VeST- Supplementary Document

Anonymous Submission

Algorithm 1: Tucker-ALS for Fully Observable Tensors (HOOI)

```
Input :  Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, and core tensor
         dimensionality $J_1, ..., J_N$.
Output:  Factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n}$
         $(n = 1, ..., N)$, and core tensor
         $\mathcal{G} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$.
1 initialize all factor matrices $\mathbf{A}^{(n)}$
2 repeat
3     for $n = 1...N$ do
4         $\mathcal{Y} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)\mathbf{T}} \cdots \times_{n-1} \mathbf{A}^{(n-1)\mathbf{T}} \times_{n+1} \mathbf{A}^{(n+1)\mathbf{T}} \cdots \times_N \mathbf{A}^{(N)\mathbf{T}}$
5         $\mathbf{A}^{(n)} \leftarrow J_n$ leading left singular vectors of $\mathcal{Y}_{(n)}$
6 until reconstruction error converges or exceeds maximum iteration;
7 $\mathcal{G} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)\mathsf{T}} \cdots \times_N \mathbf{A}^{(N)\mathsf{T}}$
```

I. Tucker ALS Algorithm

A widely used technique for minimizing the loss functions Eq. (1) and Eq. (2) of the main paper in a standard tensor factorization is alternating least squares (ALS) [1], which updates a factor matrix or a core tensor while keeping all others fixed.

Algorithm 1 describes a vanilla Tucker factorization algorithm based on ALS, which is called the *higher-order orthogonal iteration* (HOOI) (see [1] for details) that works on fully observable tensor. Notice that Algorithm 1 assumes missing entries of $\mathcal{X}$ as zeros during the update process (lines 4-5). However, setting missing values to zero enforces Tucker ALS to factorize the original tensor such that missing values becomes zero when reconstructed. Note that the missing values are often nonzero values that are unknown. Thus, setting missing values to zero inserts false information into the factorization which results in higher reconstruction error as well as higher generalization error. Moreover, Algorithm 1 computes SVD (singular vector decomposition) given $\mathcal{Y}_{(n)}$, which often results in dense matrices, thus tensor-ALS results in overall dense core tensor and factor matrices. Also, Algorithm 1 requires storing a full-dense matrix $\mathcal{Y}_{(n)}$, and the amount of memory needed for storing $\mathcal{Y}_{(n)}$ is $O(I_n \prod_{m \neq n} J_m)$. The required memory grows rapidly when the order, the mode dimensionality, or the rank of a tensor increase, and ultimately causes *intermediate data explosion* [2].

In summary, the vanilla Tucker-ALS algorithm results in high generalization error in the presence of missing data, results in dense and thus hard-to-interpret core tensor and factor matrices, and cannot be applied to large data.

Many tucker factorization algorithms have been developed. Recent works [3]–[5] efficiently update factor matrices and core tensor by dealing with only observed values. However, all of the methods do not consider the sparsity of factor matrices and core tensor. Although Oh et al. [5] truncate noisy entries of core tensor, total sparsity is very small since the size of core tensor is relatively smaller than that of factor matrices. Therefore, Algorithm 1 needs to be revised to focus only on observed entries, make sparse outputs, and be scaled for large-scale tensors at the same time.

II. Proofs of Update Rules and Responsibilities

A. Proof of Lemma 1 (Updates of Factor Matrices with L_1 regularization)

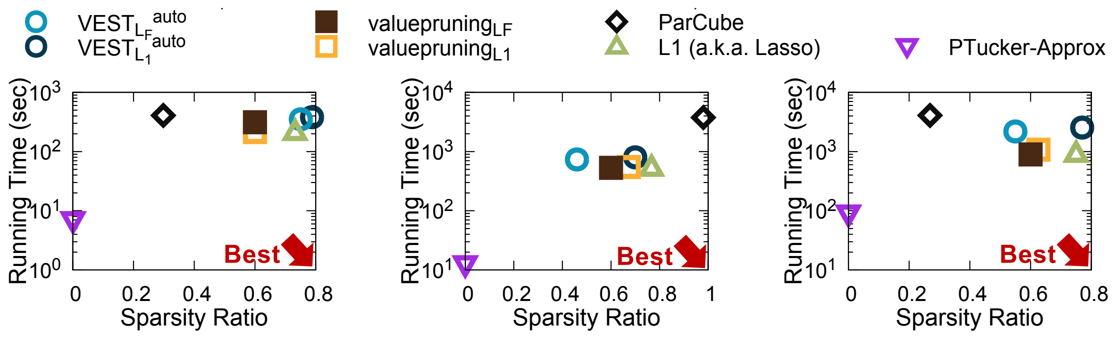
Proof 1 The partial derivative of L_1 regularization loss function with regard to the factor matrix entry $a^{(n)}_{i_n j_n}$ is

$$\begin{aligned}\frac{\partial L_1}{\partial a^{(n)}_{i_n j_n}} &= \left[2 \sum_{\forall \alpha \in \Omega^{(n)}_{i_n}} \left((\mathcal{X}_\alpha - \sum_{\forall t \neq j_n} \boldsymbol{\delta}^{(n)}_\alpha(t) a^{(n)}_{i_n t}) \cdot (-\boldsymbol{\delta}^{(n)}_\alpha(j_n))\right)\right] \\ &\quad + \left[2 \sum_{\forall \alpha \in \Omega^{(n)}_{i_n}} \left(\boldsymbol{\delta}^{(n)}_\alpha(j_n)\right)^2 \cdot a^{(n)}_{i_n j_n}\right] + \lambda \frac{\partial |a^{(n)}_{i_n j_n}|}{\partial a^{(n)}_{i_n j_n}} \\ &= g_{fm} + d_{fm} \cdot a^{(n)}_{i_n j_n} + \lambda \frac{\partial |a^{(n)}_{i_n j_n}|}{\partial a^{(n)}_{i_n j_n}} \\ &= \begin{cases} g_{fm} + d_{fm} \cdot a^{(n)}_{i_n j_n} + \lambda & \text{if } a^{(n)}_{i_n j_n} > 0 \\ g_{fm} + d_{fm} \cdot a^{(n)}_{i_n j_n} - \lambda & \text{if } a^{(n)}_{i_n j_n} < 0 \end{cases}\end{aligned} \tag{2}$$

Case 1 $(g_{fm} > \lambda (> 0))$: if $a^{(n)}_{i_n j_n} > 0$, then $\frac{\partial L_1}{\partial a^{(n)}_{i_n j_n}} > 0$ since $g_{fm} + \lambda > 0$, and $d_{fm} > 0$. $\frac{\partial L_1}{\partial a^{(n)}_{i_n j_n}} = 0$ when $a^{(n)}_{i_n j_n} = (\lambda - g_{fm})/d_{fm} (> 0)$, so that the value of $\frac{\partial L_1}{\partial a^{(n)}_{i_n j_n}}$ becomes negative when $a^{(n)}_{i_n j_n} < (\lambda - g_{fm})/d_{fm}$. In sum, L_1 decreases if $a_{i_n j_n} < (\lambda - g_{fm})/d_{fm}$, and increases if $a_{i_n j_n} > (\lambda - g_{fm})/d_{fm}$. Thus, the loss becomes minimum when $a^{(n)}_{i_n j_n} = (\lambda - g_{fm})/d_{fm}$.

Case 2 $(g_{fm} < -\lambda (< 0))$: likewise, if $a^{(n)}_{i_n j_n} < 0$, then $\frac{\partial L_1}{\partial a^{(n)}_{i_n j_n}} < 0$ since $g_{fm} - \lambda < 0$, and $d_{fm} > 0$. $\frac{\partial L_1}{\partial a^{(n)}_{i_n j_n}} = 0$ when $a^{(n)}_{i_n j_n} = -(\lambda + g_{fm})/d_{fm} (> 0)$, so that the value of $\frac{\partial L_1}{\partial a^{(n)}_{i_n j_n}}$ becomes positive when $a^{(n)}_{i_n j_n} > -(\lambda + g_{fm})/d_{fm}$. In sum, the loss decreases if $a_{i_n j_n} < -(\lambda + g_{fm})/d_{fm}$, and increases if $a_{i_n j_n} > -(\lambda + g_{fm})/d_{fm}$. Thus, the loss becomes minimum when $a^{(n)}_{i_n j_n} = -(\lambda + g_{fm})/d_{fm}$ respect to $a^{(n)}_{i_n j_n}$.

Case 3 $(-g_{fm} < \lambda$, and $g_{fm} < \lambda)$: If $a^{(n)}_{i_n j_n} > 0$, then $\frac{\partial L}{\partial a^{(n)}_{i_n j_n}} > 0$; if $a^{(n)}_{i_n j_n} < 0$, then $\frac{\partial L}{\partial a^{(n)}_{i_n j_n}} < 0$. In sum, the loss decreases if $a_{i_n j_n} < 0$, and increases if $a_{i_n j_n} > 0$. Thus, the loss becomes minimum when $a^{(n)}_{i_n j_n} = 0$.



(a) Yelp (b) Amazon (c) Movie

Figure 1. Sparsity and running time of VEST and competitors on 3 real-world datasets. VEST obtains sparse results with sacrifice of the computational time compared to PTucker, i.e., highly scalable method. However is faster for all three datasets compared to ParCube.

B. Proof of Lemma 2 (Updates of Core Tensor with L_1 regularization)

Proof 2 The partial derivative of L_1 regularization loss function with regard to core tensor entry $\mathcal{G}_\beta$ is

$$\frac{\partial L_1}{\partial \mathcal{G}_\beta}$$

$$= 2\sum_{\forall\alpha\in\Omega}(\mathcal{X}_\alpha - \sum_{\forall\gamma\neq\beta}\mathcal{G}_\gamma\prod_{n=1}^{N}a^{(n)}_{i_nj_n} - \mathcal{G}_\beta\prod_{n=1}^{N}a^{(n)}_{i_nj_n})(-\prod_{n=1}^{N}a^{(n)}_{i_nj_n}) + \lambda\frac{\partial|\mathcal{G}_\beta|}{\partial\mathcal{G}_\beta}$$

$$= \left[-2\sum_{\forall\alpha\in\Omega}(\mathcal{X}_\alpha - \sum_{\forall\gamma\neq\beta}\mathcal{G}_\gamma\prod_{n=1}^{N}a^{(n)}_{i_nj_n})\cdot(\prod_{n=1}^{N}a^{(n)}_{i_nj_n})\right] + \left[2\sum_{\forall\alpha\in\Omega}(\prod_{n=1}^{N}a^{(n)}_{i_nj_n})^2\cdot\mathcal{G}_\beta\right] + \lambda\frac{\partial|\mathcal{G}_\beta|}{\partial\mathcal{G}_\beta}$$

$$= g_c + d_c\cdot\mathcal{G}_\beta + \lambda\frac{\partial|\mathcal{G}_\beta|}{\partial\mathcal{G}_\beta} = \begin{cases} g_c + d_c\cdot\mathcal{G}_\beta + \lambda & \text{if } \mathcal{G}_\beta > 0 \\ g_c + d_c\cdot\mathcal{G}_\beta - \lambda & \text{if } \mathcal{G}_\beta < 0 \end{cases} \tag{4}$$

The remaining steps are the same as those of update rule for factor matrices with L_1 regularization (Lemma 1).

C. Correctness for the Derivation of Responsibility Values of Factor Matrix Entries

Proof 3 From its definition,

$$(RE)^2||\mathcal{X}||_F^2 = \sum_{\forall\alpha\in\Omega}(\mathcal{X}_\alpha - B(\alpha))^2$$
$$= \sum_{\forall\alpha\in\Omega^{(n)}_{i_n}}(\mathcal{X}_\alpha - B(\alpha))^2 + \sum_{\forall\alpha\notin\Omega^{(n)}_{i_n}}(\mathcal{X}_\alpha - B(\alpha))^2$$
$$= \sum_{\forall\alpha\in\Omega^{(n)}_{i_n}}(\mathcal{X}_\alpha - B_{j_n=j}(\alpha) - B_{j_n\neq j}(\alpha))^2 + \sum_{\forall\alpha\notin\Omega^{(n)}_{i_n}}(\mathcal{X}_\alpha - B(\alpha))^2 \tag{6}$$

Note that

$$\sum_{\forall\alpha\notin\Omega^{(n)}_{i_n}}(\mathcal{X}_\alpha - B(\alpha))^2 = (RE)^2||\mathcal{X}||_F^2 - \sum_{\forall\alpha\in\Omega^{(n)}_{i_n}}(\mathcal{X}_\alpha - B_{j_n=j}(\alpha) - B_{j_n\neq j}(\alpha))^2 \tag{8}$$

Thus,

$$(RE(a^{(n)}_{ij}))^2||\mathcal{X}||_F^2$$
$$= (\sum_{\forall\alpha\in\Omega^{(n)}_{i_n}}(\mathcal{X}_\alpha - B_{j_n\neq j}(\alpha))^2) + \sum_{\forall\alpha\notin\Omega^{(n)}_{i_n}}(\mathcal{X}_\alpha - B(\alpha))^2$$
$$= (\sum_{\forall\alpha\in\Omega^{(n)}_{i_n}}(\mathcal{X}_\alpha - B_{j_n\neq j}(\alpha))^2) + (RE)^2||\mathcal{X}||_F^2 - \sum_{\forall\alpha\in\Omega^{(n)}_{i_n}}(\mathcal{X}_\alpha - B_{j_n=j}(\alpha) - B_{j_n\neq j}(\alpha))^2$$
$$= (RE)^2||\mathcal{X}||_F^2 + \sum_{\forall\alpha\in\Omega^{(n)}_{i_n}}(2\mathcal{X}_\alpha - 2B_{j_n\neq j}(\alpha) - B_{j_n=j}(\alpha))B_{j_n=j}(\alpha)$$
$$= (RE)^2||\mathcal{X}||_F^2 + \sum_{\forall\alpha\in\Omega^{(n)}_{i_n}}(2\cdot(\mathcal{X}_\alpha - B(\alpha)) + B_{j_n=j}(\alpha))B_{j_n=j}(\alpha) \tag{10}$$

Dividing both sides of Eq. (10) with $||\mathcal{X}||_F^2$, we get

$$(RE(a^{(n)}_{ij}))^2 = RE^2 + \frac{\sum_{\forall\alpha\in\Omega^{(n)}_{i_n}}\big(2(\mathcal{X}_\alpha - B(\alpha)) + B_{j_n=j}(\alpha)\big)(B_{j_n=j}(\alpha))}{||\mathcal{X}||_F^2} \tag{12}$$

III. An example of pruning based on responsibility

Here we show an toy example of importance of pruning based on responsibility values compared to pruning based on small factor values. Suppose sparse tensor $\mathcal{X}\in\mathbb{R}^{3\times3\times3}$, whose frontal slices is

$$\mathcal{X}_1 = \left\{\begin{array}{ccc} & & \\ & & \\ & 0.89 & 0.85 \end{array}\right\}, \mathcal{X}_2 = \left\{\begin{array}{ccc} 0.44 & 0.27 & \\ & 0.88 & \\ 0.47 & & \end{array}\right\}, \mathcal{X}_3 = \left\{\begin{array}{ccc} 0.48 & & 0.14 \\ 0.95 & & 0.40 \\ & & \end{array}\right\}. \tag{14}$$

where a blank indicates a missing values. The result of $\mathcal{X}$'s tucker factorization (implemented by VEST^{man} with sparsity $s = 0$) is shown as below ($RE = 0.12$).

$$\mathcal{G} = \{-5.905566\},$$
$$\mathbf{A}^{(1)} = \begin{Bmatrix}-0.33\\-0.87\\-0.36\end{Bmatrix}, \mathbf{A}^{(2)} = \begin{Bmatrix}-0.77\\-0.48\\-0.42\end{Bmatrix}, \mathbf{A}^{(3)} = \begin{Bmatrix}-0.92\\-0.33\\-0.23\end{Bmatrix} \tag{16}$$

If we prune an element with the smallest absolute value, $a^{(3)}_{33}$ has to be pruned, leading $RE = 0.571673$. On the other hand, if we prune an element with the smallest Resp value, $a_{11}^{(1)}$ has to be pruned, leading $RE = 0.37$. Therefore, the above example indicates that it is better to consider the responsibility rather than simply remove small values.

IV. Additional Experiment

A. Trade-offs between running time and sparsity

We compared the sparsity and the running time of VeST^{auto} (default VeST) with VeST^{auto} (VeST with L_F regularization) and those of the competitors on three real-world datasets: Yelp, AmazonFood, and MovieLens. Since TTP, Sparse CP, Tucker-ALS, and APG-NTD have scalability issues, we do not compare VeST with them. In Fig. 1, VeST^{auto} is one of the closest methods to the bottom-right region which indicates the best point. Value Pruning approach is also close to the best point, but Value Pruning achieves a higher error than VeST^{auto} as shown in Fig. 1 of the main paper. VeST^{auto} achieves a higher sparsity than the L1 approach although VeST^{auto} requires an additional computational time to prune factor matrices and core tensor compared to the L1 approach. PTucker-Approx is the fastest method, but a sparsity of PTucker-Approx is too low.

References

[1] T. G. Kolda and B. W. Bader, “Tensor decompositions and applications,” *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.
[2] U. Kang, E. E. Papalexakis, A. Harpale, and C. Faloutsos, “Gigatensor: scaling tensor analysis up by 100 times - algorithms and discoveries,” in *KDD*, pp. 316–324, 2012.
[3] D. Lee, J. Lee, and H. Yu, “Fast tucker factorization for large-scale tensor completion,” in *ICDM 2018*, pp. 1098–1103, 2018.
[4] S. Oh, N. Park, J. Jang, L. Sael, and U. Kang, “High-performance tucker factorization on heterogeneous platforms,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 10, pp. 2237–2248, 2019.
[5] S. Oh, N. Park, L. Sael, and U. Kang, “Scalable Tucker factorization for sparse tensors - algorithms and discoveries,” in *ICDE*, (Paris, France), IEEE Computer Society, 2018.