Other Decompositions

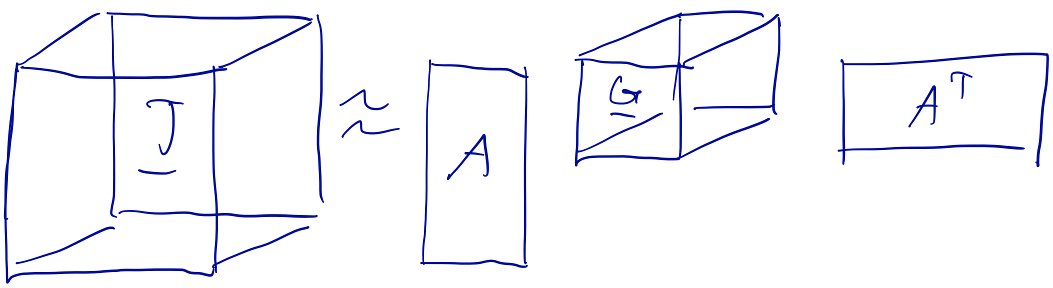
RESCAL

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

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

$$\underline{I} \approx [\![\underline{G} ; A, A, I]\!] \iff I_k \approx A G_k A^T,$$

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



We do not necessarily require that the frontal slices of $\underline{I}$ are symmetric.

To compute RESCAL, consider the mode-1 matricization:

$$T_{(1)} = A\, G_{(1)} (I \otimes A)^T.$$

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

$$\| Y - AH(I_{2k} \otimes A)^T \|_F$$

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

$$A = Y\left(H(I_{2k} \otimes A)^T\right)^+$$
$$= \left(\sum_{k=1}^{K} (T_k G_k^T + T_k^T A G_k)\right)\left(\sum_{k=1}^{K} (B_k + C_k)\right)^+$$

with $B_k = G_k A^T A G_k^T$ and $C_k = G_k^T A^T A G_k$

To update the core, we update each frontal slice thereof

$$G_k = \arg\min_{G_k} \|vec(T_k) - (A \otimes A) vec(G_k)\|$$

giving $vec(G_k) = (A \otimes A)^{+} vec(T_k)$. This still requires computing a pseudo-inverse of a big matrix $(A \otimes A)$. We can reduce the computation with a <u>skinny QR</u> decomposition of A. Let $A = QU$, where $Q \in \mathbb{R}^{I \times R}$ 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

$$\begin{aligned} \|T_k - A G_k A^T\| &= \|T_k - Q U G_k U^T Q^T\| \\ &= \|Q^T T_k Q - U G_k U^T\| \end{aligned}$$

which is optimized by

$$vec(G_k) = (U \otimes U)^{+} vec(Q^T T_k Q).$$

RESCAL can be used when we want to model non-symmetric data with DEDICOM type decomposition. Using the AG_kA^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 AA^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 $\underline{G}$ and $A: np \Rightarrow s \cup o$.

DEDICOM

DEDICOM decomposition is a matrix decomposition that is equivalent to a frontal slice of RESCAL:

$$T = AGA^T.$$

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

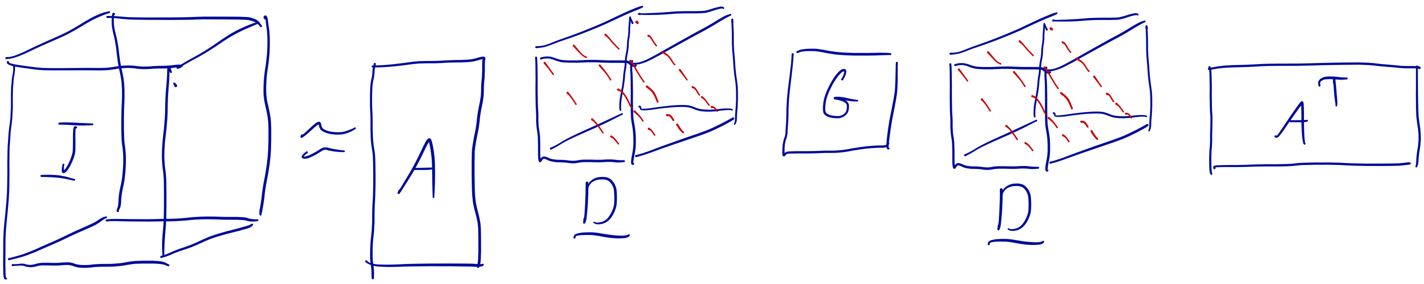
The three-way DEDICOM adds weights for each entity factors participation in each position in the third mode. For example, if we have countries-by-countries-by-time tensor $\mathcal{T} \in \mathbb{R}^{I \times I \times K}$, where t_{ijk} has the value of trade from country i to country j

at time point k, 3-way DEDICOM adds information on how much does a country factor $\vec{a}_i$ act as a seller or buyer at time k.

Each frontal slice of a 3-way DEDICOM is

$$T_k \simeq A D_k G D_k A^T,$$

where A and G are as in the matrix case, and $\underline{D}$ 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.



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

Computing DEDICOM: ASALSAN

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

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

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 $T_k T_k^T$ to get

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

The error function becomes

$$\| Y - AH(I_{2K} \otimes A^T) \|$$

with

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

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

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

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

Matrix G we can update using the vectorized representation

$$G \leftarrow \arg\min_{G} \left\| \begin{pmatrix} \mathrm{vec}(T_1) \\ \mathrm{vec}(T_2) \\ \vdots \\ \mathrm{vec}(T_k) \end{pmatrix} - \begin{pmatrix} AD_1 \otimes AD_1 \\ \vdots \\ AD_k \otimes AD_k \end{pmatrix} \mathrm{vec}(G) \right\|$$

Finally, to update $\underline{D}$, 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:

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

where A, $\underline{D}$, and G are as in DEDICOM.

Tensor train (TT) decomposition

Given a tensor $\underline{T} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, the Tensor train (or TT) decomposition has N 3-way factor tensors $\underline{G}^{(n)} \in \mathbb{R}^{R_{n-1} \times R_n \times I_n}$, 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 $\underline{T}$ as a product of the frontal slices of $\underline{G}^{(n)}$'s:

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

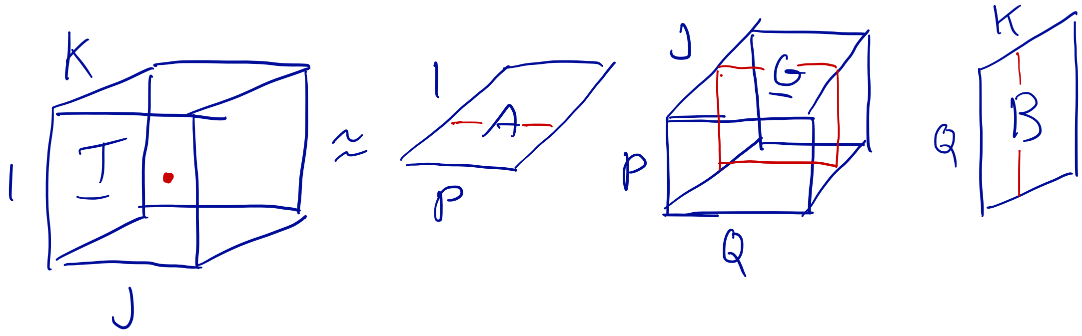
Notice that this is a scalar as $G^{(1)}_{i_1} \in \mathbb{R}^{1 \times R_1}$ and $G^{(N)}_{i_N} \in \mathbb{R}^{R_{N-1} \times 1}$. Writing the products open, we have

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

For a 3-way tensor, this simplifies to

$$t_{ijk} = \sum_{p=1}^{P} \sum_{q=1}^{Q} A(p, i)\, \underline{G}(p, q, j)\, B(q, k)$$

where $A \in \mathbb{R}^{P \times I}$, $\underline{G} \in \mathbb{R}^{P \times Q \times J}$, and $B \in \mathbb{R}^{Q \times K}$



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

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

Now

$$t_{ijk} = \begin{pmatrix} i & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ j & 1 \end{pmatrix} \begin{pmatrix} 1 \\ k \end{pmatrix} = \begin{pmatrix} i+j & 1 \end{pmatrix} \begin{pmatrix} 1 \\ k \end{pmatrix} = i + j + k.$$

Tensor $\underline{T}$ has 60 elements, while the tensor train has only $3 \cdot 2 + 4 \cdot 2 \cdot 2 + 5 \cdot 2 = 32$. In general, TT can express an $\underbrace{I \times I \times \cdots \times I}_{N}$ tensor with I^N elements with

$$I \cdot N \cdot R^2$$

elements, where $R = \max_{n=1}^{N} R_n$.

This number $R = \max\limits_{n=1}^{N} R_N$ is called the (maximal) TT-rank.

TT allows certain operations to be effective for tensors stored in the TT-format. If $\underline{T}$ and $\underline{S}$ are such that

$$t_{i_1 \cdots i_N} = G^{(1)}_{i_1} G^{(2)}_{i_2} \cdots G^{(N)}_{i_N} \quad \text{and} \quad s_{i_1 \cdots i_N} = H^{(1)}_{i_1} \cdots H^{(N)}_{i_N},$$

then $\underline{U} = \underline{T} + \underline{S}$ has TT decomposition to

$$F^{(n)}_{i_n} = \begin{pmatrix} G^{(n)}_{i_n} & \\ & H^{(n)}_{i_n} \end{pmatrix} \quad (n = 2, \dots, N-1)$$

$$F^{(1)}_{i_1} = \begin{pmatrix} G^{(1)}_{i_1} & H^{(1)}_{i_1} \end{pmatrix}$$

$$F^{(N)}_{i_N} = \begin{pmatrix} G^{(N)}_{i_N} \\ H^{(N)}_{i_N} \end{pmatrix}$$

If the TT-ranks of $\underline{T}$ are $R^T_1, R^T_2, \dots,$ and of $\underline{S}$ are $R^S_1, R^S_2, \dots,$ then for $\underline{U}$ they are $R^T_1 + R^S_1, R^T_2 + R^S_2, \dots$.

Similarly for the Hadamart (element-wise) product we have that

$$\underline{U} = \underline{T} * \underline{S} \iff F^{(n)}_{i_n} = G^{(n)}_{i_n} \otimes H^{(n)}_{i_n}, \quad n = 1, \dots, N$$

To see this, notice that

$$\begin{aligned} u_{i_1 i_2 \cdots i_N} &= \left(G^{(1)}_{i_1} \cdots G^{(N)}_{i_N}\right)\left(H^{(1)}_{i_1} \cdots H^{(N)}_{i_N}\right) \\ &= \left(G^{(1)}_{i_1} \cdots G^{(N)}_{i_N}\right) \otimes \left(H^{(1)}_{i_1} \cdots H^{(N)}_{i_N}\right) \\ &= \left(G^{(1)}_{i_1} \otimes H^{(1)}_{i_1}\right)\left(G^{(2)}_{i_2} \otimes H^{(2)}_{i_2}\right) \cdots \left(G^{(N)}_{i_N} \otimes H^{(N)}_{i_N}\right). \end{aligned}$$

Hence, $R^{U}_n = R^{T}_n R^{S}_n$.

Computing the TT

The basic algorithm for computing the TT decomposition within any accuracy ϵ is the TT-SVD. Given $\underline{T}$, it finds $\underline{S}$ that has the smallest possible ranks R^{S}_n such that

$$\|\underline{T} - \underline{S}\| \leq \epsilon \|\underline{T}\|.$$

The algorithm is based on computing SVD to obtain unfolded $\underline{G}$'s.

$\underline{\text{TT-SVD}(\underline{T}, \varepsilon)}$

$\delta \leftarrow \varepsilon (N-1)^{-1/2} \| \underline{T} \|$

$\underline{S} \leftarrow \underline{T}$; $J \leftarrow I_1 \cdot I_2 \cdots I_N$; $R_0 = 1$

<u>for</u> $n = 1, \dots, N$ <u>do</u>

$\quad S \leftarrow \text{reshape}(\underline{S}, R_{n-1} I_n, J/(R_{n-1} I_n))$

$\quad (U, \Sigma, V) \leftarrow \text{SVD}(S)$

$\quad R_n \leftarrow \arg\min_r \{ \sum_{i=r+1}^{J/(R_{n-1} I_n)} \sigma_i^2 \leq \delta \}$

$\quad \underline{G}^{(n)} \leftarrow \text{reshape}(U(:, 1{:}R_n), R_{n-1}, I_n, R_n)$ $\quad \underline{G}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$

$\quad \underline{S} \leftarrow \Sigma V^T$

$\quad J \leftarrow J R_n / (I_n R_{n-1})$

<u>end</u>

$\underline{G}^{(N)} \leftarrow \underline{S}$

<u>return</u> $(\underline{G}^{(1)}, \underline{G}^{(2)}, \dots, \underline{G}^{(N)})$

N.B. Here $\underline{G}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$, <u>not</u> $\mathbb{R}^{R_{n-1} \times R_n \times I_n}$.

Applications of TT

TT is not (often?) used for data analysis as such, as it lacks easy

interpretation of the "cars" $\underline{G}^{(1)}, G^{(2)}, \dots$
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 Tucker3.

The idea of CORCONDIA is to use the fact that CP is a special case of Tucker. Given $\underline{T}$, we can first compute CP $[\![A, B, C]\!]$ and then use these as the factor matrices for Tucker3, and find the core tensor $\underline{G}$ as

$$\underline{G} \leftarrow (A \otimes B \otimes C)^{+} \operatorname{vec}(\underline{T})$$

Now CORCONDIA statistic measures how diagonal $\underline{G}$ is

$$\mathrm{CORCONDIA} = 100\left(1 - \frac{\sum_{i,j,k}\left(g_{ijk} - \mathbb{1}(i=j=k)\right)^2}{R}\right)$$

Here

$$\mathbb{1}(i=j=k) = \begin{cases} 1 & \text{if } i=j=k \\ 0 & \text{otherwise} \end{cases}$$

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!