Tensors in Data Analysis

Winter Semester 17/18
Block course

Dr. Pauli Miettinen
Tensors in Data Analysis

Winter Semester 2017/2018

9.-13. October

Exam: 20 October. Oral exam; time & place are announced later. Registration to HISPOS by 13 October. Exam material: these lecture notes & Kolda and Bader's "Tensor Decompositions and applications" (available from the course web page).

Tutorials: 4 problem sheets, 4 questions per sheet. To mark a problem solved, you must be present at the tutorial the whole time and be willing to present your solution. You must mark at least 8 problems to be allowed to take the final exam. There are no bonus points, but trying to solve all problems and attending all tutorials is strongly recommended.
Lectures: Lectures take place 10:15 - 11:45 and 12:30 - 14:00 every day. Tutorials follow at 14:15 through 15:30 Tuesday to Friday. (On Tuesday, the tutorial session starts at 14:30.) Lecture notes are made available following the day's lectures.

Topics: A tentative list of topics is

Monday: Tensor algebra and manipulations; products with matrices and vectors

Tuesday: CP decomposition; algorithms; tensor rank

Wednesday: Variations of the CP; applications thereof; Tucker3 decomposition

Thursday: Variations of the Tucker; applications thereof; tensor train decomposition

Friday: Applications; CORCONDIA
What is a Tensor?

Tensors are:

- Multi-linear mappings
  \[ T: \mathbb{R}^{i_1} \times \mathbb{R}^{i_2} \times \cdots \times \mathbb{R}^{i_n} \rightarrow \mathbb{R} \]

- Multi-dimensional arrays
  \[ T \in \mathbb{R}^{i_1} \times \mathbb{R}^{i_2} \times \cdots \times \mathbb{R}^{i_n} \]
  \[ T = (t_{i_1 i_2 \cdots i_n}) \]

- Generalizations of matrices
- N-ary relations
- Cubes

- Matrices and vectors are also tensors
Terminology

We say that a tensor is \textit{N-way} array. A matrix is a 2-way array, a vector is a 1-way array. Other sources can use \textit{N-dimensional} instead, but then a 3-dimensional vector is a 1-dimensional tensor. Yet others say rank-\(n\), but we have another meaning for the word "rank".

A 3-way tensor can be \textit{N-by-M-by-K} \textit{dimensional} (like vectors are \textit{n-dimensional}).

3-way

5-by-7-by-3
Modes and slices

A 3-way tensor has three modes, rows, columns, and tubes. Vectors along any of the modes are called fibres. Matrices along any two modes are called slices.

Mode-1 (column) fibres

Mode-2 (row) fibres

Mode-3 (tube) fibres

Horizontal slice

Lateral slice

Frontal slice
Why Tensors?

Matrices are a common way to store or interpret data in data analysis. One way to interpret matrices is to think of them as binary relations or functions. Think, for example, values of pixels in images or frequency of terms in documents.

Pixels

<table>
<thead>
<tr>
<th>Images</th>
<th>Pixels</th>
<th>Terms</th>
</tr>
</thead>
</table>

Tensors generalize this relation from binary to multi-ary. For example, if we want to store the values of different colours in images, we need three matrices, for red, green, and blue.
In hyperspectral imaging, the tensor would have more than just three colours.

Tensors can also be thought to contain many binary relations between the same entities. In Resource Definition Framework (ROF), all data is stored in subject-predicate-object (or s,p,o) triples. These triples can be seen as a 3-way binary tensor where each frontal slice has one predicate (i.e., relation between subjects and objects).
Another use of tensors is to store matrices (or tensors) over different discrete time stamps. For example, a series of adjacency matrices can be stored in a vertices-by-vertices-by-days tensor.

\[ \begin{array}{ccc}
V & V & V \\
& \text{time} & \\
& V & 
\end{array} \]

This is perhaps the most common way to "generalize" matrix-form data to tensors. Care must be taken, however, as most tensor techniques are invariant over permutations of the indices, that is, they don't take into account that time has a strict ordering.
Tensor Algebra

Indexing

Tensor $I \in \mathbb{R}^{l_1 \times l_2 \times \ldots \times l_n}$ has elements

$$I = (t_{i_1, i_2, \ldots, i_n})$$

We can write $i^\alpha = i_1 \alpha \ldots i_n$ to denote an index vector $i^\alpha \in \mathbb{I}_{l_1} \times \mathbb{I}_{l_2} \times \ldots \times \mathbb{I}_{l_n}$, where $\mathbb{I}_{l_n} = \{1, 2, 3, \ldots, l_n\}$.

The fibres are denoted $t_{i_1, i_2, \ldots, i_n}$. 

$$t_{i_1, i_2, \ldots, i_n}$$

$$t_{i_1, i_2, \ldots, i_n}$$

$$t_{i_1, i_2, \ldots, i_n}$$
The slices are denoted

\[ T_{1:i_1:i_2:i_3:k_1:k_2:k_3:i_4} \]

For 3-way tensors, the frontal slice \( T_{::i_3} \) has a special short-hand notation

\[ T_{::i_3} = T_{i_3} \]

Matricization

A tensor \( T \) can be matricized or unfolded or flattened in a matrix by reordering its elements. Mode-n matricization takes the mode-n fibres and stacks them as the columns of
the new matrix. If $T \in \mathbb{R}^{l_1 \times l_2 \times \cdots \times l_n}$, its mode-$1$ matricization is a matrix $T_{(1)} \in \mathbb{R}^{l_1 \times (l_2 \cdots l_n)}$, and in general, the mode-$n$ matricization is a matrix $T_{(n)} \in \mathbb{R}^{l_n \times (l_1 \cdots l_{n-1} l_{n+1} \cdots l_N)}$.

In mode-$n$ matricization, element at $(i_1, i_2, \ldots, i_N)$ maps to $(i_n, j)$, where

$$j = 1 + \sum_{k=1}^{n-1} (i_k - 1) J_k,$$

where

$$J_k = \prod_{m=1}^{n-1} l_m \quad \text{for} \quad m \neq n.$$

An example (next page) will clarify the process.
MATRIFICATION EXAMPLE

Let $\mathbf{T} \in \mathbb{R}^{4 \times 3 \times 2}$ with frontal slices

$$
\mathbf{T}_1 = \begin{pmatrix}
1 & 5 & 9 \\
2 & 6 & 10 \\
3 & 7 & 11 \\
4 & 8 & 12
\end{pmatrix}
\quad \text{and} \quad
\mathbf{T}_2 = \begin{pmatrix}
13 & 17 & 21 \\
14 & 18 & 22 \\
15 & 19 & 23 \\
16 & 20 & 24
\end{pmatrix}.
$$

Now we have:

$$
\mathbf{T}_{(1)} = \begin{pmatrix}
1 & 5 & 9 & 13 & 17 & 21 \\
2 & 6 & 10 & 14 & 18 & 22 \\
3 & 7 & 11 & 15 & 19 & 23 \\
4 & 8 & 12 & 16 & 20 & 24
\end{pmatrix},
$$

$$
\mathbf{T}_{(2)} = \begin{pmatrix}
1 & 2 & 3 & 4 & 13 & 14 & 15 & 16 \\
5 & 6 & 7 & 8 & 17 & 18 & 19 & 20 \\
9 & 10 & 11 & 12 & 21 & 22 & 23 & 24
\end{pmatrix},
$$

$$
\mathbf{T}_{(3)} = \begin{pmatrix}
1 & 2 & 3 & 4 & 5 & \cdots & 9 & 10 & 11 & 12 \\
13 & 14 & 15 & 16 & 17 & \cdots & 21 & 22 & 23 & 24
\end{pmatrix}.
$$

VECTORIZATIONS

A vectorization of a tensor stacks the columns of its mode-1 matrification in a column vector. For $\mathbf{T}$ as above, we have $\text{vec}(\mathbf{T}) = (1, 2, 3, \ldots, 24)^T$. 

Tensor times a scalar, $a\mathcal{I}$, scales every element of $\mathcal{I}$ by $a \in \mathbb{R}$, 
\[ a\mathcal{I} = (ax) \]

The n-mode vector-tensor product of a tensor $\mathcal{I} \in \mathbb{R}^{l_1 \times l_2 \times \ldots \times l_N}$ with vector $\mathcal{v} \in \mathbb{R}^l$ is
\[ \mathcal{I} \times^*_n \mathcal{v} \in \mathbb{R}^{l_1 \times l_2 \times \ldots \times l_{n-1} \times l_{n+1} \times \ldots \times l_N} \]

\[ (\mathcal{I} \times^*_n \mathcal{v})_{i_1 \ldots i_{n-1} i_n i_{n+1} \ldots i_N} = \sum_{i_n = 1}^l t_{i_1 i_2 \ldots i_N} v_i \]

\[ = \left[ \langle t_{i_1 \ldots i_{n-1} i_n \ldots i_N}, \mathcal{v} \rangle \right]_{i_1 \ldots i_{n-2} i_{n+1} \ldots i_N} \]

where $\langle \cdot , \cdot \rangle$ is the inner product of two vectors. The idea is to take the inner products between all mode-n fibres and \( \mathcal{v} \).

In tensor-vector products, precedence matters:
\[ \mathcal{I} \times_m \mathcal{a} \times_n \mathcal{b} = (\mathcal{I} \times_m \mathcal{a}) \times_{n-m} \mathcal{b} = (\mathcal{I} \times_n \mathcal{b}) \times_{m-n} \mathcal{a} \]
If $J$ is 2-way (a matrix), we have
\[ J \bar{x}_1 \vec{v} = \vec{v}^T \bar{I} \] and
\[ J \bar{x}_2 \vec{v} = \bar{I} \vec{v}. \]

Let $J \in \mathbb{R}^{4 \times 3 \times 2}$ with frontal slices
\[ T_1 = \begin{pmatrix} 7 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{pmatrix} \quad \text{and} \quad T_2 = \begin{pmatrix} 13 & 17 & 21 \\ 14 & 18 & 22 \\ 15 & 19 & 23 \\ 16 & 20 & 24 \end{pmatrix}, \]
and let $\vec{v} = (5, 10)$. Then
\[
J \bar{x}_3 \vec{v} = \begin{pmatrix} 5 + 130 & 35 + 170 & 45 + 210 \\ 10 + 140 & 30 + 180 & 50 + 220 \\ 15 + 150 & 35 + 190 & 55 + 230 \\ 20 + 160 & 40 + 200 & 60 + 240 \end{pmatrix} = \begin{pmatrix} 135 & 195 & 255 \\ 150 & 210 & 270 \\ 165 & 225 & 285 \\ 180 & 240 & 300 \end{pmatrix}.
\]
The $n$-mode matrix-tensor product of tensor $I \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_n}$ and matrix $M \in \mathbb{R}^{J_1 \times I_1}$ is

$$I \times_n M \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J_1 \times I_{n+1} \times \cdots \times I_n}$$

$$(I \times_n M)_{i_1 \ldots i_{n-2}, j, i_{n+1}, \ldots, i_n}^{} = \sum_{i_n = 1}^{I_n} a_{i_1 \ldots i_{n-1}, j, i_n}^{} m_{i_n}^{}$$

$$= \left[ M_{i_1 \ldots i_{n}, i_{n+1} \ldots i_n}^{} \right]_{i_1 \ldots i_n}^{}$$

That is, we multiply each mode-$n$ fibre with $M$. Equivalently, we can use unfolding:

$$S = I \times_n M \iff S_{(n)} = M T_{(n)}.$$

If $I$ is 2-way (i.e. a matrix), $T_{(1)} = I$ and $T_{(2)} = I^T$. So

$$I \times_1 M = M I \quad \text{and} \quad I \times_2 M = M I^T$$
The order of tensor-matrix multiplications over different modes doesn't matter: if \( m \neq n \), then

\[
J \times_m A \times_n B = J \times_n B \times_m A.
\]

If the modes are the same and \( A \in \mathbb{R}^{j \times j} \) and \( B \in \mathbb{R}^{j \times j} \), then

\[
J \times_n A \times_n B = J \times_n (BA).
\]

Let \( J \in \mathbb{R}^{4 \times 3 \times 2} \) with frontal slices

\[
T_1 = \begin{pmatrix}
7 & 5 & 9 \\
2 & 6 & 10 \\
3 & 7 & 11 \\
4 & 8 & 12
\end{pmatrix}
\quad \text{and} \quad
T_2 = \begin{pmatrix}
13 & 17 & 21 \\
14 & 18 & 22 \\
15 & 19 & 23 \\
16 & 20 & 24
\end{pmatrix},
\]

and let

\[
M = \begin{pmatrix}
1 & 2 & 4 & 6 \\
10 & 20 & 40 & 60
\end{pmatrix}
\]

Then, if \( S = J \times_1 M \), we have

\[
S_{111} = 1 \cdot 1 + 2 \cdot 2 + 4 \cdot 3 + 6 \cdot 4 = 41
\quad \text{and}
\]

\[
S_1 = \begin{pmatrix}
41 & 93 & 145 \\
910 & 930 & 1470
\end{pmatrix}
\quad \text{and} \quad
S_2 = \begin{pmatrix}
197 & 299 & 301 \\
1970 & 2490 & 3010
\end{pmatrix}
\]
The tensor inner product of two tensors $\mathbf{S}, \mathbf{T} \in \mathbb{R}^{l_1 \times l_2 \times \cdots \times l_N}$ is the sum of their element-wise products,

$$\langle \mathbf{S}, \mathbf{T} \rangle = \sum_{i_1=1}^{l_1} \sum_{i_2=1}^{l_2} \cdots \sum_{i_N=1}^{l_N} S_{i_1i_2\cdots i_N} T_{i_1i_2\cdots i_N}.$$

Tensor norm

The norm of a tensor $\mathbf{I} \in \mathbb{R}^{l_1 \times l_2 \times \cdots \times l_N}$ is the square root of the sum of the squares of its elements:

$$\| \mathbf{I} \| = \left( \sum_{i_1=1}^{l_1} \sum_{i_2=1}^{l_2} \cdots \sum_{i_N=1}^{l_N} I_{i_1i_2\cdots i_N}^2 \right)^{1/2}.$$

This can be alternatively be defined as $\sqrt{\langle \mathbf{I}, \mathbf{I} \rangle}$, $\| \mathbf{T}_{(i)} \|_F$, $\| \mathbf{T}_{(i)} \|_F$, $\| \text{vec}(\mathbf{I}) \|_2$, or $\text{trace} \left( \mathbf{T}_{(i)} \mathbf{T}_{(i)}^T \right)^{1/2}$, or using any other way to define the Euclidean/Frobenius norm of a vector/matrix.
Symmetry

Tensor $I$ is **cubical** if all of its modes have the same dimensionality:

$$I \in \mathbb{R}^{1 \times 1 \times 1 \times \cdots \times 1}$$

Cubical tensor is **(super-) symmetric**, if its elements remain constant under any permutation of the indices. If $I \in \mathbb{R}^{1 \times 1 \times 1}$, it is symmetric if and only if

$$t_{ijk} = t_{ikj} = t_{jki} = t_{kij} = t_{kj} = t_{ji}; \text{ for all } i, j, k \in [1]$$

Tensor $I \in \mathbb{R}^{1 \times 1 \times 1 \times \cdots \times 1}$ is (hyper-) diagonal if

$$t_{i_1 i_2 \cdots i_n} \neq 0 \text{ only if } i_1 = i_2 = \cdots = i_n.$$  If $i_1 = i_2 = \cdots = i_n$, $I$ is also symmetric.

If the diagonal entries are all 1s, tensor behaves similarly to the identity matrix.
The CP Decomposition and the Rank of a Tensor

Vector outer product of $N$ vectors $\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(N)}$ is an $N$-way tensor

$$\mathbf{T} = \mathbf{a}^{(1)} \mathbf{a}^{(2)} \cdots \mathbf{a}^{(N)}$$

with every element defined as the product of the corresponding elements of the vectors

$$t_{i_1 i_2 \ldots i_N} = a_{i_1}^{(1)} a_{i_2}^{(2)} \cdots a_{i_N}^{(N)}.$$
The CP decomposition

The exact CP decomposition of an $N$-way tensor $I \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ has the form

$$I = \sum_{r=1}^{R} \tilde{a}^{(1)} \circ \tilde{a}^{(2)} \circ \cdots \circ \tilde{a}^{(N)}$$

where $R \in \mathbb{N}$ and $\tilde{a}^{(i)} \in \mathbb{R}^{I_i}$ for all $i \in [N]$ and $r \in [R]$.

In the approximate (or fixed-rank) CP decomposition, the size $R$ is given, and we're looking for the least-error decomposition

$$|| I - \sum_{r=1}^{R} \tilde{a}^{(1)} \circ \tilde{a}^{(2)} \circ \cdots \circ \tilde{a}^{(N)} ||.$$

For now, we concentrate on 3-way tensors, and write

$$I = \sum_{r=1}^{R} \tilde{a}^{(1)} \circ \tilde{b}^{(2)} \circ \tilde{c}^{(3)}.$$
Visually, the 3-way CP is

\[ T = \sum_{k} \alpha_k \otimes \beta_k \otimes \gamma_k + \ldots \]

We can gather the vector for each mode in factor matrices. In the 3-way setting, for \( T \in \mathbb{R}^{1 \times J \times K} \), we have

\[
A = [\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_R]_{1 \times R}, \quad B = [\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_J]_{J \times R}, \quad C = [\vec{c}_1, \vec{c}_2, \ldots, \vec{c}_K]_{K \times R}
\]

We can express the 3-way CP decomposition using the frontal slices of \( T \) and the factor matrices:

\[
T_k = A \Pi^{(k)} B^T,
\]

where \( \Pi^{(k)} = \text{diag}(C_{(k,:)}), \) i.e. a diagonal matrix with the \( k \)-th row of \( C \) on its diagonal.
The frontal slice formulation doesn't generalize easily for more than 3 modes. For more generalized representation, we need the Khatri-Rao matrix product: given matrices $A \in \mathbb{R}^{j \times lc}$ and $B \in \mathbb{R}^{j \times lc}$, their Khatri-Rao product is

$$A \odot B = \begin{pmatrix}
    a_{11}b_1 & a_{12}b_2 & \cdots & a_{1k}b_k \\
    a_{21}b_1 & a_{22}b_2 & \cdots & a_{2k}b_k \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{l1}b_1 & a_{l2}b_2 & \cdots & a_{lk}b_k
\end{pmatrix} \in \mathbb{R}^{lj \times lc}.$$ 

That is, each column of $B$ is copied 1 times, and the $i$-th copy of the $k$-th column of $B$ multiplied by $a_{ik}$. The Khatri-Rao product can be written more concisely using the
The Kronecker product \( A \otimes B \), if \( A \in \mathbb{R}^{l \times j} \) and \( B \in \mathbb{R}^{k \times l} \), is defined to be the block matrix:

\[
A \otimes B = \begin{pmatrix}
    a_{11}B & a_{12}B & \cdots & a_{1j}B \\
    a_{21}B & a_{22}B & \cdots & a_{2j}B \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{l1}B & a_{l2}B & \cdots & a_{lj}B
\end{pmatrix} \in \mathbb{R}^{(lj) \times (jl)}.
\]

Notice that in the Kronecker product, the matrices can be of arbitrary size, whereas in the Khatri-Rao product, they must have the same number of columns.

The Khatri-Rao product of \( A \in \mathbb{R}^{l \times jk} \) and \( B \in \mathbb{R}^{j \times k} \) can now be written as:

\[
A \bowtie B = [\hat{a}_1 \otimes \hat{b}_1, \hat{a}_2 \otimes \hat{b}_2, \ldots, \hat{a}_{lj} \otimes \hat{b}_{lk}],
\]

that is, Khatri-Rao is "column-wise Kronecker" product.

If \( A \in \mathbb{R}^{l \times k} \), \( B \in \mathbb{R}^{j \times k} \), and \( C \in \mathbb{R}^{k \times k} \) are the factor matrices of a CP decomposition of tensor \( I \in \mathbb{R}^{l \times j \times k} \), then...
$T_{(1)} = A (C(0) B)^T$

$T_{(2)} = B (C(0) A)^T$

$T_{(3)} = C (B O A)^T$.

More generally, if $T$ has $N$ modes and factor matrices $A^{(1)}, A^{(2)}, \ldots, A^{(n)}$,

$T_{(n)} = A^{(n)} (A^{(n)} O \ldots O A^{(n-1)} O \ldots O A^{(1)})^T$.

To gain intuition on the Khatri-Rao formulation, consider the frontal slice formulation of CP:

$T_k = A D^{(k)} B^T$

with $D^{(k)} = \text{diag}(C(:,k))$. The same factor $A$ appears with all frontal slices, so we can just stack them:

$[T_1, T_2, \ldots, T_K] = A \underbrace{[D^{(1)} B^T, D^{(2)} B^T, \ldots, D^{(K)} B^T]}_E$
The first row of $E$ has the first column of $B$ multiplied by $c_{11}$ followed by the first column of $B$ multiplied by $c_{21}$, and so on. Hence
\[ \vec{e}_1 = [c_{11} b_1, c_{21} b_1, \ldots, c_{k1} b_1] = (\overrightarrow{c_{10}} \otimes \vec{b}_1)^T. \]

Extending this to all rows of $E$ we see that
\[ E^T = (C \otimes B)^T \]

and hence
\[ T_{(1)} = \begin{bmatrix} T_1 & T_2 & \cdots & T_K \end{bmatrix} = A (C \otimes B)^T. \]

The connections in the other modes can be derived analogously.
One sometimes normalizes the columns of the factor matrices to unit length. The lengths are then stored in factors $\lambda_r = \|\overrightarrow{a}_r\|, \|\overrightarrow{b}_r\|, \|\overrightarrow{c}_r\|$, collected in a vector $\overrightarrow{\lambda} \in \mathbb{R}^r$, or in a matrix $\Lambda = \text{diag}(\overrightarrow{\lambda}) \in \mathbb{R}^{r \times r}$. Then $T_{(n)} = \Lambda (\overrightarrow{c} \otimes \overrightarrow{b})^T$ etc. and

$$ I = \sum_{r=1}^{R} \lambda_r \overrightarrow{a}_r \otimes \overrightarrow{b}_r \otimes \overrightarrow{c}_r. $$

A common notation for the CP decomposition is to write

$$ I = [A, B, C] = \sum_{r=1}^{R} \overrightarrow{u}_r \overrightarrow{b}_r \overrightarrow{c}_r, $$

or with the scaling

$$ I = \sum_{r=1}^{R} \lambda_r \overrightarrow{a}_r \overrightarrow{b}_r \overrightarrow{c}_r. $$
ALD algorithm for CP

The formulations

\[ T_{(n)} = A (C \Omega B)^T \]

provide a way to solve the (approximate) CP decomposition. When \( C \) and \( B \) are fixed \((C \Omega B)\) is a fixed matrix, call it \( D \), and the problem becomes: "Given matrices \( T_{(n)} \) and \( D \), find matrix \( A \) that minimizes \( \| T_{(n)} - AD^T \|_F \)." This can be solved using the SVD and pseudo-inverse as \( A = T_{(n)} (D^T)^+ \), where \((D^T)^+\) is the Moore-Penrose pseudo-inverse. This leads to the following algorithm:

\begin{verbatim}
    sample random 13 and C
    repeat
        let \( A \leftarrow T_{(n)} (C (13B)^T)^+ \)
        let \( B \leftarrow T_{(3)} (C (13A)^T)^+ \)
        let \( C \leftarrow T_{(3)} (13 (13C A)^T)^+ \)
    until convergence
\end{verbatim}
The ALS algorithm requires us to compute the pseudo-inverses of \((COB)^T\), \((COA)^T\), and \((BGA)^T\), which are \(R\)-by-\(JK\), \(R\)-by-\(1K\), and \(R\)-by-\(1J\) matrices, respectively. This is an expensive operation, but if these matrices have a full row rank—which is likely, as often \(R \ll \min\{1J, 1K, JK\}\)—then we can use the following equality:

\[
(AOB)^+ = ((AT)(B^T))^+(AOB)^T , \tag{*}
\]

where \(X \ast Y\) is the Hadamart matrix product (or element-wise product) between \(X \in \mathbb{R}^{1 \times J}\) and \(Y \in \mathbb{R}^{1 \times J}\):

\[
X \ast Y = \begin{pmatrix}
    x_{11}y_{11} & x_{12}y_{12} & \cdots & x_{1J}y_{1J} \\
    x_{21}y_{11} & x_{22}y_{12} & \cdots & x_{2J}y_{1J} \\
    \vdots & \vdots & \ddots & \vdots \\
    x_{11}y_{11} & x_{12}y_{12} & \cdots & x_{1J}y_{1J}
\end{pmatrix} \in \mathbb{R}^{1 \times J}
\]
The proof of identity (8) is left as a homework, but it involves the following identity that is also occasionally useful on its own:

For $X \in \mathbb{R}^{1 \times k}$ and $Y \in \mathbb{R}^{3 \times k}$, we have

$$(X \circ Y)^T (X \circ Y) = X^T X \ast Y^T Y.$$  

Proof: Let $X' = X^T X$ and notice that

$$x'_{ik} = \langle \vec{x}_i, \vec{x}_k \rangle.$$  

Similarly, for $Y' = Y^T Y$, we have

$$y'_{jk} = \langle \vec{y}_j, \vec{y}_k \rangle.$$  

Now, let

$$Z = (X \circ Y)^T (X \circ Y)$$

$$= \begin{bmatrix} x_1 \otimes \vec{y}_1 & \cdots & x_k \otimes \vec{y}_k \end{bmatrix}^T \begin{bmatrix} x_1 \otimes \vec{y}_1 & \cdots & x_k \otimes \vec{y}_k \end{bmatrix},$$

and consider a single element $Z_{kl}$:

$$Z_{kl} = \langle x_k \otimes \vec{y}_k, x_l \otimes \vec{y}_l \rangle = \sum_{i=1}^k \langle x_{ik} \otimes \vec{y}_k, x_{il} \otimes \vec{y}_l \rangle$$

$$= \sum_{i=1}^k x_{ik} x_{il} \langle \vec{y}_k, \vec{y}_l \rangle$$

$$= \langle x_k, x_l \rangle \langle \vec{y}_k, \vec{y}_l \rangle = x'_{kl} y'^{\dagger}_{kl},$$

and hence $Z = X' \ast Y' = X^T X \ast Y^T Y$. □
With equality (6) we can write

\[ A = T_{(i)} \left[(C \mathbf{O} \mathbf{B})^T\right]^+ \]

as

\[ A = T_{(i)} (C \mathbf{O} \mathbf{B}) \left[C^T C \mathbf{B} \mathbf{B}^T\right]^+ \]

and we only have to take the pseudo-inverse from a much smaller matrix. This formulation can, however, cause issues with the numerical stability.

ALS is not the only possibility. We can instead use, for instance, gradient-based methods: each row \( \vec{a}_i \) can be updated based on the gradient

\[ \vec{a}_i \leftarrow \vec{a}_i - \delta \frac{\partial}{\partial \vec{a}_i} \sum_{j=1}^{JK} \left( T_{(i)}(i, j) - (\vec{a}_i (C \mathbf{O} \mathbf{B})^T)_j \right)^2 \]

ALS is the most commonly used approach, though.
Tensor rank

An $N$-way tensor $I$ is rank-1 if it is an outer product of $N$ vectors

$$I = \overrightarrow{a}^{(1)} \circ \overrightarrow{a}^{(2)} \circ \ldots \circ \overrightarrow{a}^{(N)}.$$

Tensor $I$ has rank $R$ if it is a sum of $R$ rank-1 tensors (and no less), e.g.

$$I = \overrightarrow{a}_1 \circ \overrightarrow{b}_1 \circ \overrightarrow{c}_1 + \overrightarrow{a}_2 \circ \overrightarrow{b}_2 \circ \overrightarrow{c}_2 + \ldots + \overrightarrow{a}_R \circ \overrightarrow{b}_R \circ \overrightarrow{c}_R$$

Equivalently, the rank of a tensor $I$ is the least $R$ such that $I$ has exact CP decomposition to $R$ components. If $I$ is all-zero tensor, its rank is agreed to be 0.

Compare this definition with that of matrix rank. The vectors don't have
to be linearly independent, but the formulation is analogous to the so-called Schein rank of a matrix: the rank of a matrix $M$ is the least $R$ s.t. $M$ has decomposition $M = AB$ with $A$ having $R$ columns.

**Tensor rank oddities**

While seemingly similar to matrix rank, tensor rank behaves very differently in many cases.

**Computational complexity**

Matrix rank is easy to compute, save precision issues using, for instance, SVD. Tensor rank is NP-hard to compute. This also means that deciding whether a tensor has an exact rank-$R$ decomposition is NP-hard.
IR vs. C

Rank of a matrix $M \in \mathbb{R}^{1 \times j}$ is the same irrespective of whether we take the factorization over IR or C. With tensors, this is not the case. Consider

$$T_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad T_2 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.$$ 

This tensor has rank 3 over IR, but rank 2 over C. For example

$$A = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \end{pmatrix}, \quad \text{and } C = \begin{pmatrix} 1 & 1 & 1 \\ -1 & 1 & 1 \end{pmatrix}.$$

over IR, but over C we have

$$\hat{A} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ -i & i \end{pmatrix}, \quad \hat{B} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ i & -i \end{pmatrix}, \quad \text{and } \hat{C} = \begin{pmatrix} 1 & 1 \\ i & -i \end{pmatrix}.$$

Maximal rank

If $M \in \mathbb{R}^{1 \times j}$, we know that $\text{rank}(M) \leq \min\{1, j\}$. With tensors, this is not necessarily the case, as we see above. For $I \in \mathbb{R}^{1 \times j \times k}$, we only know the weak upper bound
\[ \text{rank}(J) \leq \min \{1, J, 1J\}. \]

**Typical rank.**

**Typical rank** is any rank that occurs with probability greater than zero if we sample over \( \mathbb{R}^{K \times L \times \cdots \times M} \). (Notice that this is not the same as sampling over the tensors represented by the floating point numbers — the set of all of those tensors has measure zero.) With matrices \( M \in \mathbb{R}^{1 \times J} \), typical rank is again \( \min \{1, J\} \), that is, all random matrices have a full rank. With tensors, this is not the case. For instance, tensors in \( \mathbb{R}^{2 \times 2 \times 2} \) have typical ranks 2 and 3 over \( \mathbb{R} \) (experiments suggest that about 79% of 2-by-2-by-2 tensors have rank 2 and 27% have rank 3; rank-1 tensors occur with zero probability).
Uniqueness of the rank decomposition

Matrix factorizations are generally not unique; if \( M = XY \), we can always have

\[ M = X'Y', \]

where \( X' = XZ \) and \( Y' = Z^\top Y \)

for some invertible \( Z \). SVD is unique only because of the orthogonality constraints and the scaling matrix \( \Sigma \).

CP decomposition, on the other hand, is often unique (up to self-canceling scaling and permutation of the factors). It is always possible to scale

\[ J = \sum_{r=1}^{R} (\alpha_r \vec{a}_r) \circ (\beta_r \vec{b}_r) \circ (\gamma_r \vec{c}_r), \]

provided that \( \alpha_r \beta_r \gamma_r = 1 \) for all \( r \in [R] \).

We can also permute the components

\[ J = \sum_{r=1}^{R} \vec{a}_{\sigma(r)} \circ \vec{b}_{\sigma(r)} \circ \vec{c}_{\sigma(r)} \]

for any permutation \( \sigma : [R] \to [R] \).
A sufficient condition for the uniqueness of the exact CP decomposition can be expressed using the concept of a \( k \)-rank: the \( k \)-rank of a matrix \( A \), denoted \( k_A \), is the largest \( k \) such that any \( k \) columns of \( A \) are linearly independent (cf. normal rank, that requires that some \( k \) columns are linearly independent). The condition for 3-way CP decomposition \( I = [A, B, C] \) is

\[
k_A + k_B + k_C \geq 2R + 2.
\]

As \( \max\{k_A, k_B, k_C\} \leq R \), it's enough that, e.g., \( A \) and \( B \) have full rank and \( C \) has \( k_C = 2 \).

For \( N \)-way tensors, the sufficient condition is

\[
\sum_{n=1}^{N} k_{A^n} \geq 2R + (N - 1).
\]

A necessary condition in 3-way case is

\[
\min \{ \text{rank}(A^2B), \text{rank}(AC), \text{rank}(BC) \} \geq R.
\]
Border rank

In approximate decompositions, the situation is reversed. The Eckart-Young theorem states that the best rank-$R$ approximation of a matrix is its rank-$R$ truncated SVD. This provides a hierarchy: best rank-$(R-1)$ factorization is a part of the best rank-$R$ approximation. CP decomposition doesn't have such hierarchy: the best rank-$1$ approximation might not be part of any higher-rank optimal approximations, for example.

The Eckart-Young theorem also shows that there's a clear difference between the best rank-$(R-1)$ and rank-$R$ decomposition:

$$||U_{R-1} \Sigma_{R-1} V_{R-1}^T - U_R \Sigma_R V_R^T||_F = \sigma_R \text{ for } \sigma \in \{2, F\}.$$
With tensors, it is possible to get arbitrarily close to the one higher rank decomposition. For an example, consider $I \in \mathbb{R}^{1 \times 1 \times 1}$ with

$$I = \overrightarrow{a} \circ \overrightarrow{b} \circ \overrightarrow{c} + \overrightarrow{a} \circ \overrightarrow{b} \circ \overrightarrow{c} + \overrightarrow{a} \circ \overrightarrow{b} \circ \overrightarrow{c},$$

where the columns of $A$, $B$, and $C$ are linearly independent. Hence $\text{rank}(I) = 3.$ Let

$$S = 2(\overrightarrow{a} + \frac{1}{2} \overrightarrow{b}) \circ (\overrightarrow{c} + \frac{1}{2} \overrightarrow{d}) \circ (\overrightarrow{e} + \frac{1}{2} \overrightarrow{f}) - 2\overrightarrow{a} \circ \overrightarrow{b} \circ \overrightarrow{c},$$

Now $\text{rank}(S) = 2$ and

$$\|I - S\| = \frac{1}{2} \|\overrightarrow{a} \circ \overrightarrow{b} \circ \overrightarrow{c} + \overrightarrow{a} \circ \overrightarrow{b} \circ \overrightarrow{c} + \overrightarrow{a} \circ \overrightarrow{b} \circ \overrightarrow{c}\| \to 0.$$

Hence, we can make $S$ arbitrarily close to $I$. Such $I$ are called degenerate. The degenerate matrices have positive Lebesgue measure (positive probability) for at least some ranks. Thus, the problem is not "rare".
The border rank is defined as the minimum number of rank-1 tensors needed to obtain arbitrarily good approximation of the tensor. Tensor $I$ in the previous example has border rank 2.

Notice that this does not contradict the uniqueness of the rank decomposition: the exact decomposition is unique, even if the approximate ones are not.
Variations of the CP

Many names of CP

- Polyadic form of a Tensor (1927)
- PARAFAC (Parallel factors) (1970)
- CANDECOMP (Canonical decomp) (1970)
- CP (CANDDECOMP/PARAFAC) (2000)

All these are the same decomposition, often invented independently as a tensor equivalent of matrix factorization

**INDSCAL**

INDSCAL (Individual Differences in Scaling) has two factor matrices and symmetric frontal slices:

\[ J \simeq [A, A, C] \]

\[ \begin{bmatrix}
    & & e \\
    & A &   \\
    I & & A^T
\end{bmatrix} \]
Usually on INDSCAL we assume that the tensor is symmetric on two modes, but this is not necessary. They do not have to have the same dimensions, though. 

$N$-way INDSCAL is

$$[A^{(1)}, A^{(2)}, ..., A^{(k-1)}, B, A^{(k+1)}, ..., A^{(n-n)}, B, A^{(n+1)}, ..., A^{(N)}]$$

which is $l_1 \times l_2 \times ... \times l_{k-1} \times l_n \times ... \times l_{n-1} \times J \times l_{n+1} \times ... \times l_N$.

INDSCAL is usually computed by first computing a CP decomposition, and "hoping" that two factor matrices converge to similar. One can also enforce similarity either at the end, or during the update:

$$A \leftarrow T_{(1)} (CA)^+$$

same if $T$ is symm.

$$A \leftarrow T_{(2)} (CA)^+$$

on 1st and 2nd mode

$$C \leftarrow T_{(3)} (AC)^+$$

A supersymmetric CP will have just one factor matrix: $I \cong \{ A, A, A \}$
Nonnegative CP

CP decomposition can be constrained to nonnegative input tensors and factor matrices to obtain a tensor version of NMF. Like with NMF, nonnegativity helps with interpretation and sometimes with sparsity of the factors.

\[
\mathbb{R}_{+}^{I \times J \times K} \cong [A, B, C], \; A \in \mathbb{R}_{+}^{l \times k}, \; B \in \mathbb{R}_{+}^{J \times k}, \; C \in \mathbb{R}_{+}^{K \times k}
\]

The projected ALS is the simplest algorithm for NCP:

\[
A \leftarrow [T_0 (COB)^+]_+, \text{ etc.}
\]

where

\[
[x]_+ = \begin{cases} 
  x & \text{if } x \geq 0 \\
  0 & \text{otherwise}
\end{cases}
\]

As with NMF, we can also develop multiplicative update rules. Starting from random \( A \in \mathbb{R}_{+}^{l \times k}, B \in \mathbb{R}_{+}^{J \times k}, \text{ and } C \in \mathbb{R}_{+}^{K \times k}, \) we update elementwise.
\[ a_{ir} \leftarrow a_{ir} \frac{\sum_{j,k} b_{jr} c_{kr} (c_{ijk}/c_{ijk})}{\sum_{j,k} b_{jr} c_{kr}} \]
\[ b_{jr} \leftarrow b_{jr} \frac{\sum_{i,k} a_{ir} c_{kr} (c_{ijk}/c_{ijk})}{\sum_{i,k} a_{ir} c_{kr}} \]
\[ c_{kr} \leftarrow c_{kr} \frac{\sum_{i,j} a_{ir} b_{jr} (c_{ijk}/c_{ijk})}{\sum_{i,j} a_{ir} b_{jr}} \]

As all involved entries are nonnegative, the update rules never violate the non-negativity constraints. They can be derived from the gradients. For \( A \), we have

\[ \frac{\partial}{\partial a_{ir}} \| I - [A, B, C] \|^2 = -\sum_{j,k} \frac{(c_{ijk}/c_{ijk}) b_{jr} c_{kr}}{\sum_{i,j} a_{ir} b_{jr}} \]

and setting

\[ a_{ir} \leftarrow a_{ir} - \eta_{ir} \frac{\partial}{\partial a_{ir}} \| I - [A, B, C] \|^2 \]

with step size \( \eta_{ir} = \frac{a_{ir}}{\sum_{j,k} b_{jr} c_{kr}} \).
Using the least-squares error has the implicit assumption of Gaussian noise. With certain types of data, this assumption is not very sensible. For example, in case of counting data with low counts, Gaussian noise is not a good model. (E.g., emails from i to j; students taking courses c.) Such data is often better explained with the Poisson distribution. The Poisson distribution is the probability of number of events occurring in a fixed interval if they occur on known average rate and independently. The Poisson distribution has a one parameter, the rate \( \lambda > 0 \). The probability mass function is \( f(k; \lambda) = \frac{\lambda^k e^{-\lambda}}{k!} \) and if \( X \sim \text{Poisson}(\lambda) \), then \( \mu = \text{E}[X] = \text{Var}[X] = \lambda \).
Notice that with larger values of $\lambda$ the distribution starts resembling the Gaussian distribution.

To model the data, we assume that the elements $t_{ijk}$ are i.i.d. Poisson distributed. We assume that the parameters come from a parameter
tensor $\Lambda$ that has a low-rank structure. That is, we assume that there exists $A, B$, and $C$ such that

$$t_{ijk} \sim \text{Poisson}(\sum_r a_{ir} b_{jr} c_{kr})$$

for all $i,j,k$. We aim to find the most likely $A, B$, and $C$ given the data tensor $I$ and the rank $R$. Hence, the quality is measured using the log-likelihood

$$\log L(t_{ijk}; A, B, C) = t_{ijk} \ln(\lambda_{ijk}) - \lambda_{ijk} - \ln(t_{ijk}!),$$

where $\lambda_{ijk} = [A, B, C]_{ijk}$ is the parameter. As $-\ln(t_{ijk}!)$ is constant, we can minimize the negative log-likelihood

$$-\log L \propto \sum_{i,j,k} \sum_{r,s} (\lambda_{ijk} - t_{ijk} \ln(\lambda_{ijk})).$$

This function is in fact the $KL$-divergence between the data and the model.

We agree that $0 \cdot \log(x) = 0$ for all $x \geq 0$. But if $\lambda_{ijk} = 0$ and $t_{ijk} > 0$, we have
modelled an observation with a zero probability and we have $t_{ijk} \log(\lambda_{ijk}) = -\infty$. To prevent this to never happen, we require that if $t_{ijk} > 0$ then $\lambda_{ijk} > 0$, too.

To solve this model, called CP-APR, we first write it open:

$$L(A) = A(C \odot \bar{B})^T - T_{(n)} \ast \log(A((\odot \bar{B})^T)).$$

We can have similar equations for the other modes, as well. Optimizing for $A$ in $L(A)$ is not trivial, though. We can use the following update rule

$$A \leftarrow A \ast \left( T_{(n)} \odot (A((\odot \bar{B})^T)) \right) ((\odot \bar{B})^T),$$

where $\odot$ is the element-wise division.

We can run this multiplicative update rule a few times before optimizing for the next factor matrix.
Applications of CP and its variations

Applications of CP-APR

CP-APR is designed to model counting data. Consider a terms–authors–journals tensor $I$, where $t_{ijk}$ is the count of how many times term $i$ was used by author $j$ in the titles of articles published in journal $k$. Considering the factor matrices of CP-APR for this data, we notice that if we have a rank-1 component $\mathbf{a}$ with high
Values, that component corresponds to some "topic" used by some authors in some journals. For example, from a 4952×6955×11 tensor of data from SIAM journals, Chi & Kolda (2012) obtained the results in the next page.

Applications of CP

Tophits

Tophits is a link-based way to rank web pages similarly to HITS. The data is \( \mathbf{T} = (t_{ijk}) \), where \( t_{ijk} \) is the number of times page \( i \) links to page \( j \) using term \( k \) in the anchor text. Computing the CP (or CP-APR) behaves similarly to HITS; each rank-1 component is one topic, \( A \) and \( B \) give the authority
Matrix C gives weights for each term in each topic.

**Fluorescence Excitation-Emission Analysis**

Fluorescence spectroscopy is a method used to analyze organic compounds. It is based on pointing a beam of UV light to the compound. The light excites electrons in molecules, and each excited electron releases a photon, which can be observed. Different materials react differently to different wavelengths, emitting photons of different wavelengths. A fluorescence landscape of a compound is a rank-1 matrix that maps the...
exciter's wavelength to the emitted photon's wavelength. Identifying the landscapes lets us identify the compounds.

\[ \begin{bmatrix} L \end{bmatrix} \]

\[ l_{ij} = \text{intensity of excitation in wavelength } j \text{ when emitted with wavelength } i. \]

Our data is a samples- by- emission wavelength- by- excitation wavelength tensor. We assume that each sample is a linear combination of the sampled compounds' landscapes. If we do a CP on this tensor to get A, B, and C, we can interpret the factors as follows:

- \( \overrightarrow{b_r} \) is the \( r \)-th landscape
- \( \overrightarrow{c_r} \) gives the abundances of compound \( r \) in different samples
Example of fluorescence analysis from Acar & Yener (2009). Bottom row shows three landscapes.

Applications of INDSCAL

INDSCAL was proposed for analysing psychological data (Carroll & Chang, 1970). In particular, they wanted to analyze how similar or dissimilar people find things, and what are the underlying factors for these similarities. Their hypothesis is that there are a small number of (latent) reasons, but that the different reasons have different effects.
on different people. Hence they collected data where 20 people rated the similarity of a number of countries. Factorizing this with INDSCAL gave the latent similarity matrices \( \mathbf{a}_i \) and a weight matrix \( \mathbf{c} \).

![Graph showing developed vs communist and west vs communist](image)

Doves and hawks and moderates (D, H, and M) in \( \mathbf{a}_i \mathbf{a}_i^T \) scatter plot.

![Graph showing political alignment and economic development](image)
The Tucker Decompositions

Let \( \mathbf{T} \in \mathbb{R}^{I \times J \times K} \). Its Tucker-3 — or just Tucker — decomposition has a core tensor \( \mathbf{G} \in \mathbb{R}^{P \times Q \times R} \) and three factor matrices \( \mathbf{A} \in \mathbb{R}^{I \times P} \), \( \mathbf{B} \in \mathbb{R}^{J \times Q} \), and \( \mathbf{C} \in \mathbb{R}^{K \times R} \), and it is defined as

\[
\mathbf{T} \approx \mathbf{G} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}
\]

\[
= \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} \mathbf{a}_p \mathbf{b}_q \mathbf{c}_r
\]

\[
= [G; A, B, C]
\]

Hence, elementwise Tucker is

\[
t_{ijk} \approx \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_{ip} b_{jq} c_{kr}.
\]
Tucker decomposition is usually applied only to 3-way tensors, but $N$-way version is straightforward to define:

\[
\mathbf{I} \approx \mathbf{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_N \mathbf{A}^{(N)}
\]

\[
\ell_{i_1 i_2 \ldots i_N} \approx \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_N=1}^{R_N} g_{r_1 r_2 \cdots r_N} a_{i_1 r_1}^{(1)} a_{i_2 r_2}^{(2)} \cdots a_{i_N r_N}^{(N)}
\]

The core tensor $\mathbf{G}$ can be considered as a compressed version of the original tensor $\mathbf{I}$ if $P \leq 1$, $Q \leq J$, and $R \leq K$. If $P = Q = R$ and $\mathbf{G}$ is (hyper-) diagonal, then Tucker3 reduces to the CP decomposition. In particular, with hyper-diagonal $\mathbf{G}$ where all diagonal entries are 1, we have that $[\mathbf{G}]_{j \Lambda \beta \gamma} = [\mathbf{A}, \beta, \gamma]$. 
The Tucker decomposition can be expressed in a matricized form:

\[
T_{(1)} \approx A G_{(1)} (C \otimes B)^T \\
T_{(2)} \approx B G_{(2)} (C \otimes A)^T \\
T_{(3)} \approx C G_{(3)} (B \otimes A)^T.
\]

To gain some intuition, let's consider the first frontal slice of \( \Pi_{G; A, B, C} \):

\[
\Pi_{G; A, B, C} = A D^{(1)} B^T,
\]

with \( D^{(1)} = \sum_{r=1}^{R} c_{1r} G_r \).

Now we have

\[
T_{(1)} = A \left[ D^{(1)} B^T \quad D^{(2)} B^T \quad \ldots \quad D^{(k)} B^T \right] \\
= A \left[ (\Sigma_r c_{1r} G_r) B^T \quad (\Sigma_r c_{2r} G_r) B^T \quad \ldots \quad (\Sigma_r c_{kr} G_r) B^T \right] \\
= A \left[ G_1 \quad G_2 \quad \ldots \quad G_k \right] \begin{pmatrix}
  (c_{11} B^T & c_{21} B^T & \ldots & c_{k1} B^T) \\
  (c_{12} B^T & c_{22} B^T & \ldots & c_{k2} B^T) \\
  \vdots & \vdots & \ddots & \vdots \\
  (c_{1R} B^T & c_{2R} B^T & \ldots & c_{kR} B^T)
\end{pmatrix} \\
= A G_{(1)} (C \otimes B)^T.
\]
In \( N \)-way case, the mode-\( n \) matricized version becomes

\[
T_{(n)} \approx A^{(n)} G_{(n)} (A^{(n)} \otimes \cdots \otimes A^{(n+1)} \otimes A^{(n-1)} \otimes \cdots \otimes A^{(1)})^T
\]

**Tucker1 and Tucker2**

The Tucker2 decomposition leaves one factor matrix as an identity matrix, e.g.

\[
J \approx G \times_1 A \times_2 B = [G; A, B, 1]
\]

where \( J \in \mathbb{R}^{(x)\times(c)} \) and \( G \in \mathbb{R}^{P \times q \times h} \).

The Tucker1 decomposition leaves two factor matrices as identity, for instance

\[
I \approx G \times_1 A = [G; A, 1, 1]
\]

This is equivalent to standard least-squares matrix factorization.
The \textit{n-rank}

Let $T \in \mathbb{R}^{1 \times 1 \times \cdots \times n}$. The \textit{n-rank} of $T$, $\text{rank}_n(T)$, is the column rank of $T_{(n)}$, i.e., the number of linearly independent columns in $T_{(n)}$. If we set $R_n = \text{rank}_n(T)$, then $T$ is rank-$\{(R_1, R_2, \ldots, R_n)\}$, though note that this definition is not compatible with the usual tensor rank.

Clearly, $R_n \leq 1_n$ for all $n \in \mathbb{N}$. Finding a Tucker decomposition of size $(R_1, R_2, \ldots, R_n)$ is easy if $\text{rank}_n(T) = R_n$ for all $n$.

\textbf{Computing Tucker}

From now on, we enforce that the columns of the factor matrices are mutually orthogonal, that is,

\[ A^TA = 1_p, \quad B^TB = 1_q, \quad \text{and} \quad C^TC = 1_r. \]
**Higher-order SVD (HOSVD)**

HOSVD is a simple method to calculate the Tucker decomposition using SVD:

\[
\text{for } n = 1, \ldots, N \text{ do } \\
(u, \Sigma, v) \leftarrow \text{SVD}(T^{(n)}) \\
A^{(n)} \leftarrow U(:, 1: R_n) \\
\text{end } \\
G \leftarrow I x_1 A^{(1)\top} x_2 A^{(2)\top} x_3 \ldots x_N A^{(N)\top}
\]

In the last row above, we use the fact that the factor matrices are column-orthogonal:

\[
I = G x_1 A \iff T^{(1)} = A G^{(1)} \iff A^{-\top} T^{(1)} = G^{(1)} \\
\ast \iff A^{\top} T^{(1)} = G^{(1)} \iff I x_1 A^{\top} = G^{(1)},
\]

where \( \ast \) is due to the column orthogonality.

HOSVD is not very good for finding a low-error decomposition, as it doesn't take into account the higher-order structure in the tensor. It is, however, useful for finding an initial solution for other algorithms.
Higher-order Orthogonal Iteration (HOI)

We know that
\[
G = I x_1 A^T x_2 B^T x_3 C \quad \text{(x)}
\]
is optimal for column-orthogonal \(A, B,\) and \(C.\)

Let us re-write the objective:
\[
\|T - [G; A, B, C]\|^2 = \|T\|^2 - 2 \langle T, [G; A, B, C]\rangle \\
+ \| [G; A, B, C]\|^2
\]
\[
\begin{align*}
&= \|T\|^2 - 2 \langle T, [G; A, B, C]\rangle + \|G\|^2 \\
&= \|T\|^2 - 2 \langle T, [G; A, B, C]\rangle + \|G\|^2 \\
&= \|T\|^2 - 2 \langle G, G\rangle + \|G\|^2 \\
&= \|T\|^2 - \|G\|^2 \\
&= \|T\|^2 - \|T x_1 A^T x_2 B^T x_3 C^T\|^2,
\end{align*}
\]

* We use the column orthogonality

** We multiply the factors out from \([G; A, B, C].\)

*** We use (x)

**** We use \(\langle G, G\rangle = \|G\|^2\)

As \(\|T\|\) is constant, we learn that
\[
\|T - [G; A, B, C]\|^2 \approx \|T x_1 A^T x_2 B^T x_3 C^T\|^2
\]
Hence, we want to maximize

$$\max_{A, B, C} \| I x_1 A^T x_2 B^T x_3 C^T \|,$$

or equivalently

$$\max_{A, B, C} \| A^T T_{(1)} (C \otimes B) \|,$$
$$\max_{A, B, C} \| B^T T_{(2)} (C \otimes A) \|,$$
$$\max_{A, B, C} \| C^T T_{(3)} (B \otimes A) \|. $$

Essentially, $G$ is fully determined by the factor matrices, and we do not have to take it into account.

To compute $H_001$, we again use SVD:

```
Initialize $A^{(n)}, A^{(a)}$ using HOSVD
repeat
  for $n = 1, ..., N$ do
    $Y = I x_1 A^{(n)T} x_2 ... x_N A^{(a)T}$
    $U, \Sigma, V = \text{svd}(Y_{(n)})$
    $A^{(m)} = U(:, 1:R_n)$
  end
until convergence
$G = I x_1 A^{(t)T} x_2 ... x_N A^{(a)T}$
```
Naive ALS for Tucker

We can also use the standard ALS approach to solve Tucker decomposition. This does not, generally, yield to orthogonal factor matrices, though. In some cases, this is what we want. To update the factor matrices, we have:

\[
A \leftarrow T_{(1)}(G_{(1)}(C \otimes B)^T)^+ \\
B \leftarrow T_{(2)}(G_{(2)}(C \otimes A)^T)^+ \\
C \leftarrow T_{(3)}(G_{(3)}(B \otimes A)^T)^+
\]

To update the core, we can use a vectorized format of Tucker and solve

\[
G = \arg \min_G \| \text{vec}(T) - (C \otimes B \otimes A) \text{vec}(G) \|,
\]

which is just a standard least-squares problem.

The ALS algorithm is rarely used due to the large pseudo-inverse.
It can be useful for computing the Tucker 2 decomposition, though. For Tucker 2, we replace $C$ with the identity matrix, and solve:

$$A \leftarrow T_r(G_{(r)} (1 \otimes B)^T)^+$$
$$B \leftarrow T_r(G_{(r)} (1 \otimes A)^T)^+$$

where we can use the fact that

$$(1 \otimes B)^T = \begin{pmatrix}
B^T \\
& B^T \\
& & \ddots \\
& & & B^T
\end{pmatrix}$$

and that

$$G_{(r)} (1 \otimes B)^T = [G_1, G_2, \ldots, G_k] (1 \otimes B)^T = \begin{pmatrix}
G_1 B^T \\
G_2 B^T \\
& \ddots \\
& & G_k B^T
\end{pmatrix}$$

The core can be updated separately for each frontal slice:

$$G_k \leftarrow B^T T_k A^+.$$
**Nonnegative Tucker3**

Similarly to NCP, we can consider the nonnegative variant of the Tucker3 decomposition. We use the unfolded versions of Tucker

\[
T_{(1)} \approx A G_{(1)} (C \otimes B)^T \\
T_{(2)} \approx B G_{(2)} (C \otimes A)^T \\
T_{(3)} \approx C G_{(3)} (B \otimes A)^T
\]

and update the factors using multiplicative rules similar to those in NCP.

To initialize \(A, B, C, \) and \(G\) we can run HOSVD on \(I\) and truncate the negative values in the result to zero. We can also use random initialization, but this can yield very bad initial solutions.
The nonnegative Tucker3 algorithm is

\[ \text{Input: } I \in \mathbb{R}_{\geq 0}^{I} \times \mathbb{R}_{\geq 0}^{\times k} \times \cdots \times \mathbb{R}_{\geq 0}^{\times t}, (R_1, R_2, \ldots, R_N) \]

\[ (G, A^{(1)}, \ldots, A^{(N)}) \leftarrow \text{HOSVD}(I, R_1, R_2, \ldots, R_N) \]

\[ G \leftarrow [G]_+, A^{(n)} \leftarrow [A^{(n)}]_+ \text{ for all } n \in [N] \]

**repeat**

\[ G \leftarrow G \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)} \]

for \( n = 1 \ldots N \) do

\[ A^{(n)} \leftarrow A^{(n)} \times_1 \left( T_{(n)} A^{(n)} \times_n G_{(n)} \right) \otimes (Q_{(n)} A^{(n)} \times_n G_{(n)}) \]

\[ \overline{a}^{(n)} \leftarrow \overline{a}^{(n)} / \| \overline{a}^{(n)} \| \]

**end**

\[ G \leftarrow G \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)} \]

**until** convergence

Here, \( A^{(n)} \times_n = A^{(n)} \times_1 \cdots \times_{n-1} A^{(n+1)} \times_n A^{(n-1)} \times_{n+1} \cdots \times N A^{(N)} \).
Applications of Tucker

Tensor Faces

We can use Tucker decomposition to separate the effects of different illuminations, expressions, poses, etc from pictures, provided that we have training data with all variations for all subjects.

In Tensor Faces, we have photos of subjects in different views (front, left, right,...), under different illuminations and having different expressions. This gives a people-by-views-by-illuminations-by-expressions-by-pixels tensor. See the next page for some examples (image from Vasilescu & Terzo-poulos: Multilinear analysis of image ensembles: TensorFaces, ECCV '02).
Applying the Tucker decomposition with no reduction in the dimensions yields

\[ G \times_1 A_{\text{people}} \times_2 A_{\text{views}} \times_3 A_{\text{illums}} \times_4 A_{\text{express}} \times_5 A_{\text{pixels}} \]

The \( A_{(n)} \) factor matrices encode the variations in these modes, while the core tensor governs the interactions.

This model generalizes Eigenfaces: we can write

\[ T(\text{pixels}) = A_{\text{pixels}} G(\text{pixels}) (A_{\text{express}} \otimes A_{\text{illums}} \otimes A_{\text{views}} \otimes A_{\text{people}})^T \]

to get a standard Eigenfaces setting. Multiplying \( G \times_5 A_{\text{pixels}} \) gives us a tensor that shows the primary variations along the modes. Some examples are in the next page (Vasilescu & Terzopoulos, 2002).

On the other hand, if we multiply

\[ G \times_2 A_{\text{views}} \times_5 A_{\text{pixels}} \]
We get the variations with the different viewpoints, as depicted in the picture in the previous page (Vasilceanu & Terzopoulos, 2002).

Finding facts in open IR

In open information retrieval, our goal is to extract structured knowledge from unstructured data using unsupervised methods. Typically, we want to extract subject-predicate-object (s, p, o) triples with disambiguated entities and relations. To do that, we can first run standard natural language parsers to obtain noun phrase-verbal phrase-noun phrase (np, vp, np) triples, e.g. Donald J. Trump, is, POTUS
Donald Trump is the president of, USA
The Donald, is the prez of, 'Murica
Donald Trump, is the son of, POTUS
Those triples encode two (s, p, o) triples

- donald_j_trump, isPresidentOf, USA
- donald_j_trump_jr, isSonOf, donald_j_trump

To extract these, we need to handle synonyms (President vs. prez) and homonyms (Donald Trump Jr.), among others. We can model this with mappings from noun phrases to entities and from verbal phrases to relations. Considering subjects and objects separately, we have A: np→s, B: np→o, and C: vp→p. We model these as matrices. If they are column orthogonal, and if our original (np, vp, np) triples are stored in a tensor $T$, we can get the true (s, p, o) triples as

$$T^T A^T x_1 B^T x_2 C^T.$$
That is, if we do Tucker decomposition to the tensor containing the surface triples, we find the latent entities and relations (though they might not contain any "real" entities or relations).

A practical problem with this approach is that the core tensor has to be very big, making it hard to work with. The core should also be sparse, which is not the case if we obtain it with the multiplication.
Other Decompositions

**RESCAL**

RESCAL decomposition is a combination of Tucker2 and INDSCAL, that is, it is a Tucker2 decomposition with symmetric frontal slices.

Given $I \in \mathbb{R}^{1 \times 1 \times K}$ and $RE \in N$, RESCAL is

$$I \cong \left[ G, A, A, I \right] \implies TR \approx AG \cdot A^T,$$

where $A \in \mathbb{R}^{1 \times R}$ and $G \in \mathbb{R}^{R \times R \times K}$.

![Diagram](image)

We do not necessarily require that the frontal slices of $I$ are symmetric.
To compute RESCAL, consider the mode-1 matrixization:

\[ T_{(1)} = A G_{(1)} (1 \otimes A)^T. \]

Like INDSCAL, we have \( A \) twice. One solution here is to "fix" the right-hand \( A \) and only update the left-hand \( A \).

If \( T_k \) are not symmetric, this alone can yield sub-optimal results. To make the updated \( A \) to work also with \( T_{(2)} \), we replace \( T_{(1)} \) with \([T_1 \ T_1^T \ T_2 \ T_2^T \cdots T_k \ T_k^T]\).

This gives

\[ \| Y - AH(l_{2k} \otimes A)^T \|_F \]

with \( Y = [T_1 \ T_1^T \ T_2 \ T_2^T \cdots T_k \ T_k^T] \) and \( H = [G_1 \ G_1^T \ G_2 \ G_2^T \cdots G_k \ G_k^T] \), with update rule

\[
A = Y (H (l_{2k} \otimes A)^T)^T
= \left( \sum_{k=1}^K (T_K G_K^T + T_K^T A G_R) \right) \left( \sum_{k=1}^K (B_K + C_K) \right)^+, \\
\text{with } B_K = G_K^T A^T A G_K \text{ and } C_K = G_K^T A^T A G_K.
\]
To update the core, we update each frontal slice thereof:

\[ G_{t_k} = \arg \min_{G_{t_k}} \| \text{vec}(T_k) - (A \otimes A) \text{vec}(G_{t_k}) \| \]

giving \( \text{vec}(G_{t_k}) = (A \otimes A)^+ \text{vec}(T_k) \). This still requires computing a pseudo-inverse of a big matrix \((A \otimes A)\). We can reduce the computation with a skinny QR decomposition of \( A \). Let \( A = QU \), where \( Q \in \mathbb{R}^{\times R} \) is column-orthogonal and \( U \in \mathbb{R}^{R \times R} \) is upper-triangular. It should be noted that every matrix has a QR decomposition. We now have:

\[
\| T_k - A G_{rA} U^T \| = \| T_k - QU G_{rR} U^T Q^T \|
\]

\[ = \| Q^T T_k Q - U G_{rR} U^T \| \]

which is optimized by

\[ \text{vec}(G_{t_k}) = (U \otimes U)^+ \text{vec}(Q^T T_k Q) \].
RESCAL can be used when we want to model non-symmetric data with DEDICOM type decomposition. Using the $A A^T$ model provides an "information flow" from mode 1 to mode 2 and vice versa. For example, if we have $(s, p, o)$ data, we can assume that subjects and objects come from the same set of entities, and there should be just one factor matrix for them. But $A A^T$ is symmetric, while most predicates are not (isPresidentOf, isSonOf, ...). If we have resolved the predicates in a surface $(np, vp, np)$ tensor (so that we have a $(np, p, np)$ tensor), we can decompose it with RESCAL to obtain $(s, p, o)$ core $G$ and $A: np\rightarrow svO$. 
DEDICOM decomposition is a matrix decomposition that is equivalent to a frontal slice of RESCAL:

\[ T = A G A T. \]

Like RESCAL, DEDICOM can be used to model asymmetric relations between entities.

The three-way DEDICOM adds weights for each entity factors participation in each position in the third mode. For example, if we have countries-by-countries-by-time tensor $T \in \mathbb{R}^{I \times J \times K}$, where $T_{ijk}$ has the value of trade from country $i$ to country $j$.
At time point $k$, 3-way DEDICOM adds information on how much does a country factor $\tilde{a}_i^k$ act as a seller or buyer at time $k$.

Each frontal slice of a 3-way DEDICOM is

$$T_k = A D_k G D_k A^T,$$

where $A$ and $G$ are as in the matrix case, and $D$ is an $R \times R \times K$ tensor such that each frontal slice $D_k$ is diagonal.

$$(D_k)_{rr} = d_{rrk}$$

is the weight of a factor $r$ at time $k$.

$$J \sim A \begin{bmatrix} D & \text{ } & G \text{ } & \text{ } & D \text{ } & \text{ } & A^T \end{bmatrix}$$

Where RESCAL decomposes each relation separately using $G_k$, DEDICOM assumes there is only one (potentially asym-
metric) relation encoded in G, but that the participation to that relation varies over time.

**Computing DEDICOM: ASALSAN**

Given $T \in \mathbb{R}^{1 \times 1 \times T}$ and $R \in \mathbb{R}^{1 \times R}$, in DEDICOM we want to find $A \in \mathbb{R}^{1 \times R}$, $D \in \mathbb{R}^{R \times R}$, and $G \in \mathbb{R}^{R \times R}$ that minimize

$$
\sum_{k=1}^{K} \| T_k - A D_k G D_k A^T \|_1.
$$

**ASALSAN** (alternating simultaneous approximation, least squares, and Newton) uses the methods in the name for optimizing different factors.

As with RESCAL, DEDICOM has matrix $A$ on both sides, and ASALSAN uses
similar approach handle it; it stacks pairs $T_k^T T_k$ to get

$$ Y = [T_1^T T_1 \cdots T_k^T T_k] $$

The error function becomes

$$ \| Y - A H (l_{2k} \otimes A^T) \| $$

with

$$ H = [ D_1 G D_1^T \ D_1 G^T D_1 \ \cdots \ D_k G D_k \ D_k G^T D_k ] $$

Similarly to RESCAL, we fix the right $A$ and update

$$ A \leftarrow \left( \sum_{k=1}^{K} (T_k A D_k G^T D_k + T_k^T A D_k G D_k) \right) \left( \sum_{k=1}^{K} (B_k + C_k) \right)^{-1} $$

where

$$ B_k = D_k G D_k^T (A^T A) D_k G^T D_k $$

$$ C_k = D_k G D_k^T (A^T A) D_k^T G D_k $$

Matrix $G$ we can update using the vectorized representation
\[ G = \arg \min_{G} \left\| \begin{pmatrix} \text{vec}(T_1) \\ \text{vec}(T_2) \\ \vdots \\ \text{vec}(T_K) \end{pmatrix} - \begin{pmatrix} \mathbf{A}_1 \otimes \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_K \otimes \mathbf{A}_K \end{pmatrix} \text{vec}(G) \right\| \]

Finally, to update \( D \), we can update each frontal slice separately, having only \( R \) unknowns. As there doesn’t seem to be any easy closed-form solution, ASALSAN uses the Newton's method.

**PARATUCK2**

PARATUCK2 (portmanteau of PARAFAC and Tucker 2) generalizes DEDICOM to allow different factors on the right-hand side:

\[ T_k = \mathbf{A}_k \mathbf{D}_k \mathbf{G} \mathbf{D}_k^T \]

where \( \mathbf{A}, \mathbf{D}, \) and \( \mathbf{G} \) are as in DEDICOM.
Tensor train (TT) decomposition

Given a tensor $T \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_N}$, the Tensor train (or TT) decomposition has $N$ 3-way factor tensors $G^{(n)} \in \mathbb{R}^{R_{n-1} \times R_n \times n_n}$, where $R_0 = R_N = 1$, and the rest are model parameters ("TT rank") $R_1, R_2, \ldots, R_{N-1}$. TT expresses an element of $T$ as a product of the frontal slices of $G^{(n)}$ is:

$$t_{i_1 i_2 \cdots i_N} = G^{(1)}_{i_1 i_0} G^{(2)}_{i_2 i_0} \cdots G^{(N)}_{i_N i_0}$$

Notice that this is a scalar as $G^{(1)}_{i_1} \in \mathbb{R}^{1 \times R_1}$ and $G^{(N)}_{i_N} \in \mathbb{R}^{R_{N-1} \times 1}$. Writing the products open, we have

$$t_{i_1 i_2 \cdots i_N} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_{N-1}=1}^{R_{N-1}} \left( G^{(1)}(r_1, i_1) G^{(2)}(r_1, r_2, i_2) \cdots \right. \times \left. G^{(N-1)}(r_{N-2}, r_{N-1}, i_{N-1}) G^{(N)}(r_{N-1}, i_N) \right)$$

For a 3-way tensor, this simplifies to

$$t_{ijk} = \sum_{p=1}^{P} \sum_{q=1}^{Q} A(p, i) G(p, q, j) B(q, k)$$
where \( A \in \mathbb{R}^{p \times l} \), \( G \in \mathbb{R}^{p \times q \times l} \), and \( B \in \mathbb{R}^{q \times k} \).

Let's consider an example: Let \( T \in \mathbb{R}^{3 \times 4 \times 5} \) be such that \( t_{ijk} = i + j + k \) for all \( i, j, \) and \( k \). Define \( A, G, \) and \( B \) so that

\[
A(i,:) = (i, 1) \quad G_j = \begin{pmatrix} 1 & 0 \\ j & 1 \end{pmatrix} \quad B(:,k) = \begin{pmatrix} 1 \\ k \end{pmatrix}.
\]

Now

\[
t_{ijk} = (i, 1) \begin{pmatrix} 1 \\ j \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ k \end{pmatrix} = (i + j + k) = i + j + k.
\]

Tensor \( T \) has 60 elements, while the tensor train has only \( 3 \cdot 2 + 4 \cdot 2 + 5 \cdot 2 = 32 \).

In general, \( TT \) can express an \( l \times l \times \cdots \times l \) tensor with \( l^N \) elements with \( l \cdot N \cdot R^2 \) elements, where \( R = \max_{n=1}^{N} R_n \).
This number $R = \max_{n=1}^{N} R_n$ is called the (maximal) TT-rank.

TT allows certain operations to be effective for tensors stored in the TT-format. If $T$ and $S$ are such that
\[ t_{i_1...i_n} = G_{i_1}^{(1)} G_{i_2}^{(2)} ... G_{i_n}^{(n)} \text{ and } s_{i_1...i_n} = H_{i_1}^{(1)} ... H_{i_n}^{(n)}, \]
then $U = T + S$ has TT decomposition to
\[
F_{i_1...i_n}^{(n)} = \begin{pmatrix} G_{i_1}^{(n)} & H_{i_1}^{(n)} \end{pmatrix} \quad (n=2, ..., N-1)
\]
\[
F_{i_1}^{(1)} = \begin{pmatrix} G_{i_1}^{(1)} & H_{i_1}^{(1)} \end{pmatrix}
\]
\[
F_{i_1...i_n}^{(N)} = \begin{pmatrix} G_{i_1}^{(N)} \\ H_{i_1}^{(N)} \end{pmatrix}
\]

If the TT-ranks of $T$ are $R_I^1, R_I^2, ..., \text{ and of } S$ are $R_S^1, R_S^2, ..., \text{ then for } U \text{ they are } R_{I+1}^1, R_{I+2}^2, ..., \text{ and } R_{S+1}^1, R_{S+2}^2, ...$.

Similarly, for the Hadamard (element-wise) product we have that
\[ U = I \otimes S \iff F^{(n)}_{i_n} = G^{(n)}_{i_n} \otimes H^{(n)}_{i_n}, \ n = 1, \ldots, N \]

To see this, notice that
\[ u_{i_1 i_2 \ldots i_N} = (G^{(1)}_{i_1} \ldots G^{(n)}_{i_n}) (H^{(1)}_{i_1} \ldots H^{(n)}_{i_n}) \]
\[ = (G^{(1)}_{i_1} \ldots G^{(n)}_{i_n}) \otimes (H^{(1)}_{i_1} \ldots H^{(n)}_{i_n}) \]
\[ = (G^{(1)}_{i_1} \otimes H^{(1)}_{i_1}) (G^{(2)}_{i_2} \otimes H^{(2)}_{i_2}) \ldots (G^{(n)}_{i_n} \otimes H^{(n)}_{i_n}). \]

Hence, \( R^U_n = R^I_n R^S_n \).

**Computing the TT**

The basic algorithm for computing the TT decomposition within any accuracy \( \varepsilon \) is the TT-SVD. Given \( I \), it finds \( S \) that has the smallest possible ranks \( R^S_n \) such that
\[ \| I - S \| \leq \varepsilon \| I \|. \]

The algorithm is based on computing SVD to obtain unfolded \( G \)'s.
\[
TT-\text{SVD} (T, \varepsilon)
\]
\[
\delta \ll \varepsilon(N-1)^{-\frac{1}{2}} \quad \text{||} \quad T \text{||}
\]
\[
S \ll T; \quad J \ll 1; 2; \ldots; n; \quad R_0 = 1
\]
for \(n=1, \ldots, N\) do
\[
S \ll \text{reshape}(S, R_{n-1} \times l_n, J/(R_{n-1} \times l_n))
\]
\[
(U, \Sigma, V) \ll \text{SVD}(S)
\]
\[
R_n \ll \arg \min_r \{ \sum_{i=1}^{J/(R_{n-1} \times l_n)} \sigma_i^2 \leq \delta^2 \}
\]
\[
G^{(n)} \ll \text{reshape}(U(J, 1: R_n), R_{n-1}, l_n, R_n)
\]
\[
S \ll \Sigma V^T
\]
\[
J \ll JR_n/(l_n R_{n-1})
\]
end
\[
G^{(n)} \ll S
\]
return \((G^{(1)}, G^{(2)}, \ldots, G^{(n)})\)

N.B. Here \(G^{(n)} \in R^{R_{n-1} \times l_n \times R_n}\), not \(R^{R_{n-1} \times R_n \times l_n}\).

Applications of TT

TT is not (often?) used for data analysis as such, as it lacks easy
interpretation of the "cars" $G^{(1)}$, $G^{(2)}$, ...

It is, however, commonly used to reduce the number of free parameters in different machine learning applications. The parameters can also be stored in a matrix: we can "fold" it into a tensor and then apply TT.

TT can also be applied to the core tensor of the Tucker decomposition, if we want to compress it more.
**CORCONDIA**

CORCONDIA (core consistency diagnostic) is not a tensor factorization per se. Rather, it's a method for selecting the rank in a CP decomposition, or, alternatively, for deciding between CP and Tucker.3

The idea of CORCONDIA is to use the fact that CP is a special case of Tucker. Given I, we can first compute CP \([A, B, C]\) and then use these as the factor matrices for Tucker, and find the core tensor \(G\) as

\[
G \leftarrow (A \otimes B \otimes C)^+ \text{vec}(I)
\]

Now CORCONDIA statistic measures how diagonal \(G\) is

\[
\text{CORCONDIA} = 100 \left(1 - \frac{\sum_{i,j,k} (g_{ij} - \mathbb{1}(i = j = k))^2}{R}\right)
\]
Here

\[ I(i = j = k) = \begin{cases} 1 & \text{if } i = j = k \\ 0 & \text{otherwise} \end{cases} \]

The CORCONDIA statistic gives us an indication how good a model rank-R CP is for the data. The statistic assumes values from \((-\infty, 100]\), and small values indicate a bad match. Usually, values less than 50 indicate some problems.

That's all folks!