1.1 Tensor Preliminaries

A tensor is a multi-dimensional array which is a generalization of a matrix and a vector. A mode or a way indicates each axis of a tensor, and an order is the number of modes or ways. We denote a tensor using boldface Euler script letters (e.g., X). A tensor \( X \in \mathbb{R}^{i_1 \times i_2 \times \cdots \times i_n} \) is an \( N \)-order tensor which has \( N \) modes whose lengths are from \( i_1 \) to \( i_N \). A vector and a matrix are regarded as a 1- and 2-order tensors, respectively. We denote a matrix and a vector using boldface uppercase (e.g., X) and lowercase letters (e.g., x), respectively. The \( i \)th row of \( A \) is denoted by \( a_{i1} \), and the \( i \)th column of \( A \) is denoted by \( a_{1i} \).

1.1.2 Tucker Decomposition for Partially Observable Tensors

Among many tensor decomposition methods, we use Tucker factorization (Tucker, 1966; De Lathauwer et al., 2000), which allows us to discover not only latent concepts but also relations between the concepts hidden in tensors. Tucker factorization decomposes a given tensor \( X \) into a core tensor \( G \) and factor matrices \( A^{(1)}, \ldots, A^{(N)} \), as defined in Definition 1.

Definition 1. (Tucker factorization) Given a tensor \( X \in \mathbb{R}^{i_1 \times i_2 \times \cdots \times i_N} \) with observable entries \( \Omega \), Tucker factorization of rank \((J_1, \ldots, J_N)\) finds a core tensor \( G \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N} \), and factor matrices \( A^{(1)} \in \mathbb{R}^{l_1 \times J_1} \), \( A^{(2)} \in \mathbb{R}^{J_1 \times l_2} \), \( \ldots \), \( A^{(N)} \in \mathbb{R}^{J_{N-1} \times l_N} \) which minimize the following objective function.

\[
\mathcal{L}(G; A^{(1)}, \ldots, A^{(N)}) = \\
\sum_{(i_1, \ldots, i_N) \in \Omega} (X_{i_1 \ldots i_N} - \sum_{(j_1, \ldots, j_N) \in \Omega} G_{(j_1, \ldots, j_N)} \prod_{s=1}^N a^{(n)}_{i_sj_s})^2 \\
+ \lambda \sum_{n=1}^N \|A^{(n)}\|_F^2
\]

(1)

Note that \( \lambda \) denotes a regularization parameter for factor matrices, and we used \( L_2 \)-regularization to prevent overfitting, which has been widely used in machine learning (Koren et al., 2009; Shin et al., 2017a).

Equation (1) only utilizes observed entries of a tensor during factorizations as missing entries of \( X \) have unknown values. In addition, there are no constraints (e.g., non-negativity or orthogonality) on factor matrices in Equation (1). Each column vector of a factor matrix generally represents a concept or a pattern. A higher value in a vector indicates that the corresponding element is highly related to the concept. The core tensor encodes how these concepts are related to each other. Assuming a given tensor is movie rating data with (movie - user - time) triples, then a column vector in a movie-factor matrix can have a concept such as a horror or comic genre.

P-Tucker is a scalable and accurate Tucker factorization method which updates factor matrices in a row-wise manner based on ALS. Algorithm 2 and Figures 1 and 2 illustrate how P-Tucker updates factor matrices. In Algorithm 2, P-Tucker first initializes all \( A^{(n)} \) and \( G \) with random real values between 0 and 1 (line 1). After which P-Tucker updates a single factor matrix in a row-wise manner and iterates it for all factor matrices (lines 3-8). When all factor matrices are updated, P-Tucker measures reconstruction error using (5) (line 9). P-Tucker stops iterations if the error converges or the maximum iteration is reached (line 10). Finally, P-Tucker performs an optional QR decomposition on all \( A^{(n)} \) to make them orthogonal and updates \( G \) (lines 12-14). Specifically, QR decomposition on each \( A^{(n)} \) is defined as follows:

\[
A^{(n)} = Q^{(n)} R^{(n)}, \quad n = 1 \ldots N
\]

(2)

where \( Q^{(n)} \in \mathbb{R}^{l_n \times l_n} \) is column-wise orthonormal and \( R^{(n)} \in \mathbb{R}^{l_n \times l_n} \) is upper-triangular. Therefore, by substituting \( Q^{(n)} \) for \( A^{(n)} \), P-Tucker succeeds in making factor matrices orthogonal. When QR decomposition is performed, core tensor \( G \) must be updated accordingly in order to maintain the same reconstruction error. The update rule of core tensor \( G \) is given as follows:

\[
G = G \times_{1} R^{(1)} \times_{2} \cdots \times_{N} R^{(N)}.
\]

(3)

Note that performing QR decomposition is optional, and GIFT skips this process as QR decomposition may sweep out latent patterns in factor matrices.

Given a row of a factor matrix, an update rule is derived by computing a gradient with respect to the given row and setting it as zero, which minimizes the loss function (1). The update rule for the \( i \)th row of the \( n \)th factor matrix \( A^{(n)} \) (see Figure 1) is given as follows.

\[
\arg \min_{c^{(n)}_{i_1 \ldots i_N}} \|d^{(n)}_{i_1 \ldots i_N} - c^{(n)}_{i_1 \ldots i_N} \|_F^2 + \lambda ||c^{(n)}_{i_1 \ldots i_N}||_2^2
\]

(4)

The proposed row-wise update rule of factor matrices with no masking constraints. The proposed row-wise update rule (4) minimizes the loss function (1) regarding the updated parameters.

Theorem 1 (Row-wise update rule of factor matrices with no masking constraints). The proposed row-wise update rule (4) minimizes the loss function (1) regarding the updated parameters.
where $\mathbf{B}_n^{(i)}$ is a $J_n \times J_n$ matrix whose $(j_1, j_2)$th entry is

$$
\sum_{\forall (i_1, \ldots, i_N) \in \Omega^{(i)}} c_{i_n}^{(a)} (x_{i_1,\ldots,i_N}^{(i)} (j_1) \delta_{i_1,\ldots,i_N}^{(i)} (j_2),
$$

(5)

$c_{i_n}^{(a)}$ is a length $J_n$ vector whose $j$th entry is

$$
\sum_{\forall (i_1, \ldots, i_N) \in \Omega^{(i)}} x_{i_1,\ldots,i_N}^{(i)} (j),
$$

(6)

$\mathbf{c}_{j_n}^{(a)}$ is a length $J_n$ vector whose $j$th entry is

$$
\sum_{\forall (i_1, \ldots, i_N) \in \Omega^{(i)}} \mathbf{a}_{i_n}^{(k)} \mathbf{B}_n^{(i)}
$$

(7)

Proof.

$$
\frac{\partial L}{\partial a^{(i)}_{i_n}} = 0, \forall j_n, 1 \leq j_n \leq J_n
$$

$$
= \sum_{\forall (i_1, \ldots, i_N) \in \Omega^{(i)}} \left( [x_n - \sum_{\alpha=1}^{J_n} \Pi_{\alpha}^{(i)} c_{\alpha}^{(a)}] \right) a_{i_n}^{(k)} = 0
$$

$$
\Rightarrow a_{i_n}^{(k)} = \sum_{\alpha=1}^{J_n} \Pi_{\alpha}^{(i)} c_{\alpha}^{(a)}
$$

$$
\Rightarrow a_{i_n}^{(k)} = \frac{\mathbf{c}_{j_n}^{(a)} \times \mathbf{B}_n^{(i)} + \lambda \mathbf{I}_{J_n}}{\sum_{\alpha=1}^{J_n} \Pi_{\alpha}^{(i)} c_{\alpha}^{(a)}}
$$

$\Omega^{(i)}$ indicates the subset of $\Omega$ whose mode’s index is $i_n$, $\lambda$ is a regularization parameter, and $\mathbf{I}_{J_n}$ is a $J_n \times J_n$ identity matrix. As shown in Figure 1, the update rule for the $i_n$th row of $\mathbf{A}^{(i)}$ requires three intermediate data $\mathbf{B}_n^{(i)}$, $\mathbf{c}_{j_n}^{(a)}$, and $\delta_{i_n}^{(a)}$. Those data are computed by the subset of observable entries $\Omega^{(i)}$. Thus, computational costs of updating factor matrices are proportional to the number of observable entries, which lets P-Tucker fully exploit the sparsity of given tensors. Note that a matrix $[\mathbf{B}_n^{(i)} + \lambda \mathbf{I}_{J_n}]$ is positive-definite and invertible, and the full version of Proof 1.3 is available at https://datalab.snu.ac.kr/ptucker_supp.pdf.

**Algorithm 2 P-Tucker**

**Input:** A tensor $\mathbf{X} \in \mathbb{R}^{N_1 \times \cdots \times N_N}$ with observable entries $\mathbf{X}$, mask matrices $\mathbf{M}^{(1)}, \ldots, \mathbf{M}^{(N)}$, rank $\{J_1, \ldots, J_N\}$, and a regularization parameter $\lambda$.

**Output:** A core tensor $\mathbf{G}$ and factor matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$.

1. Initialize $\mathbf{G}$ and $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$ randomly
2. repeat
3. for $n = 1, \ldots, N$ do
4. for $i_n = 1, \ldots, I_n$ do
5. calculate intermediate data $\delta_{i_n}^{(a)}$, $\mathbf{c}_{j_n}^{(a)}$, and $\mathbf{B}_n^{(i)}$ by Eq. (2) – (4)
6. update a row $a_{i_n}^{(k)}$ by $a_{i_n}^{(k)} = \frac{\mathbf{c}_{j_n}^{(a)} \times \mathbf{B}_n^{(i)} + \lambda \mathbf{I}_{J_n}}{\sum_{\alpha=1}^{J_n} \Pi_{\alpha}^{(i)} c_{\alpha}^{(a)}}$
7. end for
8. end for
9. compute reconstruction error by Eq. (5)
10. until a convergence criterion is met
11. for $n = 1, \ldots, N$ do
12. $\mathbf{A}^{(n)} \leftarrow \mathbf{Q}^{(n)} \mathbf{R}^{(n)}$
13. $\mathbf{A}^{(n)} \leftarrow \mathbf{Q}^{(n)}$
14. $\mathbf{G} \leftarrow \mathbf{G} \times \mathbf{A}^{(n)}$
15. end for

1.3 GIFT Algorithm for General $N$-order Tensors

GIFT adopts the row-wise update rule proposed for P-Tucker with modified loss function. The derivation of the update rules are similar to that of P-Tucker as provided in the following Theorem 2 and proof.

Theorem 2 (Row-wise update rule of factor matrices with masking constraints). The proposed row-wise update rule (8) minimizes the loss function (6) regarding the updated parameters.

$$
\min_{\mathbf{D}^{(i)}_{J_n \times J_n}} \frac{1}{2} \sum_{\forall (i_1, \ldots, i_N) \in \Omega^{(i)}} \left( [x_{i_1,\ldots,i_N}^{(i)} (j_1) \delta_{i_1,\ldots,i_N}^{(i)} (j_2),
$$

$$
\mathbf{c}_{j_n}^{(a)} \mathbf{B}_n^{(i)}
$$

(8)

$$
\frac{\partial L}{\partial a^{(i)}_{i_n}} = 0, \forall j_n, 1 \leq j_n \leq J_n
$$

$$
= \sum_{\forall (i_1, \ldots, i_N) \in \Omega^{(i)}} \left( [x_{i_1,\ldots,i_N}^{(i)} (j_1) \delta_{i_1,\ldots,i_N}^{(i)} (j_2),
$$

$$
\mathbf{c}_{j_n}^{(a)} \mathbf{B}_n^{(i)}
$$

(9)

**Proof.**

$$
\frac{\partial L}{\partial a^{(i)}_{i_n}} = 0, \forall j_n, 1 \leq j_n \leq J_n
$$

$$
= \sum_{\forall (i_1, \ldots, i_N) \in \Omega^{(i)}} \left( [x_{i_1,\ldots,i_N}^{(i)} (j_1) \delta_{i_1,\ldots,i_N}^{(i)} (j_2),
$$

$$
\mathbf{c}_{j_n}^{(a)} \mathbf{B}_n^{(i)}
$$

(10)
1.4 Theoretical analyses of GIFT

In this section, we offer theoretical analyses of GIFT in terms of time, memory, and convergence. Specifically, we analyze time and memory complexity of GIFT. Note that we calculate time complexity per iteration, and focus on memory complexity of intermediate data, not of all variables.

Table 2. Complexity analysis of GIFT with respect to time and memory. Note that memory complexity indicates the space requirement for intermediate data. \(|T_i|\) is the number of observable entries in \(X\) and \(T\) is the number of threads.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time Complexity (per iteration)</th>
<th>Memory Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIFT</td>
<td>(O(NIJ^3 + N^2(2IJ)^3))</td>
<td>(O(TJ^3))</td>
</tr>
</tbody>
</table>

Theorem 3 (Time complexity of GIFT). The time complexity of GIFT is \(O(NIJ^3 + N^2(2IJ)^3)\).

Proof. Given the \(i_{th}\) row of \(A^{(n)}\) (lines 3-4) in Algorithm 1, computing \(\delta\) (line 5) takes \(O(N|\delta^{(n)}|JN)\). Updating \(B^{(n)}\), \(e_i^{(n)}\), and \(D^{(n)}\) (lines 5-6) takes \(O(|\delta^{(n)}|J^2)\) since \(\delta\) is already calculated. Inverting \(\{B^{(n)} + \lambda D^{(n)}\}\) (line 7) takes \(O(J^3)\), and updating a row (line 7) takes \(O(|\delta^{(n)}|J^3)\). Thus, the time complexity of updating the \(i_{th}\) row of \(A^{(n)}\) (lines 5-7) is \(O(J^3 + |\delta^{(n)}|J^3)\). Iterating it for all rows of \(A^{(n)}\) (lines 5-7) takes \(O(NJ^3 + N|\delta^{(n)}|J^3)\). Finally, updating all \(A^{(n)}\) takes \(O(NIJ^3 + N^2|\delta^{(n)}|J^3)\). According to (5), reconstruction (line 10) takes \(O(N|\delta^{(n)}|JN)\). Thus, the time complexity of GIFT is \(O(NIJ^3 + N^2|\delta^{(n)}|J^3)\).

Theorem 4 (Memory complexity of GIFT). The memory complexity of GIFT is \(O(TJ^3)\).

Proof. The intermediate data of GIFT consist of two vectors \(\delta\) and \(e_i^{(n)}\) \((\in \mathbb{R}^J)\), and two matrices \(B^{(n)}\) and \(\{B^{(n)} + \lambda D^{(n)}\}^{-1}\) \((\in \mathbb{R}^{J \times J})\). Memory spaces for those variables are released after updating the \(i_{th}\) row of \(A^{(n)}\). Thus, they are not accumulated during the iterations. Since each thread has their own intermediate data, the total memory complexity of GIFT is \(O(TJ^3)\).

Theorem 5 (Convergence of GIFT). GIFT converges since (1) is bounded and decreases monotonically.

Proof. According to Theorem 1, the loss function (1) never increases since every update in GIFT minimizes it, and (1) is bounded by 0. Thus, GIFT converges.
2.3 Hyperparameter Selection
One of the strengths of GIFT is that there are only few hyperparameters to be manually adjusted. The major parameter is a regularization coefficient $\lambda$, and we selected $\lambda = 10$ through various experiments, as presented in Table 5 (bold indicates the best one). Our criteria for choosing $\lambda$ include 1) high interpretability 2) low test RMSE. Regarding interpretability, $\lambda = 100$ is the best choice since it clearly distinguishes masked and unmasked entries (refer to Figure 4). Although it provides more interpretable results, it is hard to reveal important masked entries when $\lambda = 100$ as masked entries have too small values due to high penalties. Meanwhile, in the case of test RMSE, $\lambda = 1$ records the lowest value. Thus, we choose the middle of the parameters as $\lambda = 10$ since it has high interpretability and almost the same test RMSE to $\lambda = 1$. We note that it is not straightforward to make the optimization process automatic as interpretability is hard to be derived in a numerical format. In the case of rank, we choose the best one $30 \times 50 \times 2$ considering running time and accuracy.

2.4 Significant Factor Value
Significance of a gene factor value is chosen based on distribution of the factor values. Fig. 3 shows gene factor value distribution of unmasked entries (gene set members) and masked entries solved via GIFT. Many unmasked entries had significant factor values ($\geq 8$ or $\leq -8$) and majority of the masked entries had insignificant factor values. This shows that GIFT has learned the latent relationships of cancer patients to gene sets and significant genes in the gene set by encoding prior knowledge during its training. P-Tucker, on the other hand, produces a gene factor matrix with value distribution that has small or no correlation to a gene set. (refer to Supplementary Figure 3 for gene factor distribution of Silenced-TF and P-Tucker).

2.5 Scalability
We vary the number of observable entries by randomly sampling 20%, 40%, 60%, 80%, and 100% from the PANCAN12 tensor. As shown in Figs 5, P-Tucker (A) and Silenced-TF (B) scale near linearly in terms of the number of observable entries.

Figure 6 shows total running time of GIFT in terms of density of a tensor. GIFT shows linear scalability regarding the number of non-zeros.
References

