–1718 (2016)