Supplementary Document
D-Tucker: Fast and Memory-Efficient Tucker Decomposition for Dense Tensors

Jun-Gi Jang and U Kang

***Abstract*—In this supplementary document, we provide an algorithm, theoretical proofs, and experimental results omitted from the main paper.**

Algorithm 1: Randomized SVD [1]

Input: matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, target rank k, and sampling parameters p and l
Output: SVD results $\mathbf{U} \in \mathbb{R}^{m \times k}$, $\mathbf{\Sigma} \in \mathbb{R}^{k \times k}$, $\mathbf{V} \in \mathbb{R}^{n \times k}$
1: draw random matrices $\mathbf{\Omega} \in \mathbb{R}^{p \times m}$ and $\mathbf{\Psi} \in \mathbb{R}^{l \times n}$
2: form matrices $\mathbf{Y} = \mathbf{\Omega A}$ and $\mathbf{Z} = \mathbf{\Psi A}^T$
3: obtain column orthogonal matrices $\mathbf{Q}$ and $\mathbf{P}$ by QR factorization of $\mathbf{Y}^T$ and $\mathbf{Z}^T$.
4: form matrices $\mathbf{W} = \mathbf{\Omega P}$ and $\mathbf{B} = \mathbf{YQ}$.
5: obtain a matrix $\mathbf{X}$ which minimizes $\|\mathbf{WX} - \mathbf{B}\|$
6: compute SVD of $\mathbf{X} = \tilde{\mathbf{U}}\mathbf{\Sigma}\tilde{\mathbf{V}}^T$
7: $\mathbf{U} \leftarrow \mathbf{P}\tilde{\mathbf{U}}_k$, $\mathbf{\Sigma} \leftarrow \tilde{\mathbf{\Sigma}}_k$, $\mathbf{V} \leftarrow \mathbf{Q}\tilde{\mathbf{V}}_k$

I. Algorithm for Randomized SVD

Algorithm 1 describes randomized SVD. We use sparse embedding matrices [2], [3] to form matrices $\mathbf{Y}$ and $\mathbf{Z}$ in line 2 of Algorithm 1.

II. Proof of Lemmas

Lemma 1. *The approximation phase of D-Tucker takes $O(I^2K^{N-2})$ where I is the largest dimensionality, and K is the remaining dimensionality.*

Proof. Performing randomized SVD of each sliced matrix takes $O(I^2)$ (Algorithm 1 of the main paper). Since the number of sliced matrices is K^{N-2}, the time complexity of the approximation phase is $O(I^2K^{N-2})$. □

Lemma 2. *The initialization phase of D-Tucker takes $O(IK^{N-2}J^2)$ where I is the largest dimensionality, K is the remaining dimensionality, and J is the rank.*

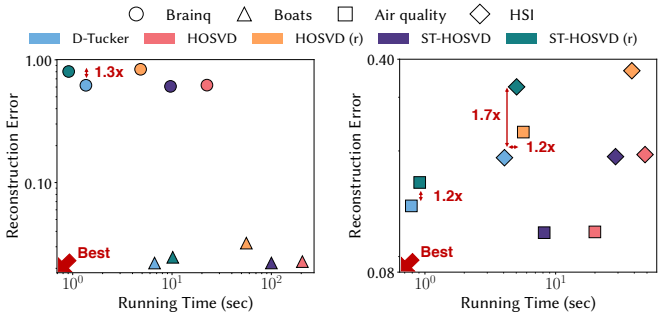
Proof. For the first mode, size of $\left[\mathbf{U}_1\mathbf{\Sigma}_1; \cdots; \mathbf{U}_l\mathbf{\Sigma}_l\right]$ is $I \times K^{N-2}J$. Then, performing SVD [4] takes $O(IK^{N-2}J^2)$ for the first mode. For the second mode, it takes $O(IK^{N-2}J^2)$ to compute $\mathbf{Y}_{(2),inter}$ (line 3 of Algorithm 5 of the main paper), $O(IK^{N-2}J^2)$ to compute $\mathbf{Y}_{(2),inter}blkdiag(\{\mathbf{\Sigma}_l\mathbf{V}_l^{\mathrm{T}}\}_{l=1}^{L})$, and $O(IK^{N-2}J^2)$ to perform SVD of $\mathbf{Y}_{(2)}$. Then, it takes $O(IK^{N-2}J^2)$ to initialize the factor matrix of the second mode. For the remaining modes, it takes $O(IK^{N-2}J^2 + \sum_{k=0}^{N-3} K^{N-2-k}J^{3+k})$ to compute the remaining n-mode products for all $n = 2, 3, ..., N$, and $O(J\sum_{k=0}^{N-3} K^{N-2-k}J^{2+k})$ to compute SVD for all $n = 3, 4, ..., N$. For all modes, the dominant term is $O(IK^{N-2}J^2)$ since $I > K > J$, thus we simplify the time complexity of the initialization phase as $O(IK^{N-2}J^2)$. □

TABLE I
Complexity analysis of D-Tucker and other HOSVD based methods with respect to time and memory. The optimal complexities are in bold. Note that memory complexity indicates the space requirement while updating factor matrices and the core tensor. I denotes the two largest dimensionalities, K is the remaining dimensionalities, M is the number of iterations, J is the dimensionality of the core tensor, and N is the order of the given tensor.

Algorithm	Time	Memory
D-Tucker	$\mathbf{O(I^2K^{N-2} + MNIK^{N-2}J^2)}$	$\mathbf{O(IK^{N-2}J)}$
HOSVD	$O(NI^2K^{N-2}J)$	$O(I^2K^{N-2})$
HOSVD (r)	$O(NI^2K^{N-2})$	$O(I^2K^{N-2})$
ST-HOSVD	$O(I^2K^{N-2}J)$	$O(I^2K^{N-2})$
ST-HOSVD (r)	$O(I^2K^{N-2}J)$	$O(I^2K^{N-2})$

Lemma 3. *The time complexity of an iteration at the iteration phase is $O(NIK^{N-2}J^2)$ where N is the order of a given tensor, I is the largest dimensionality, K is the remaining dimensionality, and J is the rank.*

Proof. For the first mode, it takes $O(IK^{N-2}J^2)$ to compute $\mathbf{Y}_{(1),inter}$ and $\mathbf{Y}_{(1),inter}blkdiag(\{\mathbf{\Sigma}_l\mathbf{U}_l^{\mathrm{T}}\}_{l=1}^{L})$ in Equation 9 of the main paper, and $O(I\sum_{k=0}^{N-3} K^{N-2-k}J^{2+k})$ for computing the remaining n-mode products for all $n = 3, 4, ..., N$. We simplify $O(I\sum_{k=0}^{N-3} K^{N-2-k}J^{2+k})$ as $O(NIK^{N-2}J^2)$ since $J < K$. Computational time of the second mode is the same as that of the first mode. Before computing for remaining modes, it takes $O(IK^{N-2}J^2 + K^{N-2}J^3)$ to compute $\mathbf{Y}_{reuse}$ in line 15 of Algorithm 6 of the main paper. For mode-i, it takes $O(-K^{N-(i-1)}J^i + \sum_{k=0}^{N-3} K^{N-2-k}J^{3+k})$ to perform n-mode products for all $n = 3, 4, i-1, i+1, ..., N$. For core tensor, it takes $O(\sum_{k=0}^{N-3} K^{N-2-k}J^{3+k})$ to perform n-mode products for all $n = 3, 4, ..., N$. We simplify the complexity $O(-K^{N-(i-1)}J^i + \sum_{k=0}^{N-3} K^{N-2-k}J^{3+k})$ and $O(\sum_{k=0}^{N-3} K^{N-2-k}\ J^{3+k})$ to $O(NK^{N-2}J^3)$ since $K > J$. Therefore, the time complexity of one iteration in the iteration phase is $O(IK^{N-2}J^2 + NIK^{N-2}J^2 + IK^{N-2}J^2 + K^{N-2}J^3 + NK^{N-2}J^3)$. Without loss of generality, we express the time complexity of the iteration phase as $O(NIK^{N-2}J^2)$ since $I > K > J$. □



(a) Running time vs. error for Brainq and Boats datasets

(b) Running time vs. error for Air quality and HSI datasets

Fig. 1. D-Tucker provides the best trade-off between error and running time. (r) indicates that a method is implemented with randomized SVD.

III. Additional Results

We additionally evaluate the performance of D-Tucker.

A. Comparison with HOSVD based methods

We additionally compare D-Tucker with HOSVD based Tucker methods.

*1) **Theoretical Comparison**:* We theoretically compare D-Tucker with HOSVD based Tucker methods, HOSVD, ST-HOSVD, randomized HOSVD, and randomized ST-HOSVD. To decompose a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, randomized HOSVD and ST-HOSVD use the randomized SVD of time complexity $O(mn)$ while HOSVD and ST-HOSVD use the standard SVD [4] of time complexity $O(mnk)$ where k is a target rank.

Time Complexity. The dominant term of HOSVD and randomized HOSVD is to perform SVD. Therefore, the time complexities of HOSVD and randomized HOSVD are $O(NI^2K^{N-2}J)$ and $O(NI^2K^{N-2})$, respectively, where N is the order, I is the two largest dimensionalities, and K denotes the remaining dimensionalities. The dominant terms of ST-HOSVD and randomized ST-HOSVD are to perform SVD and mode-1 product for the first mode when processing in descending order. Therefore, the time complexities of ST-HOSVD and randomized ST-HOSVD are $O(I^2K^{N-2}J + I^2K^{N-2}J)$ and $O(I^2K^{N-2} + I^2K^{N-2}J)$, respectively.

The time complexity of D-Tucker is smaller than those of HOSVD and randomized HOSVD when $I > M \times J$. Moreover, the time complexity of D-Tucker is faster than those of ST-HOSVD and randomized ST-HOSVD when $I > M \times N \times J$.

Space Complexity. All the HOSVD based methods require $O(I^2K^{N-2})$ space to deal with a matricized matrix of an input tensor of size $I_1 \times I_2 \times K_3 \cdots \times K_N$ when performing SVD of a matricized matrix. Therefore, D-Tucker requires less space than HOSVD based methods since D-Tucker processes an input tensor block by block in the approximation phase and a small approximated result of size $IK^{N-2}J$ in the initialization and the iteration phases. Note that the size of each block processed in the approximation phase is I^2.

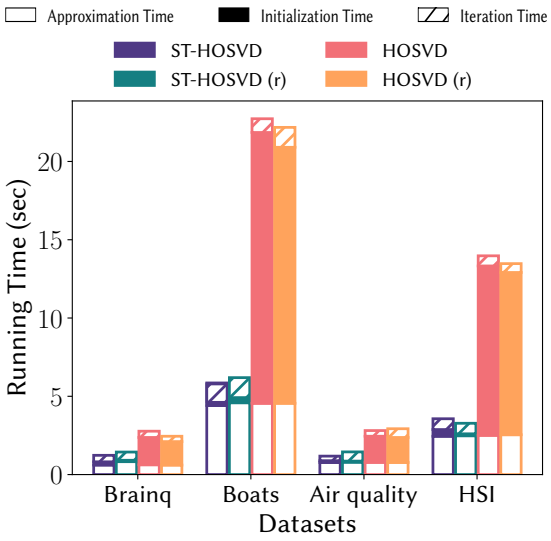


Fig. 2. Running time of D-Tucker initialized by HOSVD based methods. D-Tucker initialized with standard ST-HOSVD gives the best performance.

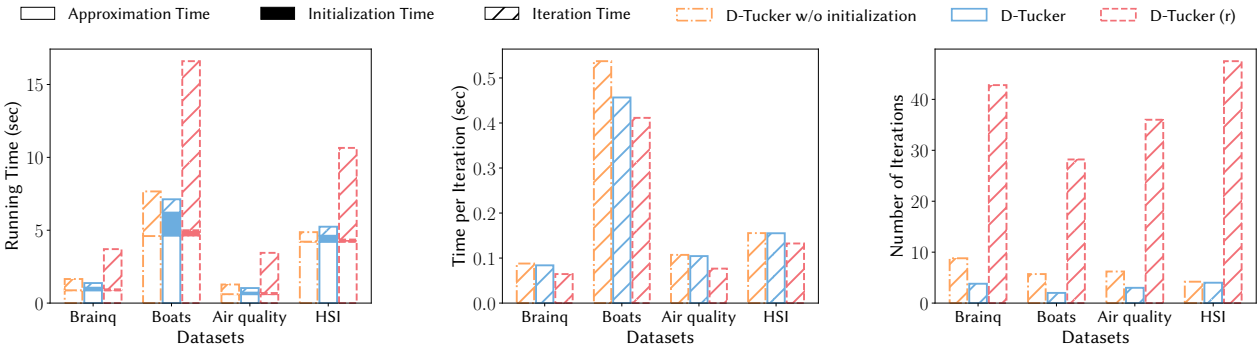
*2) **Experimental Comparison**:* We experimentally compare D-Tucker with HOSVD based Tucker methods. All the methods are implemented by MALTAB (R2019b).

- **HOSVD**: Higher-Order SVD. For each mode, we perform matricization and SVD of the matricized matrix. We use the implementation in Tensor Toolbox [5].
- **HOSVD (r)**: Higher-Order SVD based on randomized SVD. For each mode, we perform matricization and randomized SVD of the matricized matrix.
- **ST-HOSVD**: Sequentially Truncated Higher Order SVD [6]. We use the implementation in Tensor Toolbox [5].
- **ST-HOSVD (r)**: Sequentially Truncated Higher Order SVD based on randomized SVD. We implement ST-HOSVD with randomized SVD.

Figure 1 shows that D-Tucker provides the best trade-off between the running time and the reconstruction error for four datasets. Note that 1) randomized ST-HOSVD gives large errors while efficiently obtaining factor matrices and core tensor, 2) randomized HOSVD is slower than D-Tucker and has large errors, and 3) standard HOSVD and ST-HOSVD are much slower than D-Tucker, but have similar errors.

B. D-Tucker Initialized by HOSVD based Methods

We measured the running times of D-Tucker initialized by HOSVD based methods in Figure 2. D-Tucker with standard ST-HOSVD has the best performance for Brainq, Boats, and Air quality datasets. For HSI dataset, D-Tucker with ST-HOSVD is slightly worse than the best performing variant, D-Tucker with randomized ST-HOSVD. By avoiding reconstruction and carefully ordering matrix operations, ST-HOSVD methods efficiently initialize D-Tucker. Contrary to ST-HOSVD, HOSVD methods are slow since they do not avoid reconstruction of an input tensor and compute SVD of large matricized matrices for mode $i = 3, 4, ..., N$.



(a) Running time (b) Time per Iteration (c) Number of Iterations

Fig. 3. We compare D-Tucker with its variants: D-Tucker without initialization, and D-Tucker applying randomized SVD in the initialization and the iteration phases (denoted by D-Tucker (r)). D-Tucker (r) indicates the implementation of D-Tucker with randomized SVD in the initialization and the iteration phases instead of standard SVD. (a) D-Tucker has better performance than the two variants of D-Tucker on the given datasets, except in the HSI dataset, where its performance is only marginally worse than the best performing variant. (b)(c) The initialization and the iteration phases with randomized SVD dramatically increase the number of iterations although it slightly reduces the running time per iteration.

C. Randomized SVD in the Initialization and Iteration Phases

We investigate the performance of D-Tucker when we use randomized SVD in the initialization and the iteration phases. Figure 3(a) shows that D-Tucker slows down when D-Tucker uses randomized SVD in the initialization and the iteration phases. The reason is that the number of iterations dramatically increases as shown in Figure 3(c) although the running time per iteration slightly decreases as shown in Figure 3(b). Therefore, we use standard SVD [4] in the initialization and the iteration phases to focus on reducing the number of iterations.

REFERENCES

[1] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert, "A fast randomized algorithm for the approximation of matrices," *Applied and Computational Harmonic Analysis*, vol. 25, no. 3, pp. 335–366, 2008.

[2] O. A. Malik and S. Becker, "Low-rank tucker decomposition of large tensors using tensorsketch," in *NeurIPS*, 2018, pp. 10 117–10 127.

[3] K. L. Clarkson and D. P. Woodruff, "Low-rank approximation and regression in input sparsity time," *JACM*, vol. 63, no. 6, p. 54, 2017.

[4] J. Baglama and L. Reichel, "Augmented implicitly restarted lanczos bidiagonalization methods," *SIAM J. Scientific Computing*, vol. 27, no. 1, pp. 19–42, 2005.

[5] B. W. Bader, T. G. Kolda *et al.*, "Matlab tensor toolbox version 3.0-dev," Available online, Oct. 2017. [Online]. Available: https://www.tensortoolbox.org

[6] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, "A new truncation strategy for the higher-order singular value decomposition," *SIAM J. Scientific Computing*, vol. 34, no. 2, 2012.