Introduction to Tensor Decompositions and their Applications in Machine Learning

Stephan Rabanser
Department of Informatics
Technical University of Munich
rabanser@in.tum.de

Oleksandr Shchur
Department of Informatics
Technical University of Munich
shchur@in.tum.de

Stephan Günnemann
Department of Informatics
Technical University of Munich
guennemann@in.tum.de

ABSTRACT

Tensors are multidimensional arrays of numerical values and therefore generalize matrices to multiple dimensions. While tensors first emerged in the psychometrics community in the 20th century, they have since then spread to numerous other disciplines, including machine learning. Tensors and their decompositions are especially beneficial in unsupervised learning settings, but are gaining popularity in other sub-disciplines like temporal and multi-relational data analysis, too.

The scope of this paper is to give a broad overview of tensors, their decompositions, and how they are used in machine learning. As part of this, we are going to introduce basic tensor concepts, discuss why tensors can be considered more rigid than matrices with respect to the uniqueness of their decomposition, explain the most important factorization algorithms and their properties, provide concrete examples of tensor decomposition applications in machine learning, conduct a case study on tensor-based estimation of mixture models, talk about the current state of research, and provide references to available software libraries.

1 INTRODUCTION

Tensors are generalizations of matrices to higher dimensions and can consequently be treated as multidimensional fields.

Tensors and their decompositions originally appeared in 1927 [13], but have remained untouched by the computer science community until the late 20th century [35]. Fueled by increasing computing capacity and a better understanding of multilinear algebra especially during the last decade, tensors have since expanded to other domains, like statistics, data science, and machine learning [38].

In this paper, we will first motivate the use of and need for tensors through Spearman's hypothesis and evaluate low-rank matrix decomposition approaches, while also considering the issues that come with them. We will then introduce basic tensor concepts and notation, which will lay the groundwork for the upcoming sections. In particular, we will analyze why low-rank tensor decompositions are much more rigid compared to low-rank matrix decompositions. Then, we will turn to some of the most widely used tensor decompositions, CP and Tucker, and the theory behind them, while also elaborating on their most important properties and key differences. Also, we will explain how tensor decompositions help us with uncovering underlying hidden low-dimensional structure in the tensor. Finally, we will explain why and how tensors and their decomposition can be used to tackle typical machine learning problems and afterwards look into two concrete examples of a tensor-based parameter estimation method for spherical Gaussian mixture models (GMMs) and single topic models. By using the proposed method, we can extract all needed information from low-order moments of the underlying probability distribution to learn simple GMMs and topic models in an efficient way. We will close by highlighting available tensor software libraries and by presenting the most prominent open research questions in the tensor field and recap some of the key learnings.

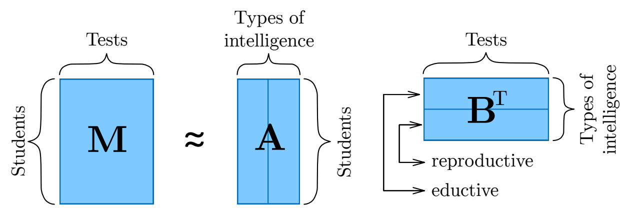

Figure 1: Spearman's hypothesis

2 MATRIX DECOMPOSITION: A MOTIVATING EXAMPLE

Matrix decompositions are important techniques used in different mathematical settings, such as the implementation of numerically efficient algorithms, the solution of linear equation systems, and the extraction of quintessential information from a matrix. As part of this section, we will focus on the *rank decomposition* of a matrix, an information extraction technique, which can be formally expressed as

$$\boldsymbol{M} = \boldsymbol{A}\boldsymbol{B}^T \quad \text{with} \quad \boldsymbol{M} \in \mathbb{R}^{n\times m},\ \boldsymbol{A} \in \mathbb{R}^{n\times r},\ \boldsymbol{B}^T \in \mathbb{R}^{r\times m} \tag{1}$$

where r represents the rank of the decomposition. Intuitively, this decomposition aims at explaining the matrix $\boldsymbol{M}$ through r different latent factors, which are encoded in the matrices $\boldsymbol{A}$ and $\boldsymbol{B}^T$.

The problem with lots of matrix factorization approaches is the fact that they are considered non-unique, meaning that a number of different matrices $\boldsymbol{A}$ and $\boldsymbol{B}^T$ can give rise to a specific $\boldsymbol{M}$ [23, 24]. In order to ensure uniqueness, additional constraints need to be imposed on a matrix, like positive-definiteness or orthogonality. In contrast, tensors do not require such strong constraints in order to offer a unique decomposition. They provide much better identifiability conditions through the usage of higher dimensions, as we will see in Section 3.

Before starting the discussion on tensors, we will briefly introduce Spearman's hypothesis and the rotation problem as an example motivating the use of tensors.

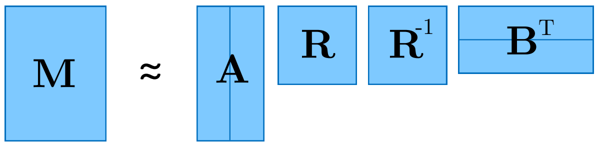

Figure 2: The rotation problem

2.1 Spearman's Hypothesis

In 1904, Charles Spearman, a British psychologist, supposed that human intelligence can be broken down into two (hidden) factors: *eductive* (the ability to make sense out of complexity) and *reproductive* (the ability to store and reproduce information) intelligence [18]. Spearman therefore was one of the first scientists to carry out an unsupervised learning technique on a data set, which is nowadays referred to as *factor analysis*.

To test his hypothesis, he invited s students to take t different tests and noted down their scores in a matrix $\boldsymbol{M} \in \mathbb{R}^{s \times t}$. He was then wondering whether it would be possible to uniquely decompose the matrix $\boldsymbol{M}$ into two smaller matrices $\boldsymbol{A} \in \mathbb{R}^{s \times h}$ and $\boldsymbol{B}^T \in \mathbb{R}^{h \times t}$ through the $h = 2$ latent factors described above. According to his explanation, $\boldsymbol{A}$ would hold a list of students with their corresponding eductive and reproductive capabilities and $\boldsymbol{B}^T$ would hold a list of tests with their corresponding eductive and reproductive requirements. This problem setting is depicted in Figure 1.

2.2 The Rotation Problem

Given a matrix $\boldsymbol{M}$, we would like to approximate it as well as possible with another matrix $\hat{\boldsymbol{M}}$ of a lower rank (for Spearman's hypothesis: rank$(\hat{\boldsymbol{M}}) = 2$). Formally, the objective can be defined as minimizing the norm of the difference between the two matrices:

$$\min_{\hat{\boldsymbol{M}}} ||\boldsymbol{M} - \hat{\boldsymbol{M}}|| \quad \text{with} \quad \hat{\boldsymbol{M}} = \boldsymbol{A}\boldsymbol{B}^T \tag{2}$$

However, this decomposition is not unique. By inserting an invertible rotation matrix $\boldsymbol{R}$ together with its inverse $\boldsymbol{R}^{-1}$ between $\boldsymbol{A}$ and $\boldsymbol{B}^T$ and absorbing $\boldsymbol{R}$ on the left with $\boldsymbol{A}$ and $\boldsymbol{R}^{-1}$ on the right with $\boldsymbol{B}^T$ we can again construct two matrices $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{B}}^T$ [23, 24]. This problem is usually referred to as the *rotation problem* and is depicted in Figure 2.

$$\begin{aligned} \hat{\boldsymbol{M}} = \boldsymbol{A}\boldsymbol{B}^T &= \boldsymbol{A}\boldsymbol{R}\boldsymbol{R}^{-1}\boldsymbol{B}^T = (\boldsymbol{A}\boldsymbol{R})(\boldsymbol{R}^{-1}\boldsymbol{B}^T) \\ &= (\boldsymbol{A}\boldsymbol{R})(\boldsymbol{B}\boldsymbol{R}^{-T})^T = \tilde{\boldsymbol{A}}\tilde{\boldsymbol{B}}^T \end{aligned} \tag{3}$$

We have seen that the rank decomposition of a matrix is generally highly non-unique. We conclude that matrix decompositions are only unique under very stringent conditions, such as orthogonality constraints which are imposed by the singular value decomposition (SVD) [24]. Soon we will see that tensor decompositions are usually unique under much milder conditions.

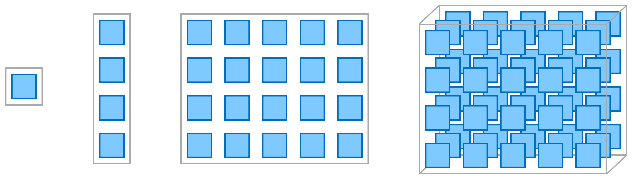

Figure 3: $x \in \mathbb{R}$, $\boldsymbol{x} \in \mathbb{R}^4$, $\boldsymbol{X} \in \mathbb{R}^{4\times5}$, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{4\times5\times3}$

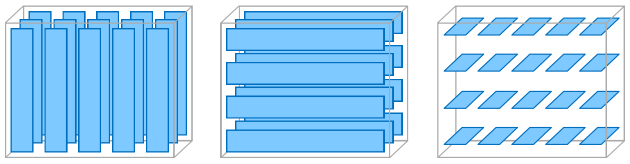

Figure 4: Column, row, and tube fibers of a mode-3 tensor

3 INTRODUCTION TO TENSORS

3.1 Basics

As we have already learned, tensors can be thought of as multi-way collections of numbers, which typically come from a field (like $\mathbb{R}$). In the simplest high-dimensional case, such a tensor would be a three-dimensional array, which can be thought of as a data cube. Throughout this paper, we will often refer to a three-dimensional tensor for motivation and simplicity. In most cases, the notation naturally extends to higher-dimensional tensors. As we introduce different concepts in this and the next sections, we will borrow most of our notation from the comprehensive reviews of Kolda et al. [38] and Sidiropoulos et al. [35].

3.1.1 Tensor Order. The *order*[1] of a tensor is the number of its dimensions. Scalars can therefore be interpreted as zeroth-order tensors, vectors as first-order tensors, and matrices as second-order tensors. We will refer to tensors of order three or higher as higher-order tensors. Notation-wise, scalars are denoted by lower case letters $x \in \mathbb{R}$, vectors by lower case bold letters $\boldsymbol{x} \in \mathbb{R}^{I_1}$, matrices by upper case bold letters $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2}$, and higher order tensors by upper case bold Euler script letters $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$. The Is denote the number of elements in the respective dimension. Figure 3 shows how we can move from scalars to tensors.

3.1.2 Tensor Indexing. We can create subarrays (or subfields) by fixing some of the given tensor's indices. *Fibers* are created when fixing all but one index, *slices* (or slabs) are created when fixing all but two indices. For a third order tensor the fibers are given as $\boldsymbol{x}_{:jk} = \boldsymbol{x}_{jk}$ (column), $\boldsymbol{x}_{i:k}$ (row), and $\boldsymbol{x}_{ij:}$ (tube); the slices are given as $\boldsymbol{X}_{::k} = \boldsymbol{X}_k$ (frontal), $\boldsymbol{X}_{:j:}$ (lateral), $\boldsymbol{X}_{i::}$ (horizontal). Graphical examples of fibers and slices for a 3-way tensor are given in Figure 4 and 5.

3.1.3 Outer and Inner Product. The *vector outer product* is defined as the product of the vector's elements. This operation is denoted by the $\circledcirc$ symbol[2]. The vector outer product of two n-sized

[1]The order of a tensor is sometimes also referred to as its *way* or *mode*.
[2]Some publications denote the tensor product with the $\otimes$ symbol which we will use to denote the Kronecker product.

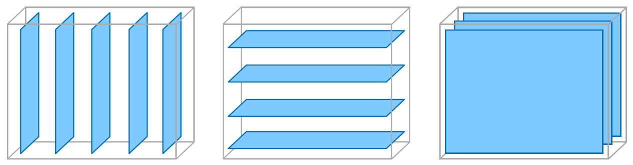

Figure 5: Lateral, horizontal, and frontal slices of a mode-3 tensor

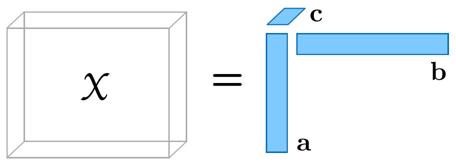

Figure 6: A rank-1 mode-3 tensor

vectors $\boldsymbol{a}, \boldsymbol{b}$ is defined as follows and produces a matrix $\boldsymbol{X}$:

$$\boldsymbol{X} = \boldsymbol{a} \circledcirc \boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}^T \tag{4}$$

By extending the vector outer product concept to the general *tensor product* for N vectors, we can produce a tensor $\mathcal{X}$:

$$\mathcal{X} = \boldsymbol{a}^{(1)} \circledcirc \boldsymbol{a}^{(2)} \circledcirc \cdots \circledcirc \boldsymbol{a}^{(N)} \text{ with } x_{i_1 i_2 \cdots i_N} = a^{(1)}_{i_1} a^{(2)}_{i_2} \cdots a^{(N)}_{i_N} \tag{5}$$

In contrast to the outer product, the *inner product* of two n-sized vectors $\boldsymbol{a}, \boldsymbol{b}$ is defined as

$$x = \langle \boldsymbol{a}, \boldsymbol{b} \rangle = \boldsymbol{a}^T \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \tag{6}$$

and produces a scalar x.

3.1.4 Rank-1 Tensors. A N-way tensor is of *rank-1* if it can be strictly decomposed into the outer product of N vectors. Intuitively, this means that we introduce different scalings of a sub-tensor as we add more dimensions when building up the complete tensor. A rank-one matrix can therefore be written as $\boldsymbol{X} = \boldsymbol{a} \circledcirc \boldsymbol{b}$ and a rank-one 3-way tensor as $\mathcal{X} = \boldsymbol{a} \circledcirc \boldsymbol{b} \circledcirc \boldsymbol{c}$. The general N-way form was already introduced in Equation (5). A graphical view of the rank-1 concept is given in Figure 6.

3.1.5 Tensor Rank. The *rank* of a tensor $\text{rank}(\mathcal{X}) = R$ is defined as the minimum number of rank-one tensors which are needed to produce $\mathcal{X}$ as their sum. A rank-R matrix can therefore be written as $\boldsymbol{X} = \sum_{r=1}^{R} \lambda_r \boldsymbol{a}_r \circledcirc \boldsymbol{b}_r = [\![\boldsymbol{\lambda}; \boldsymbol{A}, \boldsymbol{B}]\!]$ and a rank-R 3-way tensor as $\mathcal{X} = \sum_{r=1}^{R} \lambda_r \boldsymbol{a}_r \circledcirc \boldsymbol{b}_r \circledcirc \boldsymbol{c}_r = [\![\boldsymbol{\lambda}; \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!]$. The general N-way form is given as:

$$\boxed{\begin{aligned}\mathcal{X} &= \sum_{r=1}^{R} \lambda_r \boldsymbol{a}^{(1)}_r \circledcirc \boldsymbol{a}^{(2)}_r \circledcirc \cdots \circledcirc \boldsymbol{a}^{(N)}_r \\ &= [\![\boldsymbol{\lambda}; \boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \cdots, \boldsymbol{A}^{(N)}]\!]\end{aligned}} \tag{7}$$

The $\boldsymbol{A}$s are called *factor matrices* and hold the combination of the vectors from the rank-one components as columns. A specific $\boldsymbol{A}$ therefore has the form $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_R \end{bmatrix}$.

We introduced a new additional factor λ_r which is often used to absorb the respective weights during normalization of the factor matrices' columns. This usually means normalizing the sum of the squares of the elements in each column to one. Note that $\boldsymbol{\lambda} \in \mathbb{R}^R$. This notation will be especially useful once turn to machine learning applications of tensor decompositions in Section 5 and once we introduce the Tucker decomposition in Section 4.2.

3.2 Tensor Reorderings

3.2.1 Vectorization. We can turn a given matrix $\boldsymbol{X} \in \mathbb{R}^{I \times J}$ into a vector by vertically stacking the columns of $\boldsymbol{X}$ into a tall vector.

$$\text{vec}(\boldsymbol{X}) = \begin{bmatrix} \boldsymbol{x}_{:1} \\ \boldsymbol{x}_{:2} \\ \vdots \\ \boldsymbol{x}_{:J} \end{bmatrix} \tag{8}$$

3.2.2 Matricization. Analogously, *matricization* is the operation that reorders a tensor into a matrix. While there are other ways of rearranging vector elements into a matrix, we will only look into the mode-n matricization of a tensor. The mode-n matricization (or unfolding) of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, denoted $\boldsymbol{X}_{(n)} \in \mathbb{R}^{I_n \times (I_1 \cdot \ldots \cdot I_{n-1} \cdot I_{n+1} \cdot \ldots \cdot I_N)}$, turns the mode-$n$ fibers of $\mathcal{X}$ into the columns of $\boldsymbol{X}_{(n)}$.

Let $x \in \mathcal{X}$ be an element of a tensor and $m \in \boldsymbol{M}$ be an element of the unfolded tensor. Then we can define the mode-n matricization via the following mapping:

$$x_{i_1, i_2, \cdots, i_N} \mapsto m_{i_n, j} \text{ with } j = 1 + \sum_{\substack{k=1 \\ k \neq n}}^{N} \left((i_k - 1) \prod_{\substack{m=1 \\ m \neq n}}^{k-1} I_m \right) \tag{9}$$

To illustrate the formula introduced above, we will present a brief example of the matricization of a third-order tensor from [38]. Let $\mathcal{X}$ be a tensor with the following frontal slices:

$$\boldsymbol{X}_1 = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix} \quad \boldsymbol{X}_2 = \begin{bmatrix} 13 & 16 & 19 & 22 \\ 14 & 17 & 20 & 23 \\ 15 & 18 & 21 & 24 \end{bmatrix}$$

Then the three mode-n matricizations are:

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

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

$$\boldsymbol{X}_{(3)} = \begin{bmatrix} 1 & 2 & 3 & 4 & \cdots & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & \cdots & 21 & 22 & 23 & 12 \end{bmatrix}$$

3.3 Important Matrix/Tensor Products

3.3.1 Kronecker Product. The *Kronecker product* between two arbitrarily-sized matrices $\boldsymbol{A} \in \mathbb{R}^{I \times J}$ and $\boldsymbol{B} \in \mathbb{R}^{K \times L}$, $\boldsymbol{A} \otimes \boldsymbol{B} \in \mathbb{R}^{(IK) \times (JL)}$, is a generalization of the outer product from vectors to matrices:

$$A \otimes B := \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1J}B \\ a_{21}B & a_{22}B & \cdots & a_{2J}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}B & a_{I2}B & \cdots & a_{IJ}B \end{bmatrix} = \begin{bmatrix} a_1 \otimes b_1 & a_1 \otimes b_2 & \cdots & a_J \otimes b_{L-1} & a_J \otimes b_L \end{bmatrix} \tag{10}$$

3.3.2 Khatri-Rao Product. The *Khatri-Rao product* between two matrices $A \in \mathbb{R}^{I\times K}$ and $B \in \mathbb{R}^{J\times K}$, $A \odot B \in \mathbb{R}^{(IJ)\times K}$, corresponds to the column-wise Kronecker product.

$$A \odot B := \begin{bmatrix} a_1 \otimes b_1 & a_2 \otimes b_2 & \cdots & a_K \otimes b_K \end{bmatrix} \tag{11}$$

3.3.3 Hadamard Product. The *Hadamard product* between two same-sized matrices $A \in \mathbb{R}^{I\times J}$ and $B \in \mathbb{R}^{I\times J}$, $A * B \in \mathbb{R}^{I\times J}$, corresponds to the element-wise matrix product.

$$A * B := \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1J}b_{1J} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2J}b_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}b_{I1} & a_{I2}b_{I2} & \cdots & a_{IJ}b_{IJ} \end{bmatrix} \tag{12}$$

3.3.4 n-mode Product. The *n-mode product* of a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\ldots\times I_N}$ and a matrix $M \in \mathbb{R}^{J\times I_n}$ is denoted by $\mathcal{Y} = \mathcal{X} \times_n M$ with $\mathcal{Y} \in \mathbb{R}^{I_1\times\cdots\times I_{n-1}\times J\times I_{n+1}\times\cdots\times I_N}$. Intuitively, each mode-n fiber is multiplied by the matrix M. The n-mode product can also be expressed through matricized tensors as $Y_{(n)} = MX_{(n)}$. Element-wise, this operation can be expressed as follows:

$$(\mathcal{X} \times_n M)_{i_1\cdots i_{n-1} j i_{n+1}\cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1\cdots i_N} m_{j i_n} \tag{13}$$

The n-mode product also exists for tensors and vectors. The n-mode product of a tensor and a vector $v \in \mathbb{R}^{I_n}$ is denoted by $\mathcal{Y} = \mathcal{X} \times_n v$. Intuitively, each mode-n fiber is multiplied by the vector v. Element-wise, this operation can be expressed as follows:

$$(\mathcal{X} \times_n v)_{i_1\cdots i_{n-1} i_{n+1}\cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1\cdots i_N} v_{i_n} \tag{14}$$

3.3.5 Multilinear Tensor Transformation. A tensor $\mathcal{X}$ can be transformed on multiple dimensions by *hitting* each of the vectors producing the tensor with a transformation matrix (or vector) from the left side [1]. Hitting a tensor $\mathcal{X}$ with a matrix M_i on the i-th dimension corresponds to a matrix-vector multiplication between M_i and the vector on the i-th dimension, $M_i^T a^{(i)}$, thereby ensuring that the result of each individual transformation will again result in a vector. Instead, hitting a tensor $\mathcal{X}$ with a vector v_i on the i-th dimension corresponds to an inner product between v_i and the vector on the i-th dimension, $\langle v_i, a^{(i)} \rangle$, resulting in a scalar.

In the 3-dimensional case, the multilinear tensor transformation using matrices is defined as follows:

$$\boxed{\tilde{\mathcal{X}} = \mathcal{X}(M_1, M_2, M_3) = \sum_{r=1}^{R} \lambda_r (M_1^T a_r) \circledcirc (M_2^T b_r) \circledcirc (M_3^T c_r)} \tag{15}$$

This equation will prove valuable when we turn to tensor-based estimation of mixture models in Section 6.

3.4 Tensor Uniqueness and Rigidness

A tensor decomposition is called *unique* if there exists only one combination of rank-1 tensors that sum to $\mathcal{X}$ up to a common scaling and/or permutation indeterminacy. Intuitively, this means that there is one and only one decomposition and we can not construct a different arrangement of rank-1 tensors that sum to $\mathcal{X}$. As we will see below, tensor decompositions are unique under much more relaxed requirements on the tensor compared to matrices and hence are considered more rigid than matrices.

Since we are usually interested in a low-rank tensor decomposition of $\mathcal{X}$, let us now take a look at an interesting property of low-rank tensors. Given a low-rank tensor $\mathcal{X}$, then any slice through that tensor

$$X_k = \sum_{r=1}^{R} (a_r \circledcirc b_r) c_{kr} \tag{16}$$

is in itself a low-rank matrix again. A low-rank tensor is therefore not just a collection of low-rank matrices, but there exist interrelations between these slices. We can easily observe, that all of these slices also share the same column and row spaces. Looking more closely, we can make an even stronger claim, namely that the slices are different scalings of the same set of rank-1 matrices [23, 24]. This is a very strong constraint on the tensor's structure, which could help us address the rotation problem we described in Section 2.2.

The low rank assumption enables us to determine whether the factors we found capture the underlying (latent) structure of the tensor. To do that we can subtract off scalings of the same rank-1 matrix, $X_k - c_k(a \circledcirc b)$, to decrease the rank of each slice of the tensor [23, 24]. For matrices, there are many possible low-rank matrices one could use to strictly reduce its rank, but for tensors it is required that the same low-rank matrix works for all tensor slices. This strong interconnection between their slices makes tensors way more *rigid* than matrices and allows for weaker uniqueness conditions.

4 TENSOR DECOMPOSITION ALGORITHMS

After familiarizing ourselves with the basics of tensors, we will now turn to the most popular tensor decomposition algorithms. While this section will provide a rather theoretical treatment, we also describe the practical applicability of tensor decompositions in Section 5 and Section 6.

In particular, we are wondering whether we can generalize the concept of the SVD from matrices to general tensors. As we will see, there is no single generalization of the SVD concept, but we will discuss two decompositions that feature different generalized properties of the matrix SVD: the *canonical polyadic decomposition* (CPD) and the *Tucker decomposition*. Both are outer product decompositions, but they have very different structural properties. As a rule of thumb it is usually advised to use CPD for latent parameter estimation and Tucker for subspace estimation, compression, and dimensionality reduction.

Since CPD and Tucker are the most important tensor decomposition and many other decompositions are based on these two techniques, discussing any other factorization approaches would go beyond the scope of this paper. Just like in Section 3, we will

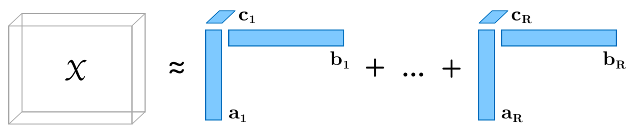

Figure 7: CP Decomposition

introduce these decompositions from Sidiropoulos et al. [35] and Kolda et al. [38], which also provide deeper theoretical insights and the most important adaptions of the mentioned decompositions.

4.1 Canonical Polyadic Decomposition (CPD)

The first type of tensor decomposition we will consider are rank decompositions. The key concept of rank decompositions is to express a tensor as the sum of a finite number of rank-one tensors. The most prominent rank decompositions are the CANonical DECOMPosition (CANDECOMP) and the PARAllel FACtors (PARAFAC) decomposition. Both have their origins in different knowledge domains and have been independently discovered many times, but they both boil down to the same principles, which is why we will from here on refer to this type of decomposition as the CANDECOMP/PARAFAC or canonical polyadic decomposition (CPD).

We can formalize the 3-way CPD case as follows:

$$\boxed{\min_{\hat{\mathcal{X}}} ||\mathcal{X} - \hat{\mathcal{X}}|| \quad \text{where} \quad \hat{\mathcal{X}} = \sum_{r=1}^{R} \boldsymbol{a}_r \circledcirc \boldsymbol{b}_r \circledcirc \boldsymbol{c}_r = [\![\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!]} \quad (17)$$

A graphical view of this concept is given in Figure 7.

Note the very similar structure to the matrix decomposition problem in (2). In the exact case, meaning that if $\min_{\hat{\mathcal{X}}} ||\mathcal{X} - \hat{\mathcal{X}}|| = 0$, we refer to $\hat{\mathcal{X}}$ being an exact low rank approximation to $\mathcal{X}$. We can restate this problem in a matricized form, too:

$$\begin{aligned} \hat{X}_{(1)} &= (\boldsymbol{C} \odot \boldsymbol{B})\boldsymbol{A}^T \\ \hat{X}_{(2)} &= (\boldsymbol{C} \odot \boldsymbol{A})\boldsymbol{B}^T \\ \hat{X}_{(3)} &= (\boldsymbol{B} \odot \boldsymbol{A})\boldsymbol{C}^T \end{aligned} \quad (18)$$

For the general case we have:

$$\hat{\mathcal{X}} = \sum_{r=1}^{R} \lambda_r \boldsymbol{a}_r^{(1)} \circledcirc \boldsymbol{a}_r^{(2)} \circledcirc \cdots \circledcirc \boldsymbol{a}_r^{(n)} = [\![\boldsymbol{\lambda}; \boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \cdots, \boldsymbol{A}^{(n)}]\!] \quad (19)$$

$$\hat{X}_{(n)} = \Lambda(\boldsymbol{A}^{(N)} \odot \cdots \odot \boldsymbol{A}^{(n+1)} \odot \boldsymbol{A}^{(n-1)} \odot \cdots \odot \boldsymbol{A}^{(1)})\boldsymbol{A}^{(n)T} \quad (20)$$

Note that $\boldsymbol{\Lambda} = \text{Diag}(\boldsymbol{\lambda})$.

There exist different algorithms to compute CPD of a given tensor and we will take a brief look at two of the most popular ones.

4.1.1 Jennrich's Algorithm. If $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are all linearly independent (i.e. have full rank), then $\mathcal{X} = \sum_{r=1}^{R} \lambda_r \boldsymbol{a}_r \circledcirc \boldsymbol{b}_r \circledcirc \boldsymbol{c}_r$ is unique up to trivial rank permutation and feature scaling and we can use Jennrich's algorithm to recover the factor matrices [23, 24]. The algorithm works as follows:

(1) Choose random vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.
(2) Take a slice through the tensor by hitting the tensor with the random vector $\boldsymbol{x}$:
$\mathcal{X}(\boldsymbol{I}, \boldsymbol{I}, \boldsymbol{x}) = \sum_{r=1}^{R} \langle \boldsymbol{c}_r, \boldsymbol{x} \rangle \boldsymbol{a}_r \circledcirc \boldsymbol{b}_r = \boldsymbol{A}\text{Diag}(\langle \boldsymbol{c}_r, \boldsymbol{x} \rangle)\boldsymbol{B}^T$.
(3) Take a second slice through the tensor by hitting the tensor with the random vector $\boldsymbol{y}$:
$\mathcal{X}(\boldsymbol{I}, \boldsymbol{I}, \boldsymbol{y}) = \sum_{r=1}^{R} \langle \boldsymbol{c}_r, \boldsymbol{y} \rangle \boldsymbol{a}_r \circledcirc \boldsymbol{b}_r = \boldsymbol{A}\text{Diag}(\langle \boldsymbol{c}_r, \boldsymbol{y} \rangle)\boldsymbol{B}^T$.
(4) Compute eigendecomposition to find $\boldsymbol{A}$:
$\mathcal{X}(\boldsymbol{I}, \boldsymbol{I}, \boldsymbol{x})\, \mathcal{X}(\boldsymbol{I}, \boldsymbol{I}, \boldsymbol{y})^\dagger = \boldsymbol{A}\text{Diag}(\langle \boldsymbol{c}_r, \boldsymbol{x} \rangle)\text{Diag}(\langle \boldsymbol{c}_r, \boldsymbol{y} \rangle)^\dagger \boldsymbol{A}^\dagger$
(5) Compute eigendecomposition to find $\boldsymbol{B}$:
$\mathcal{X}(\boldsymbol{I}, \boldsymbol{I}, \boldsymbol{x})^\dagger\, \mathcal{X}(\boldsymbol{I}, \boldsymbol{I}, \boldsymbol{y}) = (\boldsymbol{B}^T)^\dagger \text{Diag}(\langle \boldsymbol{c}_r, \boldsymbol{x} \rangle)^\dagger \text{Diag}(\langle \boldsymbol{c}_r, \boldsymbol{y} \rangle)\boldsymbol{B}^T$
(6) Pair up the factors and solve a linear system to find $\boldsymbol{C}$.

While this algorithm works well for some problems, it only takes random slices of a tensor and hence does not use the full tensor structure. Moreover, it requires good eigen-gap[3] on the eigendecompositions of the factor matrices, the lack of which could lead to numerical instability [14].

4.1.2 Alternating Least Squares (ALS) Algorithm. An other way of computing the CP decomposition of a tensor, and in fact the work-horse of modern tensor decomposition approaches, is the *alternating least squares* (ALS) algorithm. The key idea behind this algorithm is to fix all factor matrices except for one in order to optimize for the non-fixed matrix and then repeat this step for every matrix repeatedly until some stopping criterion is satisfied.

For the 3-way tensor case, the ALS algorithm would perform the following steps repeatedly until convergence.

$$\boxed{\begin{aligned} \boldsymbol{A} &\leftarrow \arg\min_{\boldsymbol{A}} ||X_{(1)} - (\boldsymbol{C} \odot \boldsymbol{B})\boldsymbol{A}^T|| \\ \boldsymbol{B} &\leftarrow \arg\min_{\boldsymbol{B}} ||X_{(2)} - (\boldsymbol{C} \odot \boldsymbol{A})\boldsymbol{B}^T|| \\ \boldsymbol{C} &\leftarrow \arg\min_{\boldsymbol{C}} ||X_{(3)} - (\boldsymbol{B} \odot \boldsymbol{A})\boldsymbol{C}^T|| \end{aligned}} \quad (21)$$

The optimal solution to this minimization problem is given by

$$\begin{aligned} \hat{\boldsymbol{A}} &= X_{(1)}[(\boldsymbol{C} \odot \boldsymbol{B})^T]^\dagger = X_{(1)}(\boldsymbol{C} \odot \boldsymbol{B})(\boldsymbol{C}^T\boldsymbol{C} * \boldsymbol{B}^T\boldsymbol{B})^\dagger \\ \hat{\boldsymbol{B}} &= X_{(2)}[(\boldsymbol{C} \odot \boldsymbol{A})^T]^\dagger = X_{(2)}(\boldsymbol{C} \odot \boldsymbol{A})(\boldsymbol{C}^T\boldsymbol{C} * \boldsymbol{A}^T\boldsymbol{A})^\dagger \\ \hat{\boldsymbol{C}} &= X_{(3)}[(\boldsymbol{B} \odot \boldsymbol{A})^T]^\dagger = X_{(3)}(\boldsymbol{B} \odot \boldsymbol{A})(\boldsymbol{B}^T\boldsymbol{B} * \boldsymbol{A}^T\boldsymbol{A})^\dagger \end{aligned} \quad (22)$$

The generalization to the order-N case is given below. Note that due to the problem definition in Equation (17), the ALS algorithm requires the rank which should be used for the approximation as an argument [38].

Algorithm 1 ALS algorithm

```
procedure CP-ALS($\mathcal{X}$, R)
    initialize $A^{(n)} \in R^{I_n \times R}$ for $n = 1, \ldots, N$
    repeat
        for n = 1,...,N do
            $V \leftarrow A^{(1)T}A^{(1)} * \cdots * A^{(n-1)T}A^{(n-1)} * A^{(n+1)T}A^{(n+1)} *$
            $\hookrightarrow \cdots * A^{(N)T}A^{(N)}$
            $A^{(n)} \leftarrow X_{(n)}(A^{(N)} \odot \cdots \odot A^{(n+1)} \odot A^{(n-1)} \odot \cdots \odot A^{(1)})V^\dagger$
            normalize columns of $A^{(n)}$ (optional)
            store norms as $\lambda$ (optional)
        end for
    until stopping criterion satisfied
    return $\lambda$, $A^{(1)}, \ldots, A^{(N)}$
end procedure
```

[3]difference between two successive eigenvalues

While the ALS algorithm outlined in Algorithm 1 is simple to understand and implement, it might take several steps to converge and it might also not converge to a global optimum. This means that the performance of this algorithm is influenced by its initialization.

4.1.3 Tensor Power Method. The last CPD algorithm we will consider is the so-called *tensor power method*, which we can use in the special case of all identical factor matrices and if the $\boldsymbol{a}_r$s are all orthogonal. This restricts the tensor we want to decompose to the following structure in the 3-way case: $\mathcal{X} = \sum_{r=1}^{R} \lambda_r \boldsymbol{a}_r \circledcirc \boldsymbol{a}_r \circledcirc \boldsymbol{a}_r$ [2]. Note that the tensor power method is an analogue to the matrix power method. While the latter aims at finding the top singular vectors in a matrix, the former tries to determine the top singular vectors in a tensor.

The core idea of the matrix power method is described by the following recurrence relation, which computes an approximation $\boldsymbol{a}_{r,i+1}$ to the eigenvector $\boldsymbol{a}_r$ corresponding to the most dominant eigenvalue λ_r based on $\boldsymbol{X}$ [30]:

$$\boldsymbol{a}_{r,i+1} = \frac{\boldsymbol{X}_r(\boldsymbol{I}, \boldsymbol{a}_{r,i})}{||\boldsymbol{X}_r(\boldsymbol{I}, \boldsymbol{a}_{r,i})||_2} = \frac{\boldsymbol{X}_r \boldsymbol{a}_{r,i}}{||\boldsymbol{X}_r \boldsymbol{a}_{r,i}||_2} \tag{23}$$

This approximation exploits the eigenvalue/-vector relationship $\boldsymbol{X}\boldsymbol{a}_r = \boldsymbol{X}(\boldsymbol{I}, \boldsymbol{a}_r) = \lambda_r \boldsymbol{a}_r$. Note that the division by the L_2-normalized matrix-vector product allows us to get a better feeling for the convergence and that the initial vector $\boldsymbol{a}_{r,0}$ can be picked randomly or can be initialized with some correlation to the true eigenvector (if such knowledge is available) [30]. The top singular value λ_r can be computed from $\boldsymbol{a}_{r,i}$ using normalization, formally $\lambda_r = ||\boldsymbol{a}_{r,i}||_2$, after convergence.

As we might be interested in not just extracting the very first dominant eigenvalue/-vector combination (r const.), but the first k ($r \in [k]$) [4] or all such combinations ($r \in [R]$), it is important to *deflate* the matrix once the dominant eigenvalue/-vector combination has been found [1]. This means removing λ_r and $\boldsymbol{a}_r$ from the matrix as follows:

$$\boldsymbol{X}_{r+1} = \boldsymbol{X}_r - \lambda_r \boldsymbol{a}_r \circledcirc \boldsymbol{a}_r \tag{24}$$

In order to transfer this concept to tensors, we only need to perform a few small adjustments. The two crucial equations are given as

$$\boxed{\begin{aligned} \boldsymbol{a}_{r,i+1} &= \frac{\mathcal{X}_r(\boldsymbol{I}, \boldsymbol{a}_{r,i}, \boldsymbol{a}_{r,i})}{||\mathcal{X}_r(\boldsymbol{I}, \boldsymbol{a}_{r,i}, \boldsymbol{a}_{r,i})||_2} \\ \mathcal{X}_{r+1} &= \mathcal{X}_r - \lambda_r \boldsymbol{a}_r \circledcirc \boldsymbol{a}_r \circledcirc \boldsymbol{a}_r \end{aligned}} \tag{25}$$

where

$$\mathcal{X}_r(\boldsymbol{I}, \boldsymbol{a}_{r,i}, \boldsymbol{a}_{r,i}) = \sum_{j \in [k]} \lambda_{r,j} \langle \boldsymbol{a}_{r,j}, \boldsymbol{a}_{r,i} \rangle^2 \boldsymbol{a}_{r,j} = \lambda_r \boldsymbol{a}_{r,i} \tag{26}$$

which follows from Equation (15) [1]. Section 6.6 will give an example of how the tensor power method can be applied to a concrete machine learning problem.

4.1.4 Uniqueness. As we have seen in Section 2.2, rank decompositions are (generally) not unique for matrices, whereas this is often the case for higher-order tensors. In order to further analyze this property, we first have to introduce the concept of k-rank. The *k-rank* (or Kruskal rank) of a matrix $\boldsymbol{M}$, denoted $k_{\boldsymbol{M}}$, corresponds to the maximum number k such that **any** k columns are linearly independent. A sufficient uniqueness condition for the general CPD case would then be:

$$\sum_{n=1}^{N} k_{\boldsymbol{A}^{(n)}} \geq 2R + (N-1) \tag{27}$$

While the the above equation provides a sufficient condition for uniqueness, it is not necessary. Necessary conditions for uniqueness are:

$$\min_{1,\ldots,N} \text{rank}(\boldsymbol{A}^{(1)} \odot \cdots \odot \boldsymbol{A}^{(n-1)} \odot \boldsymbol{A}^{(n+1)} \odot \cdots \odot \boldsymbol{A}^{(1)}) = R \tag{28}$$

$$\min_{1,\ldots,N} \left(\prod_{\substack{m=1 \\ m \neq n}}^{N} \text{rank}(\boldsymbol{A}^{(m)}) \right) \geq R \tag{29}$$

4.1.5 Tensor Rank Peculiarities. Finally, since we are looking at a rank decomposition, we still want to shed some light on some tensor rank peculiarities. While the definition of tensor rank is a natural extension of the matrix rank, some of its properties are noticeably different.

- There is no trivial algorithm to determine the rank of a tensor as the problem is NP-hard [12]. Most algorithms therefore try to determine the fit for multiple CPDs and then pick the one which yields the best approximation. We note though that in practice a 100% fit is close to impossible, as data is usually corrupted by noise and in fact the fit alone cannot determine the rank in such cases.
- The rank of a tensor with real entries can be different depending on the field it is defined over. For $\mathcal{X} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ and $\hat{\mathcal{X}} \in \mathbb{C}^{I_1 \times \ldots \times I_N}$ where $\mathcal{X} = \hat{\mathcal{X}}$ it is possible that $\text{rank}(\mathcal{X}) \neq \text{rank}(\hat{\mathcal{X}})$.
- The rank of a three-way tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ is bounded from above by the following inequality:

$$\text{rank}(\mathcal{X}) \leq \min\{IJ, IK, JK\} \tag{30}$$

- The rank of a tensor is constant under mode permutation.
- While the best rank-k approximation for a matrix is given by the truncated SVD, this is not true for tensors. In fact it is even possible that the best rank-k approximation of a tensor does not exist at all. We call such tensors degenerate, which means that they can be approximated arbitrarily well by a lower-rank factorization. More formally, a tensor $\mathcal{X}$ of $\text{rank}(\mathcal{X}) = m$ is called degenerate if

$$\forall \epsilon > 0 \quad \exists \hat{\mathcal{X}}, \ \text{rank}(\hat{\mathcal{X}}) = k < m \ \text{ s.t. } ||\mathcal{X} - \hat{\mathcal{X}}|| < \epsilon \tag{31}$$

4.2 Tucker Decomposition

The Tucker decomposition decomposes a tensor into a so-called core tensor and multiple matrices which correspond to different core scalings along each mode. Therefore, the Tucker decomposition can be seen as a higher-order PCA.

In the 3-way tensor case, we can express the problem of finding the Tucker decomposition of a tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ with $\mathcal{G} \in$

[4] $x \in [k] \equiv x \in \{0, 1, \ldots, k\}$

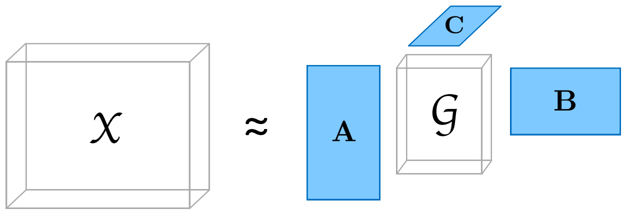

Figure 8: Tucker Decomposition

$\mathbb{R}^{P\times Q\times R}$, $\boldsymbol{A} \in \mathbb{R}^{I\times P}$, $\boldsymbol{B} \in \mathbb{R}^{J\times Q}$, $\boldsymbol{C} \in \mathbb{R}^{K\times R}$ as follows:

$$\boxed{\begin{aligned} \min_{\hat{\mathcal{X}}} \|\mathcal{X} - \hat{\mathcal{X}}\| \quad \text{with} \quad \hat{\mathcal{X}} &= \sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{r=1}^{R} g_{pqr}\boldsymbol{a}_r \circledcirc \boldsymbol{b}_r \circledcirc \boldsymbol{c}_r \\ &= \mathcal{G} \times_1 \boldsymbol{A} \times_2 \boldsymbol{B} \times_3 \boldsymbol{C} \\ &= [\![\mathcal{G}; \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!] \end{aligned}} \tag{32}$$

A graphical view of this concept is given in Figure 8.

In this setting, $\mathcal{G}$ is the core tensor, which expresses how and to which extend different tensor elements interact with each other. The factor matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are often referred to as the principal component in the respective tensor mode. We can already see, that if we pick $P < I$, $Q < J$, and $R < K$, this will result in a compression of $\mathcal{X}$, with $\mathcal{G}$ being the compressed version of $\mathcal{X}$.

The matricized version of the above tensor is given as:

$$\begin{aligned} \hat{\boldsymbol{X}}_{(1)} &= \boldsymbol{A}\boldsymbol{G}_{(1)}(\boldsymbol{C} \otimes \boldsymbol{B})^T \\ \hat{\boldsymbol{X}}_{(2)} &= \boldsymbol{B}\boldsymbol{G}_{(1)}(\boldsymbol{C} \otimes \boldsymbol{A})^T \\ \hat{\boldsymbol{X}}_{(3)} &= \boldsymbol{C}\boldsymbol{G}_{(1)}(\boldsymbol{B} \otimes \boldsymbol{A})^T \end{aligned} \tag{33}$$

In the general N-way case we get:

$$\begin{aligned} \hat{\mathcal{X}} &= \sum_{r_1=1}^{R_1}\sum_{r_2=1}^{R_2}\cdots\sum_{r_N=1}^{R_N} g_{r_1 r_2 \cdots r_N}\boldsymbol{a}^{(1)}_{i_1 r_1} \circledcirc \cdots \circledcirc \boldsymbol{a}^{(N)}_{i_N r_N} \\ &= \mathcal{G} \times_1 \boldsymbol{A}^{(1)} \times_2 \cdots \times_N \boldsymbol{A}^{(N)} \\ &= [\![\mathcal{G}; \boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}]\!] \end{aligned} \tag{34}$$

$$\hat{\boldsymbol{X}}_{(n)} = \boldsymbol{A}^{(n)}\boldsymbol{G}_{(n)}(\boldsymbol{A}^{(N)} \otimes \cdots \otimes \boldsymbol{A}^{(n+1)} \otimes \boldsymbol{A}^{(n-1)} \otimes \cdots \otimes \boldsymbol{A}^{(1)})^T \tag{35}$$

Before we can look into the exact computation of the Tucker decomposition, we first need to introduce the concept of n-rank. The n-rank of a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\ldots\times I_N}$ corresponds to the column rank of the n-th unfolding of the tensor $\boldsymbol{X}_{(n)}$. Formally, this is denoted $\text{rank}_n(\mathcal{X})$. It is important to not confuse this concept with the tensor rank.

4.2.1 Higher Order Singular Value Decomposition (HOSVD). We can now make use of the fact that for a given tensor $\mathcal{X}$, we can easily find an exact Tucker decomposition of rank $(R_1, R_2, \ldots, R_N)$ where $R_n = \text{rank}_n(\mathcal{X})$. This gives rise to the *higher order singular value decomposition* (HOSVD). The key idea behind the HOSVD is to find the components that best capture the variation in mode n, while not considering the other modes at this point in time [38]. This directly corresponds to the basic PCA concept and can be formalized as seen in Algorithm 2.

Algorithm 2 HOSVD

```
procedure HOSVD($\mathcal{X}, R_1, \ldots, R_N$)
    for n = 1,…,N do
        $\boldsymbol{A}^{(n)} \leftarrow R_n$ leading left singular vectors of $\boldsymbol{X}_{(n)}$
    end for
    $\mathcal{G} \leftarrow \mathcal{X} \times_1 \boldsymbol{A}^{(1)T} \times_2 \cdots \times_N \boldsymbol{A}^{(N)T}$
    return $\mathcal{G}, \boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}$
end procedure
```

4.2.2 Higher Order Orthogonal Iteration (HOOI). An alternative approach to computing the Tucker decomposition is provided by the *higher order orthogonal iteration* (HOOI). The HOOI is essentially an ALS algorithm, which uses the outcome of performing HOSVD on a tensor as a starting point for initializing the factor matrices [38]. This algorithm, which is outlined in Algorithm 3, is especially advised in cases where we only have access to a truncated HOSVD, since the successive application of the ALS algorithm allows for more accurate decompositions.

Algorithm 3 HOOI

```
procedure HOOI($\mathcal{X}$, $R_1, \ldots, R_N$)
    initialize $\boldsymbol{A}^{(n)} \in R^{I_n\times R}$ for $n = 1, \ldots, N$ using HOSVD
    repeat
        for n = 1,…,N do
            $\mathcal{Y} \leftarrow \mathcal{X} \times_1 \boldsymbol{A}^{(1)T} \times_2 \cdots \times_{n-1} \boldsymbol{A}^{(n-1)T} \times_{n+1} \boldsymbol{A}^{(n+1)T} \times_{n+2}$
            $\hookrightarrow \cdots \times_N \boldsymbol{A}^{(N)T}$
            $\boldsymbol{A}^{(n)} \leftarrow R_n$ leading left singular vectors of $\boldsymbol{Y}_{(n)}$
        end for
    until stopping criterion satisfied
    $\mathcal{G} \leftarrow \mathcal{X} \times_1 \boldsymbol{A}^{(1)T} \times_2 \cdots \times_N \boldsymbol{A}^{(N)T}$
    return $\mathcal{G}$, $\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}$
end procedure
```

4.2.3 Non-Uniqueness. In contrast to the CPD and the (matrix) SVD, the Tucker decomposition is generally not unique. This intuitively follows from the fact that the core tensor $\mathcal{G}$ can be arbitrarily structured and might allow interactions between any component. Imposing additional constraints on the structure of $\mathcal{G}$ can therefore lead to more relaxed uniqueness properties. For instance, the CPD can be expressed in the Tucker model through a superdiagonal core tensor. The HOSVD generates an all-orthogonal core and hence relies on yet another type of special core structure.

5 TENSOR APPLICATIONS IN MACHINE LEARNING

We will now briefly discuss how tensor decompositions can be used in various machine learning models and mention some example applications.

5.1 Temporal Data

Whenever some kind of relationship can be represented as a matrix (e.g. user preferences in a recommender system, adjacency matrix of a graph), tensors provide a straightforward way to model the *temporal component*. Similarly to SVD and non-negative matrix factorization (NMF) in the case of a matrix, performing decomposition on the resulting tensor allows to detect latent structure in the data. Typical tasks in temporal tensor analysis include discovering patterns [40], predicting evolution [9] and spotting anomalies [31].

As another example, the classic problem of community detection can be extended to finding so-called *temporal communities*, that come into existence and subsequently disappear as time progresses [4]. Tensor methods can even be applied when new data arrives in a never-ending continuous stream, without having to deal with infinite time dimension [37].

One important remark we would like to make is the fact that temporal data add jet another structural constraint to the tensor, restricting arbitrary permutations in terms of the tensor's dimensions. This results from the fact that the temporal interpretation adds an additional relationship between the data points stored in the tensor.

5.2 Multi-relational Data

Another domain where tensors arise naturally is representation of *multirelational data*. As an example, one can think of social networks, where users communicate by exchanging messages, tagging pictures and following each other. Each interaction here can be stored as a (subject, relation, object) triplet. Such a multimodal structure is generalized by the notion of multilayer network [20]. Again, tensors lend themselves to a concise description of such data, with each slice representing one of the underlying relations.

Applying tensor factorization algorithms in this setting allows to determine interdependencies occurring on multiple levels simultaneously. For instance, it enables to solve challenging tasks in *statistical relational learning*, such as collective classification [26], word representation learning [17], community detection [32], and coherent subgraph learning [6].

A prominent area of research concerned with multirelational data is analysis of *knowledge networks*. Knowledge graphs, such as Google Knowledge Graph, YAGO or Microsoft Academic Graph [8], are a special case of multilayer networks that are used to store facts about relationships between real-world entities. Main challenge in analyzing such graphs is inferring new relations between objects given the existing data. Here, tensor decomposition approaches provide state-of-the art performance in terms of both quality and computational efficiency [27, 29]. This information can then be used in applications such as question answering and entity resolution [36].

5.3 Latent Variable Modeling

One more area of machine learning where tensor decomposition methods have been gaining significant traction over the last decade is *inference in latent variable models*. Tensor methods have been successfully applied to hidden Markov models [14], independent component analysis [5] and topic models [3].

The standard problem setting is as follows: it is assumed that we are given a (generative) probabilistic model that describes how a set of hidden variables gives rise to observed data. The inference problem now lies in determining the most likely setting of the hidden variables given the observations. Classic algorithms such as maximum likelihood estimation are asymptotically consistent, but usually do not perform well in very high dimensions.

Tensor decomposition methods propose a different approach based on the so-called method of moments [33]. The main idea lies in computing the empirical moments of the data (such as mean, variance, and skewness), and then finding the configuration of latent variables that would give rise to similar quantities under the given model. It has been shown in recent work that in many popular probabilistic models the low-order moment tensors exhibit a specific structure [1]. This fact, combined with recent advances in multilinear algebra enables us to construct effective and efficient algorithms for solving such problems. These methods are known to scale well to larger problems and in general do not suffer much from the curse of dimensionality [28].

We will provide more details on the method of moments, and show how it can be used to perform inference in Gaussian mixture model and topic model by performing decomposition of the corresponding moment tensors.

6 CASE STUDY: ESTIMATION OF MIXTURE MODELS

In order to give the reader a tangible example of how tensor decompositions can be applied to a concrete machine learning problem, we will now take a more detailed look at how we can estimate the parameters for latent variable models by using tensor decompositions. While the basic concepts we present here will work for multiple different latent variable models, we will motivate the derivation with respect to two popular models in machine learning: a Gaussian mixture model and a topic model. Most of the following content is heavily based on Anandkumar et al. [1], which also has additional insights on the computational efficiency and numerical aspects of the presented estimation procedure.

The goal of unsupervised learning approaches is to discover hidden structure (latent variables) in data where no labels are present during training. Typically, there are two main challenges in such settings.

Conditions for identifiability. As one of the basic statistical questions we face in unsupervised learning problems, we have to determine whether a proposed model contains all relevant information for parameter derivation.

Efficient learning of latent variable models. While maximum likelihood estimation (MLE) possesses nice properties such as asymptotic optimality and consistency, it is NP-hard in its derivation for a variety of different problem settings. In practice, iterative algorithms like expectation maximization (EM) are often used. Such algorithms usually suffer from slow convergence and local optima as they don't come with any consistency guarantees [34]. In contrast, the presented way to learn GMMs and topic models is efficient, both with respect to computational complexity and statistical hardness barriers regarding the number of data points needed for learning.

In the following, we will first introduce basic notation and concepts of two different mixture models, namely a Gaussian mixture model and a topic model. Next, we will explain the method of moments, which we will use to construct data moments up to order 3. Finally, we will learn how to derive the wanted parameters from the third order moment by first whitening it and by consequently finding an eigendecomposition for this tensor through the tensor power method.

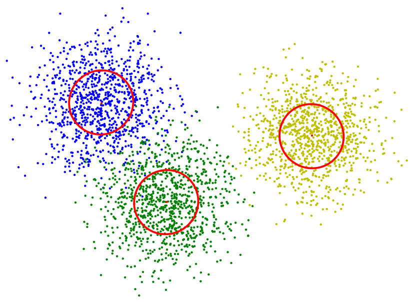

Figure 9: Spherical Gaussian mixture example

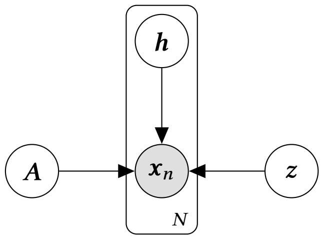

Figure 10: Graphical model for Gaussian mixture model

6.1 Gaussian Mixture Model

We will now briefly revise the concepts of Gaussian mixture modeling, which will be crucial for the upcoming paragraphs. A Gaussian mixture model is a probabilistic model which assumes that data points are generated from a mixture of k different Gaussian distributions/clusters with unknown parameters. We will denote the hidden variable representing a specific Gaussian component by $\boldsymbol{h}$, which is a categorical variable and can take k distinct states. With respect to the representation of $\boldsymbol{h}$, we will use the basis vector form, which means that $\boldsymbol{h} \in \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_k\}$ where $\boldsymbol{e}_i$ is the basis vector along the i-th direction. These vectors can therefore analogously be interpreted as 1-hot-encoded vectors. The expectation of $\boldsymbol{h}$ is defined as the probability of $\boldsymbol{h}$ taking one of these different states and is encoded on the vector $\boldsymbol{w}$: $\mathbb{E}[\boldsymbol{h}] = \boldsymbol{w}$. The means of the Gaussian mixture components are stored in the matrix $\boldsymbol{A} = [\boldsymbol{a}_1 \cdots \boldsymbol{a}_k]$ as columns where $\boldsymbol{a}_i$ is the mean of the i-th component. Hence, $\boldsymbol{A} \in \mathbb{R}^{d \times k}$ where d corresponds to the dimensionality of the data. Each sample is generated as follows:

$$\boldsymbol{x}_n = \boldsymbol{A}\boldsymbol{h} + \boldsymbol{z} \quad \text{with} \quad \boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I}) \tag{36}$$

Note that we constrain the covariance of all Gaussian components to be spherical and identical for all components, as the covariance matrix is only governed by a single variance term σ^2 across all dimensions. While the generalization to differing but still spherical covariances is relatively easy to achieve, the generalization to arbitrary covariance matrices is more challenging and therefore beyond the scope of this paper [11]. Figure 9 shows samples drawn from a GMM with $k = 3$ components and $d = 2$ dimensions, while a graphical model is given in Figure 10.

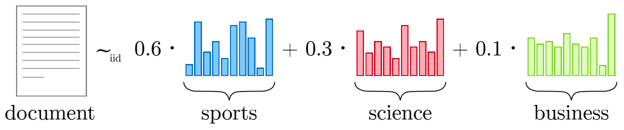

Figure 11: General topic model distribution example

6.2 Topic Model

Another popular latent variable model is the so-called topic model, where we try to explain words in a document through hidden topics (see Figure 11) in order to classify documents or to summarize them. A document consists of a mixture of k different topics, again denoted by the hidden variable $\boldsymbol{h}$, from which $l \geq 3$ words are drawn from an exchangeable bag of words model with vocabulary dimension d to form the text in the document. Every word therefore has the same probability of occurring in the text, given a specific topic. These probabilities are stored in a topic-word matrix $\boldsymbol{A} \in \mathbb{R}^{d \times k}$ with

$$a_{ji} = P_{ji} = P(x_j | h_i) \tag{37}$$

where the word x_j is generated from topic h_i with probability P_{ji}.

The most general treatment of the topic model setting is given by the latent Dirichlet allocation (LDA), where the probability distribution over the latent variable $\boldsymbol{h}$ is given by the Dirichlet distribution. Just as with the GMM, we will only take a look at a simplified model, where each document only contains one single topic, which is chosen with probability w_i with $i \in [k]$. The latent variable $\boldsymbol{h} \in \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_k\}$ is therefore also again a categorical/1-hot-encoded variable and is therefore interpreted as the sole topic given a document. A graphical model is given in Figure 12.

Since the vocabularies of different topics are discrete, the probability distributions over these vocabularies are discrete, too, in contrast to the GMM case. Therefore, the word distribution, given a certain topic, can also be thought of as a distribution over 1-hot-encoded vectors, which are of the size of the vocabulary. Our general linear model is consequently given as follows:

$$\mathbb{E}_{\boldsymbol{x}_j \sim \boldsymbol{h}}[\boldsymbol{x}_j | \boldsymbol{h}] = \boldsymbol{A}\boldsymbol{h} \tag{38}$$

Based on the topic-word matrix, the usage of 1-hot-encoded vectors for the words, and the fact that we assume $\boldsymbol{h}$ is fixed to a single topic i per document, we can interpret the means as probability vectors:

$$P(\boldsymbol{x}_j | \boldsymbol{t}_i) = \mathbb{E}_{\boldsymbol{x}_j \sim \boldsymbol{t}_i}[\boldsymbol{x}_j | \boldsymbol{t}_i] = \boldsymbol{a}_i \tag{39}$$

Note that we write $\boldsymbol{t}_i$ (meaning "topic i") as an equivalent to $\boldsymbol{h} = \boldsymbol{e}_i \in \mathbb{R}^k$. Moreover, since we assume an exchangeable model where the words are sampled independently given a certain topic, the probability of a specific word conditioned on $\boldsymbol{h}$ is the same for all words.

$$P(\boldsymbol{x} | \boldsymbol{t}_i) = P(\boldsymbol{x}_j | \boldsymbol{t}_i) \quad \forall j \in [d] \tag{40}$$

This makes our subsequent moment calculations in the upcoming subsections a lot easier.

6.3 Algorithm Overview

Now that we are familiar with the basics of both models, we will give a brief overview of the parameter estimation procedure for both models.

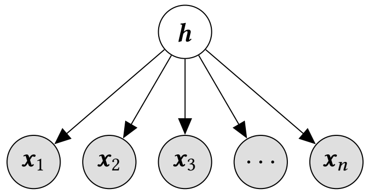

Figure 12: Graphical model for single topic model

(1) **Calculate moments**: Based on the method of moments, we can find a moment formulation of the model in which the latent factors (the $\boldsymbol{a}_i$s and the weighting factors w_i) are producing these moments exclusively.
(2) **Whiten the data**: Once we have obtained the second and the third moment of the underlying data, we are still not able to extract the latent information from these two moments directly. Since the $\boldsymbol{a}_i$s producing these moments might not be orthogonal to each other, it is difficult to recover them uniquely for the moments. Therefore, we propose to orthogonalize (or whiten) the third moment to recover the $\boldsymbol{a}_i$s more easily.
(3) **Decompose the whitened moment tensor**: By using one of the available tensor decomposition methods, the tensor power method, we can then extract the latent factors $\boldsymbol{v}_i$ present in the whitened moment tensor.
(4) **Un-whiten the $\boldsymbol{v}_i$s**: As the resulting latent factors live in the whitened space, we have to transform them back to the original space by applying the inversion of the whitening transformaton.

6.4 Method of Moments (MoM)

An alternative approach to classical MLE, which is often used for parameter estimation, is the previously discussed method of moments. The *method of moments* tries to extract model information by building up higher order moments of the underlying probability distribution and therefore tries to infer model information based on averages of the data. Although we will not prove this fact, we claim that deriving first, second, and third order moment suffices to learn both the spherical GMM and the topic model. Most importantly, it is possible to reduce both (and even more) models to the following moment form:

$$\boxed{\boldsymbol{M}_2 = \sum_{i\in[k]} w_i \boldsymbol{a}_i \circledcirc \boldsymbol{a}_i \qquad \mathcal{M}_3 = \sum_{i\in[k]} w_i \boldsymbol{a}_i \circledcirc \boldsymbol{a}_i \circledcirc \boldsymbol{a}_i} \tag{41}$$

Recall that in the GMM case, the $\boldsymbol{a}_i$s correspond to the cluster mean vectors, whereas they represent word probability-vectors (given a certain topic) in the topic model case.

6.4.1 GMM. In the GMM setting, the first order moment is the mean of all Gaussian components which can be computed as follows:

$$\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{x}] = \boldsymbol{A}\boldsymbol{w} = \sum_{i\in[k]} w_i \boldsymbol{a}_i \tag{42}$$

The second moment corresponds to the covariance matrix of the mixture distribution and can be expressed as follows:

$$\begin{aligned} \Sigma = \mathbb{E}[\boldsymbol{x} \circledcirc \boldsymbol{x}] = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T] &= \mathbb{E}[(\boldsymbol{A}\boldsymbol{h} + z)(\boldsymbol{A}\boldsymbol{h} + z)^T] \\ &= \mathbb{E}[(\boldsymbol{A}\boldsymbol{h})(\boldsymbol{A}\boldsymbol{h})^T] + \mathbb{E}[zz^T] \\ &= \sum_{i\in[k]} w_i(\boldsymbol{a}_i)(\boldsymbol{a}_i)^T + \sigma^2 \boldsymbol{I} \\ &= \sum_{i\in[k]} w_i \boldsymbol{a}_i \circledcirc \boldsymbol{a}_i + \sigma^2 \boldsymbol{I} \end{aligned} \tag{43}$$

By applying $(k-1)$ principal component analysis (PCA)[5] on the covariance matrix of this model, we can obtain span($\boldsymbol{A}$). Also, we can already extract the common variance term σ^2 which corresponds to the smallest eigenvalue of the covariance matrix Σ. If the covariances are governed by different variance terms, then the smallest eigenvalue will yield the average of all of these variances: $\overline{\sigma}^2 = \sum_{i=1}^k w_i \sigma_i^2$. Given these results, we can then apply a technique called spectral clustering, where we would project the samples onto span($\boldsymbol{A}$) and try to classify the points there by using a distance-based clustering approach (e.g k-means). This method, however, requires that the clusters satisfy a sufficient separation criterion. In other words: the separation between the cluster means needs to be sufficiently large to cope with the variance in the data. Otherwise, too many points will be misclassified and hurt the model performance badly.

The third order moment, also called skewness[6], can be expressed in the following way:

$$\begin{aligned} \mathcal{S} = \mathbb{E}[\boldsymbol{x} \circledcirc \boldsymbol{x} \circledcirc \boldsymbol{x}] &= \mathbb{E}[(\boldsymbol{A}\boldsymbol{h}) \circledcirc (\boldsymbol{A}\boldsymbol{h}) \circledcirc (\boldsymbol{A}\boldsymbol{h})] \\ &\quad + \mathbb{E}[(\boldsymbol{A}\boldsymbol{h}) \circledcirc (z) \circledcirc (z)] \\ &\quad + \mathbb{E}[(z) \circledcirc (\boldsymbol{A}\boldsymbol{h}) \circledcirc (z)] \\ &\quad + \mathbb{E}[(z) \circledcirc (z) \circledcirc (\boldsymbol{A}\boldsymbol{h})] \\ &= \sum_{i\in[k]} w_i \boldsymbol{a}_i \circledcirc \boldsymbol{a}_i \circledcirc \boldsymbol{a}_i + \sigma^2 \sum_{i\in[d]} (\boldsymbol{\mu} \circledcirc \boldsymbol{e}_i \circledcirc \\ &\quad \circledcirc \boldsymbol{e}_i + \boldsymbol{e}_i \circledcirc \boldsymbol{\mu} \circledcirc \boldsymbol{e}_i + \boldsymbol{e}_i \circledcirc \boldsymbol{e}_i \circledcirc \boldsymbol{\mu}) \end{aligned} \tag{44}$$

Note that uneven-order moments of the underlying Gaussian components are 0 and are therefore missing from these equation (uneven occurrence of $\boldsymbol{z}$ or $\boldsymbol{e}_i$ respectively).

At this point, both the shared variance σ^2 and the overall mean vector $\boldsymbol{\mu}$ are already known and we can simplify the equations for the second and third moment to bilinear and trilinear equations in the mean vectors, as expressed by Equation (41).

6.4.2 Topic model. In the (single) topic model setting, we think of the k-th moment as the joint distribution of k words in a document.

The first moment of the topic model is given by the probability distribution over a single word, which corresponds to the average of the topic probability vectors.

$$P(x_1) = \mathbb{E}_{\boldsymbol{t}_i \sim \boldsymbol{h}}[\mathbb{E}_{\boldsymbol{x}_1 \sim \boldsymbol{t}_i}[\boldsymbol{x}_1 | \boldsymbol{t}_i]] = \mathbb{E}_{\boldsymbol{t}_i \sim \boldsymbol{h}}[\boldsymbol{a}_i] = \sum_{i\in[k]} w_i \boldsymbol{a}_i \tag{45}$$

[5]Applying PCA with the rank of the projection dimension being $k-1$, i.e. $\min_{P\in\mathbb{R}^{d\times d}} \frac{1}{n} \sum_{i\in[n]} ||x_i - Px_i||^2$ with rank$(P) = k-1$.
[6]asymmetry of the probability distribution about its mean

The joint distribution of two different words in a document can then be expressed as the weighted sum of the means. Here we see that the exchangeability assumption makes our calculations a lot easier, since the noise is independent conditioned on the topic.

$$\begin{aligned}P(x_1, x_2) &= \mathbb{E}_{\boldsymbol{t}_i \sim \boldsymbol{h}}[\mathbb{E}_{\boldsymbol{x}_1, \boldsymbol{x}_2 \sim \boldsymbol{t}_i}[\boldsymbol{x}_1, \boldsymbol{x}_2 | \boldsymbol{t}_i]] \\ &= \mathbb{E}_{\boldsymbol{t}_i \sim \boldsymbol{h}}[\mathbb{E}_{\boldsymbol{x}_1 \sim \boldsymbol{t}_i}[\boldsymbol{x}_1 | \boldsymbol{t}_i] \circledcirc \mathbb{E}_{\boldsymbol{x}_2 \sim \boldsymbol{t}_i}[\boldsymbol{x}_2 | \boldsymbol{t}_i]] \\ &= \mathbb{E}_{\boldsymbol{t}_i \sim \boldsymbol{h}}[\boldsymbol{a}_i \circledcirc \boldsymbol{a}_i] \\ &= \sum_{i \in [k]} w_i \boldsymbol{a}_i \circledcirc \boldsymbol{a}_i\end{aligned} \tag{46}$$

We can now easily extend this concept to the co-occurrence of three words in a document.

$$\begin{aligned}P(x_1, x_2, x_3) &= \mathbb{E}_{\boldsymbol{t}_i \sim \boldsymbol{h}}[\mathbb{E}_{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3 \sim \boldsymbol{t}_i}[\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3 | \boldsymbol{t}_i]] \\ &= \mathbb{E}_{\boldsymbol{t}_i \sim \boldsymbol{h}}[\mathbb{E}_{\boldsymbol{x}_1 \sim \boldsymbol{t}_i}[\boldsymbol{x}_1 | \boldsymbol{t}_i] \circledcirc \mathbb{E}_{\boldsymbol{x}_2 \sim \boldsymbol{t}_i}[\boldsymbol{x}_2 | \boldsymbol{t}_i] \circledcirc \\ &\qquad \circledcirc \mathbb{E}_{\boldsymbol{x}_3 \sim \boldsymbol{t}_i}[\boldsymbol{x}_3 | \boldsymbol{t}_i]] \\ &= \mathbb{E}_{\boldsymbol{t}_i \sim \boldsymbol{h}}[\boldsymbol{a}_i \circledcirc \boldsymbol{a}_i \circledcirc \boldsymbol{a}_i] \\ &= \sum_{i \in [k]} w_i \boldsymbol{a}_i \circledcirc \boldsymbol{a}_i \circledcirc \boldsymbol{a}_i\end{aligned} \tag{47}$$

The topic model's moments therefore directly correspond to the moments we presented in Equation (41).

After discovering that we can represent the moments from both models through Equation (41), we are interested in extracting the latent information from the moments. The problem we are facing now is that we are not able to acquire the $\boldsymbol{a}_i$s from the moments directly without simplifying assumptions. While we don't know the $\boldsymbol{a}_i$s at this point, we will introduce an orthogonality assumption on the $\boldsymbol{a}_i$s and relax this constraint again as we move on.

Recall from Section 4.1.3: if the $\boldsymbol{a}_i$s were orthogonal to each other, then the $\boldsymbol{a}_i$s would also directly correspond to the eigenvectors of the third-order moment tensor $\mathcal{M}_3$. If we knew one specific $\boldsymbol{a}_i$, for example $\boldsymbol{a}_1$, then hitting $\mathcal{M}_3$ with $\boldsymbol{a}_1$ on two dimensions yields the same eigenvector again, scaled by w_1:

$$\mathcal{M}_3(\boldsymbol{I}, \boldsymbol{a}_1, \boldsymbol{a}_1) = \sum_{i \in [k]} w_i \langle \boldsymbol{a}_i, \boldsymbol{a}_1 \rangle^2 \boldsymbol{a}_i = w_1 \boldsymbol{a}_1 \tag{48}$$

This directly corresponds to the concept of matrix eigenvectors. Recall: $\boldsymbol{M}\boldsymbol{v} = \boldsymbol{M}(\boldsymbol{I}, \boldsymbol{v}) = \lambda \boldsymbol{v}$. Equation (48) therefore allows us to verify whether a certain vector corresponds to an eigenvector of $\mathcal{M}_3$.

As we have discussed before, we can usually not assume that the $\boldsymbol{a}_i$s are orthogonal to each other. It is, however, possible to orthogonalize the third moment $\mathcal{M}_3$, which implicitly also results in an orthogonalization of the $\boldsymbol{a}_i$s. This enables us to use the nice properties of Equation (48) on more general $\boldsymbol{a}_i$s.

6.5 Orthogonalization Through Whitening

Using the second moment $\boldsymbol{M}_2$, we can obtain a *whitening transformation*[7], that orthogonalizes $\mathcal{M}_3$. Formally this is expressed as $\boldsymbol{W}^T \boldsymbol{M}_2 \boldsymbol{W} = \boldsymbol{I}$, where $\boldsymbol{W}$ is called the whitening matrix. By applying $\boldsymbol{W}$ as a multilinear transformation on the third moment, we get

$$\begin{aligned}\mathcal{V} = \mathcal{M}_3(\boldsymbol{W}, \boldsymbol{W}, \boldsymbol{W}) &= \sum_{i \in [k]} w_i (\boldsymbol{W}^T \boldsymbol{a}_i)^{\circledcirc 3} \\ &= \sum_{i \in [k]} w_i \boldsymbol{v}_i \circledcirc \boldsymbol{v}_i \circledcirc \boldsymbol{v}_i\end{aligned} \tag{49}$$

where $(\boldsymbol{W}^T \boldsymbol{a}_i)^{\circledcirc 3} = (\boldsymbol{W}^T \boldsymbol{a}_i) \circledcirc (\boldsymbol{W}^T \boldsymbol{a}_i) \circledcirc (\boldsymbol{W}^T \boldsymbol{a}_i)$.

Since we are operating in a whitened space after the transformation of $\mathcal{M}_3$, we should ensure that we can invert this transformation to recover the $\boldsymbol{A}$ in the original space. We are only able to perform this un-whitening if the $\boldsymbol{a}_i$s are linearly independent. Given the whitening matrix $\boldsymbol{W}$, we can relate the whitened latent factors (columns of $\boldsymbol{V}$) and the original latent factors (columns of $\boldsymbol{A}$) as follows: $\boldsymbol{V} = \boldsymbol{W}^T \boldsymbol{A}$.

Note that this whitening procedure leads to a dimensionality reduction, since $\mathcal{M}_3 \in \mathbb{R}^{d \times d \times d}$ and $\mathcal{V} \in \mathbb{R}^{k \times k \times k}$, which in turn imposes limitations on k and d, namely $k \leq d$. While this is usually not a problem for topic models, since the size of the vocabulary can be assumed to be larger than the number of distinct topics, this imposes severe constraints on the GMM, where the number of components may easily exceed the dimensionality of the data. Hence, the presented method's applicability is limited for GMMs.

We can obtain the whitening matrix $\boldsymbol{W}$ through an eigendecomposition of the second moment $\boldsymbol{M}_2$ as follows:

$$\boxed{\begin{aligned}\boldsymbol{M}_2 = \boldsymbol{U}\text{Diag}(\tilde{\lambda})\boldsymbol{U}^T \;\Rightarrow\; \boldsymbol{W} &= \boldsymbol{U}\text{Diag}(\tilde{\lambda}^{-1/2}) \\ \boldsymbol{V} &= \boldsymbol{W}^T \boldsymbol{A}\text{Diag}(w^{1/2})\end{aligned}} \tag{50}$$

The computation of $\mathcal{V}$ for this specific transformation is therefore given by

$$\begin{aligned}\mathcal{V} = \mathcal{M}_3(\boldsymbol{W}, \boldsymbol{W}, \boldsymbol{W}) &= \sum_{i \in [k]} \frac{1}{\sqrt{w_i}} (\boldsymbol{W}^T \boldsymbol{a}_i \sqrt{w_i})^{\circledcirc 3} \\ &= \sum_{i \in [k]} \lambda_i \boldsymbol{v}_i \circledcirc \boldsymbol{v}_i \circledcirc \boldsymbol{v}_i\end{aligned} \tag{51}$$

After this transformation step, we are now capable of decomposing the tensor $\mathcal{V}$ in order to uncover the latent structure present in the whitened third moment, which we will do through the tensor power method.

6.6 Decomposition Through Tensor Power Method

Recall that we can confirm whether $\boldsymbol{v}_1$ is an eigenvector of $\mathcal{V}$ through the following tensor transformation:

$$\mathcal{V}(\boldsymbol{I}, \boldsymbol{v}_1, \boldsymbol{v}_1) = \sum_{i \in [k]} \lambda_i \langle \boldsymbol{v}_i, \boldsymbol{v}_1 \rangle^2 \boldsymbol{v}_i = \lambda_1 \boldsymbol{v}_1 \tag{52}$$

What is still left is to find the k dominant eigenvectors of the orthogonalized tensor $\mathcal{V}$. We can achieve this by applying the tensor power method introduced in 4.1.3. Note that, in order to correspond with the rest of the notation and variables introduced in this chapter, the deflation counter is referred to as i and the power iteration counter as j. After randomly initializing a vector $\boldsymbol{v}_{i,j}$ we can feed the vector into the tensor power step and repeat this step until

[7]linear transformation that transforms a set of random variables with a known covariance matrix into a set of new variables whose covariance is the identity matrix (each variable has variance 1)

Algorithm 4 Tensorized mixture model learning

```
procedure TENSOR_MIXTURE_EST(d, k)
    $X \leftarrow$ read data / generate sample data using d, k
    Compute 1st data moment (mean): $\boldsymbol{\mu} \leftarrow \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i$
    Compute 2nd data moment (covariance): $\Sigma \leftarrow \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^T$
    Decompose 2nd data moment: $\boldsymbol{U}, \boldsymbol{S} \leftarrow \mathrm{SVD}(\Sigma)$
    GMM only: Extract shared variance: $\sigma^2_{\mathrm{est}} \leftarrow \min(\boldsymbol{S})$
    Compute whitening matrix $\boldsymbol{W}$ using Eq. (50)
    Whiten the data: $\boldsymbol{X}_{whit} = \boldsymbol{X} \times \boldsymbol{W}$
    for i = 1,...,k do
        Generate random $\boldsymbol{v}_{\mathrm{old}} \in \mathbb{R}^k$
        Normalize random vector: $\boldsymbol{v}_{\mathrm{old}} \leftarrow \frac{\boldsymbol{v}_{\mathrm{old}}}{||\boldsymbol{v}_{\mathrm{old}}||}$
        repeat
            Compute multilin. transf.: $\boldsymbol{v}_{\mathrm{new}} \leftarrow \frac{\boldsymbol{X}^T_{whit} \times (\boldsymbol{X}_{whit} \times \boldsymbol{v}_{\mathrm{old}})^2}{n}$
            if $i > 1$ then
                for $l = 1, \ldots, i-1$ do
                    Deflate tensor using Eq. (54)
                end for
            end if
            $\boldsymbol{l} = ||\boldsymbol{v}_{\mathrm{new}}||$
            Normalize new vector using Eq. (53)
            if convergence criterion satisfied then
                Add $\boldsymbol{v}_{\mathrm{new}}$ as column $i$ to $\boldsymbol{V}_{\mathrm{est}}$
                Add $\boldsymbol{l}$ as entry $i$ to $\boldsymbol{\lambda}$
            end if
            $\boldsymbol{v}_{\mathrm{old}} \leftarrow \boldsymbol{v}_{\mathrm{new}}$
        until maximum number of iterations reached
    end for
    Perform back-transformation using Eq. (55)
    return $\boldsymbol{A}_{\mathrm{est}}$ (, $\sigma^2_{\mathrm{est}}$)
end procedure
```

convergence (which generally only takes about a dozen iterations):

$$\boldsymbol{v}_{i,j+1} = \frac{\mathcal{V}_i(\boldsymbol{I}, \boldsymbol{v}_{i,j}, \boldsymbol{v}_{i,j})}{||\mathcal{V}_i(\boldsymbol{I}, \boldsymbol{v}_{i,j}, \boldsymbol{v}_{i,j})||_2} \tag{53}$$

Afterwards, we deflate the tensor, i.e. remove eigenvalue λ_i with eigenvector $\boldsymbol{v}_i$ that we just extracted from the tensor as follows:

$$\mathcal{V}_{i+1} = \mathcal{V}_i - \lambda_i \boldsymbol{v}_i \circledcirc \boldsymbol{v}_i \circledcirc \boldsymbol{v}_i \tag{54}$$

This process can be repeated until we have extracted k dominant eigenvalue/-vector pairs from the tensor $\mathcal{V}$.

As an alternative, one could also use other algorithms from the CPD family to solve this problem, as all of these methods try to recover a unique $\boldsymbol{A}$. Determining which algorithm is better suited depends on a number of different implementation decisions that we cannot address here in its entirety.

Finally, since we are not directly interested in the $\boldsymbol{v}_i$s but in the $\boldsymbol{a}_i$s, we still need to perform a backwards transformation through un-whitening via the following equation:

$$\boldsymbol{A} = (\boldsymbol{W}^T)^{\dagger} \boldsymbol{V} \mathrm{Diag}(\boldsymbol{\lambda}) \tag{55}$$

Note that $\boldsymbol{M}^{\dagger} = (\boldsymbol{M}^T\boldsymbol{M})^{-1}\boldsymbol{M}^T$ denotes the Moore-Penrose pseudo-inverse of the matrix $\boldsymbol{M}$.

Library	Available for	Source
TensorLy	Python	[16]
N-way Toolbox	Matlab	[7]
pytensor	Python	[41]
scikit-tensor	Python	[25]
SPLATT	C/C++, Octave, Matlab	[19]
rTensor	R	[22]
Tensor Toolbox	Matlab	[21]
Tensorlab	Matlab	[39]

Table 1: Popular tensor libraries

6.7 Algorithm Summary

To sum up the above approach to estimate latent variable models using tensor decomposition, we present the algorithm outline in Algorithm 4.

For the more practically inclined reader we have also created both Matlab and Python scripts for estimating GMMs as part of this case study, which can be accessed here[8]. A similar Matlab code for estimating exchangeable hidden variable models can be found here[9].

7 AVAILABLE SOFTWARE LIBRARIES

While lots of programming languages provide data structures for multi-dimensional arrays either as part of their standard libraries or as part of widely used external packages, we would briefly like to mention a few popular tensor libraries. These libraries usually provide a more optimized way of storing and treating tensors, as well as techniques for efficiently decomposing them using the algorithms we described in Section 4. Since tensors and their decompositions have only started to gain traction in the applied computer science community over the recent years, most libraries are still only available for proprietary environments like Matlab.

An overview of these libraries is given in Table 1.

8 CURRENT RESEARCH

Finally, we would like to point out some of the current research directions at the intersection between tensors and machine learning. At the moment, most research is centered around the following two main questions [10]: *a) How can we formulate other machine learning problems as tensor decompositions?* While some machine learning problems can already be solved very efficiently through tensor decompositions (see Section 5), the effort of determining whether tensor methods can also be beneficial to other machine learning algorithms, like neural networks [15], is still ongoing. *b) How can we compute tensor decompositions under weaker assumptions?* While tensor decompositions usually have weak conditions for uniqueness, the requirements for effectively using them in machine learning settings are quite strong. Recall for example that the GMM estimation in Section 6 requires $k \leq d$, which is a rather strong limitation.

[8]https://github.com/steverab/tensor-gmm

[9]https://bitbucket.org/kazizzad/tensor-power-method

9 CONCLUSIONS

Before closing with this paper, we would like to briefly recap some of the most important take-away messages. First, we have looked at the rotation problem for matrices, which prevented us from trying to find a unique low-rank decomposition for said matrix. Then, we have introduced basic tensor notation and properties and explained that low rank tensors are generally more rigid than low-rank matrices because of the interrelations between different slices along the tensor's dimensions. Afterwards, we introduced two of the most widely used tensor decomposition approaches, the CP decomposition and the Tucker decomposition, where we elaborated on how these decompositions are computed (which is often done through an ALS algorithm) and analyzed under which conditions these decompositions are unique. To build a bridge to the machine learning world, we have discussed how and why tensor decompositions are used in various machine learning sub-disciplines and also gave a detailed example of estimating Gaussian mixture models and simple topic models by using the method of moments and the tensor power method to extract the needed parameters. And lastly, we have given a list of the most prominent tensor libraries and a brief overview of current research questions in the tensor decomposition field.

REFERENCES

[1] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky. 2012. Tensor decompositions for learning latent variable models. *ArXiv e-prints* (Oct. 2012). arXiv:cs.LG/1210.7559

[2] A. Anandkumar, R. Ge, and M. Janzamin. 2014. Analyzing Tensor Power Method Dynamics in Overcomplete Regime. *ArXiv e-prints* (Nov. 2014). arXiv:1411.1488

[3] Animashree Anandkumar, Daniel Hsu, and Sham M Kakade. 2012. A method of moments for mixture models and hidden Markov models. In *Conference on Learning Theory*. 33–1.

[4] Miguel Araujo, Spiros Papadimitriou, Stephan Günnemann, Christos Faloutsos, Prithwish Basu, Ananthram Swami, Evangelos E Papalexakis, and Danai Koutra. 2014. Com2: fast automatic discovery of temporal ('Comet') communities. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 271–283.

[5] Christian F Beckmann and Stephen M Smith. 2005. Tensorial extensions of independent component analysis for multisubject FMRI analysis. *Neuroimage* 25, 1 (2005), 294–311.

[6] Brigitte Boden, Stephan Günnemann, Holger Hoffmann, and Thomas Seidl. 2012. Mining coherent subgraphs in multi-layer graphs with edge labels. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1258–1266.

[7] Rasmus Bro and Claus A. Andersson. 2013. The N-way toolbox for MATLAB. (2013). http://www.models.life.ku.dk/nwaytoolbox

[8] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 601–610.

[9] Daniel M Dunlavy, Tamara G Kolda, and Evrim Acar. 2011. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 5, 2 (2011), 10.

[10] Rong Ge. 2015. Tensor Methods in Machine Learning. (2015). http://www.offconvex.org/2015/12/17/tensor-decompositions/

[11] R. Ge, Q. Huang, and S. M. Kakade. 2015. Learning Mixtures of Gaussians in High Dimensions. *ArXiv e-prints* (March 2015). arXiv:cs.LG/1503.00424

[12] C. Hillar and L.-H. Lim. 2009. Most tensor problems are NP-hard. *ArXiv e-prints* (Nov. 2009). arXiv:cs.CC/0911.1393

[13] F. L. Hitchcock. 1927. The expression of a tensor or a polyadic as a sum of products. *J. Math. Phys* 6, 1 (1927), 164–189.

[14] Daniel Hsu, Sham M Kakade, and Tong Zhang. 2012. A spectral algorithm for learning hidden Markov models. *J. Comput. System Sci.* 78, 5 (2012), 1460–1480.

[15] M. Janzamin, H. Sedghi, and A. Anandkumar. 2015. Beating the Perils of Non-Convexity: Guaranteed Training of Neural Networks using Tensor Methods. *ArXiv e-prints* (June 2015). arXiv:cs.LG/1506.08473

[16] Jean Kossaifi, Yannis Panagakis, and Anima Anandkumar. 2017. TensorLy - Simple and Fast Tensor Learning in Python. (2017). https://github.com/tensorly/tensorly

[17] Rodolphe Jenatton, Nicolas L Roux, Antoine Bordes, and Guillaume R Obozinski. 2012. A latent factor model for highly multi-relational data. In *Advances in Neural Information Processing Systems*. 3167–3175.

[18] Arthur R Jensen. 1985. The nature of the Black–White difference on various psychometric tests: Spearman's hypothesis. *Behavioral and Brain Sciences* 8, 2 (1985), 193–219.

[19] George Karypis. 2015. SPLATT - Parallel Sparse Tensor Decomposition. (2015). http://glaros.dtc.umn.edu/gkhome/splatt/overview

[20] Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P Gleeson, Yamir Moreno, and Mason A Porter. 2014. Multilayer networks. *Journal of complex networks* 2, 3 (2014), 203–271.

[21] Tamara G. Kolda and Brett W. Bader. 2015. MATLAB Tensor Toolbox. (2015). http://www.sandia.gov/~tgkolda/TensorToolbox/index-2.6.html

[22] James Li, Jacob Bien, and Martin Wells. 2015. rTensor: Tools for Tensor Analysis and Decomposition. (2015). https://cran.r-project.org/web/packages/rTensor/index.html

[23] Ankur Moitra. 2014. Algorithmic Aspects of Machine Learning. (2014). http://people.csail.mit.edu/moitra/docs/bookex.pdf

[24] Ankur Moitra. 2014. Tensor Decompositions and their Applications. (2014). http://people.csail.mit.edu/moitra/docs/Tensors.pdf

[25] Maximilian Nickel. 2016. Python library for multilinear algebra and tensor factorizations. (2016). https://github.com/mnick/scikit-tensor

[26] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th international conference on machine learning (ICML-11)*. 809–816.

[27] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2012. Factorizing yago: scalable machine learning for linked data. In *Proceedings of the 21st international conference on World Wide Web*. ACM, 271–280.

[28] Ivan V Oseledets and Eugene E Tyrtyshnikov. 2009. Breaking the curse of dimensionality, or how to use SVD in many dimensions. *SIAM Journal on Scientific Computing* 31, 5 (2009), 3744–3759.

[29] Ankur Padia, Kostantinos Kalpakis, and Tim Finin. 2016. Inferring Relations in Knowledge Graphs with Tensor Decompositions. In *IEEE International Conference on Big Data*. IEEE.

[30] M. Panju. 2011. Iterative Methods for Computing Eigenvalues and Eigenvectors. *ArXiv e-prints* (May 2011). arXiv:1105.1185

[31] Evangelos Papalexakis, Konstantinos Pelechrinis, and Christos Faloutsos. 2014. Spotting Misbehaviors in Location-based Social Networks Using Tensors. In *Proceedings of the 23rd International Conference on World Wide Web (WWW '14 Companion)*. ACM, New York, NY, USA, 551–552. DOI: http://dx.doi.org/10.1145/2567948.2576950

[32] Evangelos E Papalexakis, Leman Akoglu, and Dino Ience. 2013. Do more views of a graph help? Community detection and clustering in multi-graphs. In *Information fusion (FUSION), 2013 16th international conference on*. IEEE, 899–905.

[33] Karl Pearson. 1894. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A* 185 (1894), 71–110.

[34] Richard A Redner and Homer F Walker. 1984. Mixture densities, maximum likelihood and the EM algorithm. *SIAM review* 26, 2 (1984), 195–239.

[35] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos. 2016. Tensor Decomposition for Signal Processing and Machine Learning. *ArXiv e-prints* (July 2016). arXiv:stat.ML/1607.01668

[36] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*. 926–934.

[37] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. 2006. Beyond Streams and Graphs: Dynamic Tensor Analysis. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '06)*. ACM, New York, NY, USA, 374–383. DOI: http://dx.doi.org/10.1145/1150402.1150445

[38] Brett W. Bader Tamara G. Kolda. 2009. Tensor Decompositions and Applications. *SIAM Rev.* 51, 3 (August 2009), 455–500. DOI: http://dx.doi.org/10.1137/07070111X

[39] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer. 2016. Tensorlab: A MATLAB package for tensor computations. (2016). http://www.tensorlab.net

[40] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, and Jaime G Carbonell. 2010. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*. SIAM, 211–222.

[41] Ji Oh Yoo, Arvind Ramanathan, and Christopher J. Langmead. 2011. PyTensor: A Python based Tensor Library. (2011). https://code.google.com/archive/p/pytensor/