Tensor Decompositions Explainer

Ibrohim Nosirov, Julian Bellavita

Basics and Notation

This section covers the subset of basic tensor notation that is necessary to understand the rest of this document. Throughout the section, let $\mathcal{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ be an order d tensor with mode j of size n_j. $\mathcal{X}$ is indexed using a d-tuple of indices $(i_1 \ldots i_d)$. The Frobenius norm generalizes in a straightforward manner to tensors, meanining

$$\|\mathcal{X}\|_F = \sum_{i_1 \ldots i_d}^{n_1 \ldots n_d} \sqrt{\mathcal{X}_{i_1,\ldots,i_d}^2}$$

Slices and Fibers

A slice of a tensor is a subset of tensor entries with one fixed index. For example, an i_j slice of $\mathcal{X}$ is given by

$$\mathcal{X}_{:\ldots:i_j:\ldots:} \in \mathbb{R}^{n_1 \times \cdots \times n_{j-1} \times n_{j+1} \times \ldots n_d}$$

A tensor fiber is a subset of tensor entries with all but one fixed index. If we consider the so-called "mode i_j fibers" of $\mathcal{X}$, there are $\prod_{k \neq j} n_k$ mode-i_j fibers, and each is a vector in $\mathbb{R}^{n_j}$ uniquely identified by a $(d-1)$-tuple of indices. Formally, each mode-i_j fiber of $\mathcal{X}$ is given by

$$\mathcal{X}_{i_1 \ldots i_{j-1} : i_{j+1} \ldots i_d} \in \mathbb{R}^{n_j}$$

Matricizations/Unfoldings

Oftentimes it is necessary to store and perform some operation on all mode-k fibers of a tensor. In such scenarios, the so-called *matricization* operation is useful. Informally, the mode-k matricization of $\mathcal{X}$ 'unfolds' the entries of $\mathcal{X}$ into a matrix with one column per mode-k fiber of $\mathcal{X}$. This essentially transforms the tensor into a matrix which can be analyzed using classical numerical linear algebra methods. Formally, the mode-k matricization of $\mathcal{X}$ is given by $\mathbf{X}_{(k)}$ and is a matrix with n_k rows and $\prod_{j \neq k} n_j$ columns.

Special Matrix Products

Kronecker Products

The Kronecker product of two matrices $A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^{n \times n}$ is defined as

$$A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \dots & A_{1n}B \\ A_{21}B & A_{22}B & \dots & A_{2n}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1}B & A_{n2}B & \dots & A_{nn}B \end{bmatrix}$$

Essentially, the Kronecker product multiplies each entry of A with the entirety of B and stores the resulting matrix in a single block of the output.

Hadamard Products

The Hadamard product of $A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^{n \times n}$ is the elementwise product of the two matrices. Formally, we have

$$A * B = \begin{bmatrix} A_{11}B_{11} & A_{12}B_{12} & \dots & A_{1n}B_{1n} \\ A_{21}B_{21} & A_{22}B_{22} & \dots & A_{2n}B_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1}B_{n1} & A_{n2}B_{n1} & \dots & A_{nn}B_{nn} \end{bmatrix}$$

Khatri-Rao Products

The Khatri-Rao product is a column-wise Kronecker product of two matrices $A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^{n \times n}$. The output is therefore a tall-skinny matrix, with the number of rows being extremely large. Formally, the Khatri-Rao product is given by

$$A \odot B = \begin{bmatrix} A_{:1}B_{:1} & A_{:2}B_{2:} & \dots & A_{:n}B_{:n} \end{bmatrix}$$

Tensor Times Matrix Product

Tensors can be multiplied with matrices using an operation with similar logic to classical matrix multiplication. The so-called *Tensor Times Matrix Product* of a tensor $\mathcal{X}$ and a matrix $U \in \mathbb{R}^{m \times n_k}$ computes an inner product between each mode-k fiber of $\mathcal{X}$ and all rows of U to produce an output tensor $\mathcal{Y} \in \mathbb{R}^{n_1 \times \cdots \times n_{k-1} \times m \times n_{k+1} \times \ldots n_k}$. Formally, this is given by

$$\mathcal{Y}_{i_1 \ldots i_{k-1}, j, i_{k+1} \ldots i_n} = \sum_{i_k}^{n_k} \mathcal{X}_{i_1 \ldots i_n} U_{j, i_k}$$

This operation can be written in two ways: in terms of normal tensors and in terms of matricized tensors

$$\mathcal{Y} = \mathcal{X} \times_k U \iff \mathbf{Y}_k = U \mathbf{X}_k$$

Tucker Decomposition

The Tucker Decomposition is a generalization of the notion of a low-rank approximation to tensors. Given a target rank r, the low-rank approximation problem for a matrix $A \in \mathbb{R}^{m \times n}$ can be formulated as

$$\min_{X \in \mathbb{R}^{m \times r}, Y \in \mathbb{R}^{n \times r}} \|A - XY^T\|_F^2, \text{rank}(X) = r, \text{rank}(Y) = r$$

and it is solved via the truncated SVD. The Tucker Decomposition generalizes this idea by decomposing a tensor $\mathcal{X}$ into a smaller *core tensor*, usually denoted $\mathcal{G} \in \mathbb{R}^{r_1 \times \cdots \times r_d}$, and a set of d factor matrices $U_1 \ldots U_d$, where $U_i \in \mathbb{R}^{n_i \times r_i}$, and the columns of each factor matrix are orthonormal. $(r_1 \ldots r_d)$ is referred to as the the *multirank* of the Tucker decomposition. Formally, the Tucker Decomposition is given by

$$\mathcal{X} = \mathcal{G} \times_1 U_1 \times_2 \cdots \times_d U_d$$

The main benefit of the Tucker Decomposition is that it allows a given tensor $\mathcal{X}$ to be stored with a reduced memory footprint, since the core tensor and factor matrices are typically much smaller than the original tensor.

Although it is possible to write down an *exact* Tucker decomposition for an arbitrary tensor, this requires an appropriate pre-facto choice of multirank, which is rarely feasible. In practical applications of Tucker, it is up to the user to specify the multirank of the decomposition, and the problem then becomes finding the optimal Tucker decomposition with the specified multirank across all such possible decompositions. This can be expressed as the following minimization problem

$$\min_{\mathcal{G} \in \mathbb{R}^{r_1 \times \cdots \times r_d}, U_1 \in \mathbb{R}^{n_1 \times r_1} \ldots U_d \in \mathbb{R}^{n_d \times r_d}} \|\mathcal{X} - \mathcal{G} \times_1 U_1 \times_2 \cdots \times_d U_d\|_F^2$$

The choice of multirank induces a tradeoff between compression and decomposition accuracy; smaller multiranks mean more compression, whereas larger multiranks mean more accurate decompositions. The compression ratio of a Tucker Decomposition with multirank $(r_1 \dots r_d)$ is given by

$$1 - \frac{\prod_{i=1}^{d} r_i}{\prod_{i=1}^{d} n_i}$$

and the error is given by

$$\frac{\|\mathcal{X} - \mathcal{G} \times_1 U_1 \cdots \times_d U_d\|_F^2}{\|\mathcal{X}\|_F^2}$$

There are two main algorithms for computing the Tucker decomposition given a particular multirank: **Higher-Order Singular Value Decomposition (HOSVD)** and **Higher-Order Orthogonal Iteration (HOOI)**. We will cover both in the following sections.

Higher-Order Singular Value Decomposition (HOSVD)

HOSVD was the first method proposed for computing the Tucker Decomposition by Tucker himself in his original paper (Tucker 1966) HOSVD sets the factor matrices $U_1 \dots U_d$ by computing a low-rank approximation of each mode-k unfolding of $\mathcal{X}$ via a truncated SVD. Then, the core tensor is computed using

$$\mathcal{G} = \mathcal{X} \times_1 U_1^T \cdots \times_d U_d^T$$

which is valid because the factor matrices are chosen to have orthonormal columns. Pseudocode depicting the approach is given below.

```
Algorithm: HOSVD
Input: Tensor X, multirank (r_1 ... r_d)
Output: Core tensor G, factor matrices U_1 ... U_d
1. For k in 1 to d do
2.     U,Sigma,V = SVD(X_k)
3.     U_k = U[:, 1:r_k]
4. G = X x U_1^T ... x U_d^T
5. return G, U_1, ..., U_d
```

Intuitively, the reason HOSVD works is because the columns of each factor matrix are optimal low-rank approximations of each unfolding, and the original tensor is reconstructed using a linear combination of these columns. HOSVD is quasi-optimal; given a multirank $(r_1 \dots r_d)$, it gives a Tucker Decomposition that is within a factor of $\sqrt{d}$ of the optimal Tucker Decomposition with this multirank.

Higher-Order Orthogonal Iteration (HOOI)

HOOI is an iterative algorithm that uses an alternating least-squares (ALS) approach to optimize the factor matrices one at a time. Letting all but the kth factor matrix be fixed, we can compute the kth factor matrix by solving the following least squares problem

$$\min_{U_k \in \mathbb{R}^{n_k \times r_k}} \|(U_d \otimes \dots U_{k+1} \otimes U_{k-1} \otimes \cdots \otimes U_1)\mathbf{G}_k^T U_k^T - \mathbf{X}_k^T\|_F^2$$

which is equivalent to

$$\min_{U_k \in \mathbb{R}^{n_k \times r_k}} \|\mathcal{G} \times_1 U_1 \cdots \times_d U_d - \mathcal{X}\|_F^2$$

We give a more formal argument for the efficacy of HOOI below, which is largely a sketch of the argument found in Chapter 6 of (Ballard and Kolda 2025)

Let $\mathcal{T} = \mathcal{G} \times_1 U_1 \cdots \times_d U_d$. By Proposition 6.1 in (Ballard and Kolda 2025), the Tucker Decomposition minimization problem formulation can be written as

$$\min_{\mathcal{T}} \|\mathcal{T} - \mathcal{X}\|_F^2 = \|\mathcal{X}\|_F^2 - \|\mathcal{X} \times_1 U_1^T \cdots \times_d U_d^T\|_F^2$$

Since $\mathcal{X}$ is fixed, the above problem is equivalent to the following maximization problem

$$\max_{U_1 \in \mathbb{R}^{n_1 \times r_1} \dots U_d \in \mathbb{R}^{n_d \times r_d}} \|\mathcal{X} \times_1 U_1^T \cdots \times_d U_d^T\|_F^2$$

Solving this problem for all d factor matrices simultaneously is quite difficult, but solving it for a *single* factor matrix at a time is feasible. Therefore, we define the following simpler maximization problem, assuming all but the kth factor matrix is given

$$\max_{U_k \in \mathbb{R}^{n_k \times r_k}} \|\mathcal{X} \times_1 U_1^T \cdots \times_d U_d^T\|_F^2$$

This is equivalent to

$$\max_{U_k \in \mathbb{R}^{n_k \times r_k}} \|U_k^T \mathbf{X}_k (U_d \otimes \dots U_{k+1} \otimes U_{k-1} \otimes \cdots \otimes U_1)\|_F^2 = \max_{U_k \in \mathbb{R}^{n_k \times r_k}} \|U_k^T Y\|_F^2$$

This problem is solved by setting U_k to be the r_k leading left singular vectors of Y. This yields the HOOI algorithm shown in the pseudocode below, where the kth factor matrix is set to be the leading left singular vectors of the SVD of the mode-k unfolding of the input tensor $\mathcal{X}$ multiplied with the Kronecker product chain of all but the kth factor matrix. A single iteration of HOOI updates all d factor matrices. The convergence criteria for HOOI is usually based on the error of the Tucker Decomposition given by the factor matrices and core tensor at a given iteration.

```
Algorithm: HOOI
Input: Tensor X, multirank (r_1 ... r_d)
Output: Core tensor G, factor matrices U_1 ... U_d
1. U_1 ... U_d = HOSVD(X, r_1 ... r_d) # Initialize factor matrices
2. While not converged do # Main loop
3.     For k in 1 to d do
4.         Y = X x U_1^T ... U_k-1 x U_k+1 x ... x U_d # Multiply tensor with factors
5.         U_k = LLSV(Y_k, r_k) # Leading left singular vectors of Y unfolding
6.     end
7.     check convergence based on Tucker error
8. end
9. G = X x U_1 ... x U_d
10.return G, U_1, ..., U_d
```

HOOI requires an initial guess for the factor matrices prior to beginning the main iterations of the algorithm. Theoeretically, one can choose to initialize the factor matrices in any arbitrary manner, as long as they have orthogonal columns. However, in practice, a good strategy is to use HOSVD to obtain an initial guess for the factor matrices. As will be shown in the numerical experiments, HOOI typically converges quite quickly, and achieves errors smaller than HOSVD.

Numerical Experiments

Here, we present basic experiments demonstrating the convergence behavior of HOOI and HOSVD. We run experiments on the EEM tensor, just as in the CP section. We run HOOI for 5 iterations with three different multiranks: $(9, 100, 10), (9, 50, 10), (4, 50, 5)$. The factor matrices are initialized using HOSVD. At each iteration, the relative error of the decomposition is computed using the previously given expression for computing the error of a Tucker Decomposition.

Figure 1 shows the relative error of HOOI at each iteration for the three multiranks considered. Note that the error reported at iteration 0 corresponds to the error produced by using the factor matrices given by HOSVD, without running HOOI. Overall, HOOI is more accurate than HOSVD for all multiranks, as evidenced by the decreasing relative error. Additionally, HOOI converges quickly, with the relative error flattening out after only a single iteration of HOOI. The errors given by the multiranks $(9, 100, 10)$ and $(9, 50, 10)$ are close, whereas the error given by the other multirank $(4, 50, 5)$ is much larger than the other two relative errors. This indicates that there is less of a low-rank structure in the mode 1 and 3 unfoldings, whereas there is more redundancy in the mode 2 unfolding.

Figure 2 shows the compression rate achieved by the Tucker Decomposition with each of the three multiranks. Overall, the compression rates range from around 65% to 85%. Finally,

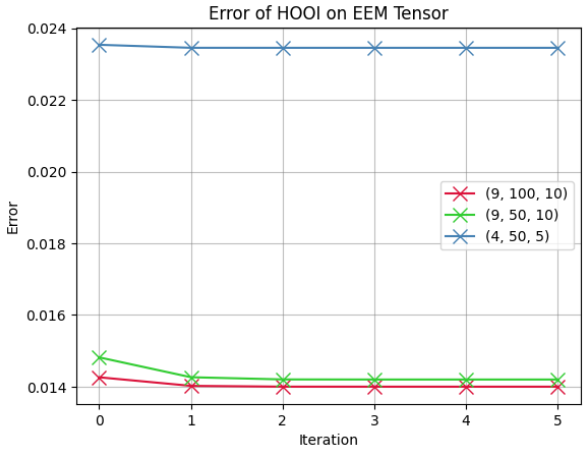

Figure 1: Tucker Decomposition accuracy on EEM tensor

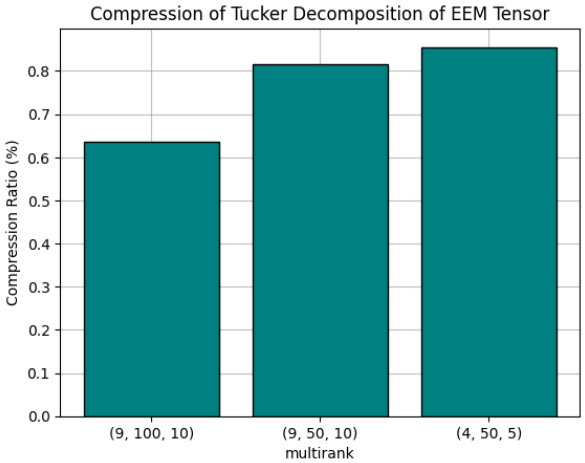

Figure 2: Compression achieved by Tucker Decomposition of EEM Tensor

the multirank that seems to achieve the best balance between compression and accuracy is $(9, 50, 10)$.

References

Ballard, Grey, and Tamara G. Kolda. 2025. *Tensor Decompositions for Data Science*. Cambridge University Press. https://www.cambridge.org/core/books/tensor-decompositions-for-data-science/640814D308696CD61CB9112EA57B2911.

Tucker, Ledyard R. 1966. "Some Mathematical Notes on Three-Mode Factor Analysis." *Psychometrika* 31 (3): 279–311.