Tensor Decomposition

Taher Haitami, Divya Kranthi, Elsie Ye, Anny Zhao

Contents

1 Abstract

Tensors can describe hefty datasets that matrices cannot. These tensors can be decomposed into simpler components to allow for extensive data analysis. Thus, tensor decomposition has several applications in signal processing, statistics, numerical linear algebra, numerical analysis, data mining, graph analysis, and machine learning. In this report, we use the Olivetti Faces dataset in scikit-learn to explore two different tensor decompositions: CP (CANDECOMP/PARAFAC) and Tucker/Higher Order Singular Value decomposition (HOSVD). By applying these algorithms to the dataset, this report provides an overview of their practical applications, limitations, and potential future developments. Overall, we seek to explain the basics of tensor operations with comparisons to matrix algebra, prove existence and uniqueness for the aforementioned tensor decompositions, and finally provide an accessible application of analogs of matrix decomposition to a dataset represented by tensors. We also identify the strengths and limitations (of CP and Tucker decompositions) in a compression task that will illustrate the fundamental difference between tensor decomposition and SVD.

2 Introduction

SVD, EVD, PCA are some of the disguises used by Linear Algebra to shield engineers, economists, and undergraduates from the reality that shapes can be made of numbers. This, of course, is perfectly acceptable given modern computational capability, which Gregorio Ricci-Curbastro anticipated in 1900 when he asked: What if shapes were made of shapes were made of numbers? And thus the first tensor was born. Tensors are, in general, much better models for natural phenomena than matrices for the simple reason that they can describe multilinearities that matrices can't. Tensors can describe datasets that matrices cannot encompass, like a year's worth of stock market data or a collection of frames in a movie. Despite their more accurate description of large-scale phenomena, tensors are relatively rare compared to matrices. This is because, while they are very useful, problems involving tensors tend to be NP-Complete, a dealbreaker for most in the age of computation. Fortunately, advances in machine learning and endless data collection has made them an indispensable tool in applied math.

2.1 What is a Tensor?

A tensor is the n-dimensional generalization of a matrix. It is more accurate to say that a matrix is the special 2D case of a tensor (2-tensor). A matrix is considered 2-dimensional because all of its entries can be indexed with two coordinates (e.g. the a 2x2 Identity matrix has entry (1,1) = 1 and (1,2) = 0). It is therefore easy to see that a 1-Dimensional tensor, whose entries can be indexed with one value, is simply a vector. A third order tensor (3-tensor), therefore, will have three indexing value (x, y, z), which is consistent with the number of dimension of the tensor.

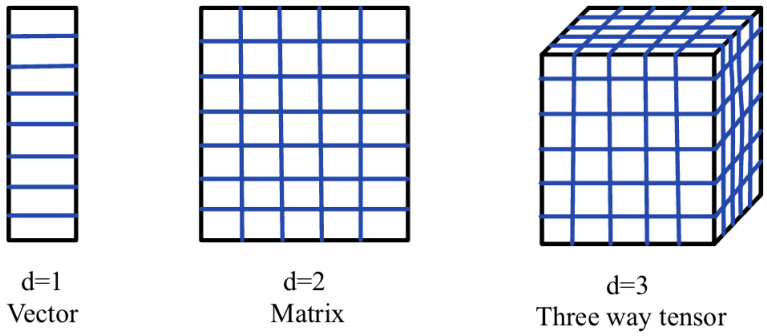


Figure 1: *A first order tensor, second order tensor, and third order tensor.*

The index of a matrix is in the form (row, column), can be shifted into form (column, row) while still maintaining its properties such as rank by transposing. A 3-tensor has many more arrangements than a matrix given that there are many more possible combinations of its index coordinates ((x, y, z), (y, x, z), ...) , allowing rearrangements like those shown below.

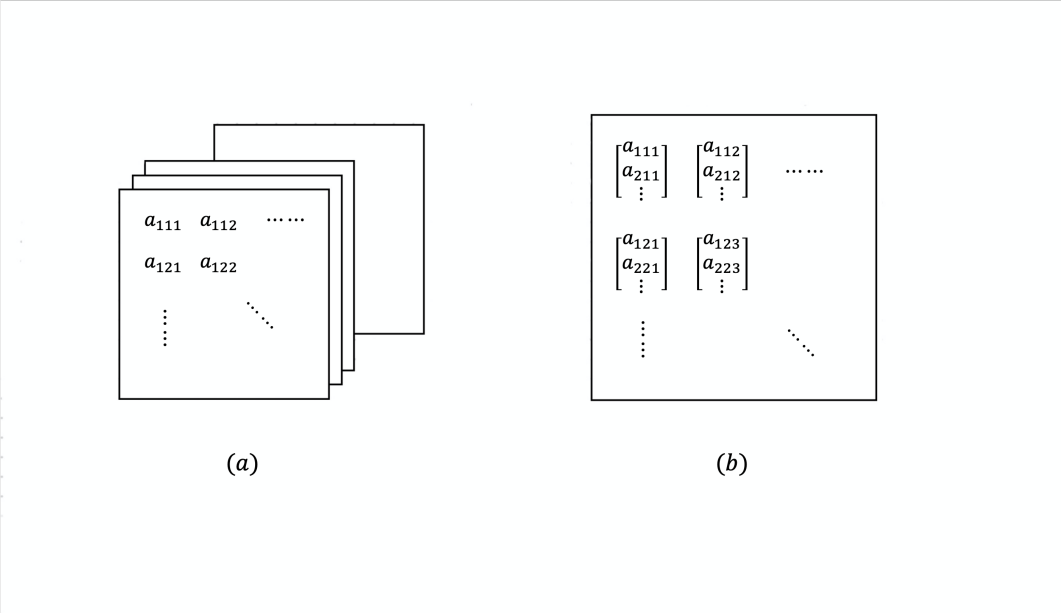


Figure 2: *Different visualizations for a 3D tensor.*

The key concept is that a tensor usually refers to a higher dimension version of a matrix, which can be thought of as a matrix where entries are also matrices. This simplification might lead one to believe that one can treat a higher order tensor like one would a matrix when carrying out linear regressions. However, the added dimensionality significantly complicates the organization of the tensor, and we find that many statements on optimality of a solution for regression on matrices (Eckart-Young Theorem, SVD) do not hold for tensor of order higher than 2. To appreciate the difficulty of working with higher dimensional tensors as opposed to matrices, we must first define basic tensor operations as traditional matrix operations become unwieldy when applied to tensors.

2.2 Matrix Products

2.2.1 Kronecker Product

The Kronecker product of two matrices A$\in\mathbb{R}^{IxK}$ and B$\in\mathbb{R}^{JxL}$ is defined as follows:

$$A \otimes B = \begin{bmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{ik} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & \cdots & b_{1l} \\ \vdots & \ddots & \vdots \\ b_{j1} & \cdots & b_{jl} \end{bmatrix} = \begin{bmatrix} a_{11}B & \cdots & a_{1k}B \\ \vdots & \ddots & \vdots \\ a_{i1}B & \cdots & a_{ik}B \end{bmatrix}$$

Note: Unlike matrix multiplication, the matrices A and B **do not** need to have the same number of rows or columns. For instance, A can be a 3x3 matrix and B can be a 1x2 vector, and A⊗B exists.

Example:

$$A = \begin{bmatrix} 1 & 2 \\ 1 & 5 \end{bmatrix}, B = \begin{bmatrix} 3 & 7 \\ 1 & 4 \end{bmatrix} \Rightarrow A \otimes B = \begin{bmatrix} 1*B & 2*B \\ 1*B & 5*B \end{bmatrix} = \begin{bmatrix} 3 & 7 & 6 & 14 \\ 1 & 4 & 2 & 8 \\ 3 & 7 & 15 & 35 \\ 1 & 4 & 5 & 20 \end{bmatrix}$$

2.2.2 Khatri-Rao Product

The Khatri-Rao product of two matrices A$\in\mathbb{R}^{IxK}$ and B$\in\mathbb{R}^{JxK}$ is defined as follows:

$$A \odot B = \begin{bmatrix} A_1 \otimes B_1 & \cdots & A_k \otimes B_k \end{bmatrix}$$

Here, A_i and B_i are columns of A and B, respectively. Therefore, the Khatri-Rao product is a column-wise Kronecker product. Thus, matrices A and B must have an equal number of columns for their Khatri-Rao product to exist.

Example:

$$A = \begin{bmatrix} 1 & 2 \\ 1 & 5 \end{bmatrix}, B = \begin{bmatrix} 3 & 7 \\ 1 & 4 \end{bmatrix} \Rightarrow A \odot B = \begin{bmatrix} 3 & 14 \\ 1 & 8 \\ 3 & 35 \\ 1 & 20 \end{bmatrix}$$

3 Tensor Rank and Decomposition Theory

3.1 Defining rank for Tensors

Now that we have the tools to describe and manipulate tensors, we can attempt to understand higher dimensional tensors with analogies to well known matrix properties. The first and simplest is rank. The rank of a matrix is simple but provides information on its invertibility, the nature of its linear transformation, and its null space. However, if instead

of a 3x3 matrix, we had a 3x3x3 tensor(Figure 1: three-way tensor), what is its rank? A simple guess is 3 since it is 3x3x3; but this doesn't really give us any additional information like the matrix rank does. Given the appearance of the tensor (matrices stacked along the z axis, Figure 2.a) it would be easy to determine the rank of every individual matrix along the z axis and sum them, but what if another tensor were indexed the exact same way with the same entries, except in the form Figure 2.b? If two of the matrices in the first tensor share one vector (up to scaling), then the two equivalent matrices would have the different ranks. This issue persists for any rearranging or unfolding of the tensor. It is clear that rank of a tensor cannot depend on its lower dimensional components the same way a matrix rank depends on relationships between its vectors. Instead we will describe tensor rank using tensors, and as follows:

The rank of a n-dimensional tensor T is the minimum number of rank 1 n-tensors that we can sum to get T. A rank 1 n-tensor is a tensor that can be written as the Kronecker product of n vectors.

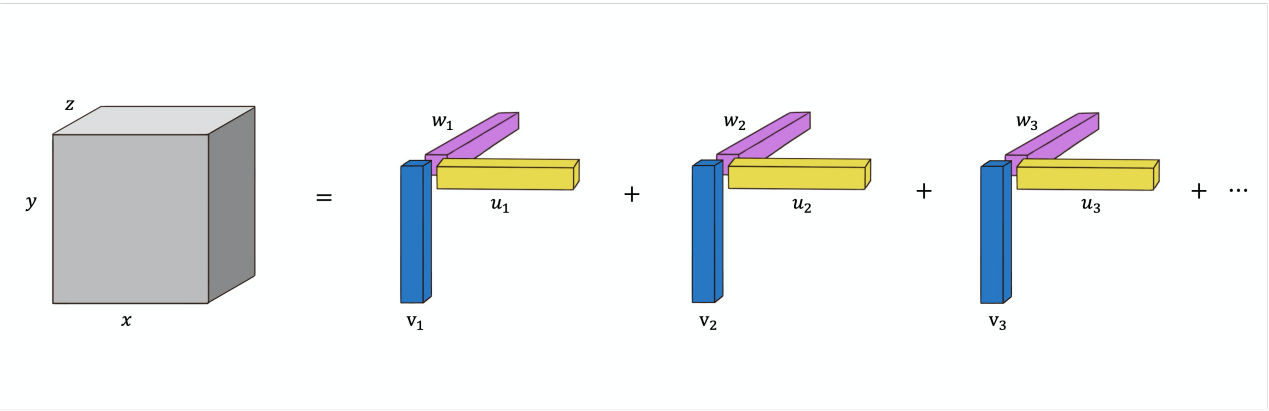


Figure 3: *Rank of a tensor.*

The key to finding a tensor's rank is to find the vectors of matrices V,W, and U (Figure 3). For nearly every tensor, this is a difficult problem [1][2].

Note here that this definition is a generalization of matrix rank, which is can be illustrated by decomposing a matrix using SVD.

We do not need to calculate the SVD of a matrix to find its rank, but it is a trivial calculation. For higher order tensors, we must decompose the tensor into components of the same dimension, which is essentially the same process as the SVD method of finding matrix rank. This begs the question: can we find a decomposition analogous to SVD, but for tensors? No.

Given a tensor T, it is possible to find a decomposition for the tensor (Figure 3 illustrates CP decomposition, which will be described later in the text), but we cannot consider this analogous to SVD simply because SVD gives us much more concrete information about the matrix than the above decomposition gives us about tensors. For now, finding the rank of a tensor is enough to motivate a study of tensor decomposition.

To find the rank of a tensor, a good place to start is CP decomposition. The goal of this decomposition is to find a unique set of vectors V,W,U such that the following holds:

Note that by definition, the CP decomposition of a tensor will also tell us about the rank of the tensor. It is essentially a decomposition of a tensor into its rank 1 component tensors, unique up to permutation and scaling of the vectors in V,W,U.

3.2 Jennrich's proof for existence and uniqueness of Tensor rank [6]

Up to scaling and permutation, tensor T has a unique decomposition into n rank-1 tensors defined by $u_i \otimes w_i \otimes v_i$.

$$T = \sum U \odot W \odot V$$

given that columns of U and V are orthogonal respectively and every pair of columns in W are linearly independent from each other.

We define:

$$T_\alpha = \sum_{i=1}^{n} (u_i \otimes w_i \otimes v_i)a = \sum_{i=1}^{n} \langle a, w_i \rangle u_i \circledcirc v_i$$

and

$$T_\beta = \sum_{i=1}^{n} \langle b, w_i \rangle u_i \circledcirc v_i$$

for a and b as random vectors of length n. This way T_α and T_β are matrices of tensor T scaled by α and β and summed along the z-axis.

Now the question is: how do we represent U, W, V from both matrices?

A simple answer is to take the Eigenvalue Decomposition (EVD) of both. However, an even easier answer is to consider the tensors an EVD where the eigenvalues are scaled by their respective vectors.

So:

$$T_\alpha = U \begin{bmatrix} \lambda_1 a_1 & \cdots & \cdots \\ \vdots & \lambda_2 a_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} V^T$$

$$T_\beta = U \begin{bmatrix} \lambda_1 b_1 & \cdots & \cdots \\ \vdots & \lambda_2 b_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} V^T$$

in this case, $\lambda_n$is the n^{th}column of W^T.

This is solvable (given that we have T_α, T_β, a, b) by setting:

$$(T_\alpha)^T (T_\beta)^{-1} = U \sum_\alpha V^T (U \sum_\beta V^T)^{-1}$$

$$= U \begin{bmatrix} \lambda_1 a_1 & \cdots & \cdots \\ \vdots & \lambda_2 a_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \lambda_1 \frac{1}{b_1} & \cdots & \cdots \\ \vdots & \lambda_2 \frac{1}{b_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} (U_T)^{-1}$$

$$= U \begin{bmatrix} \frac{a_1}{b_1} & \cdots & \cdots \\ \vdots & \frac{a_2}{b_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} (U^T)^{-1}$$

and

$$(T_\alpha)^{-1}(T_\beta)^T = V^{-1} \begin{bmatrix} \frac{1}{a_1\lambda_1} & \cdots & \cdots \\ \vdots & \frac{1}{a_2\lambda_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} (U^{-1})^T (U \sum_\beta V^T)^{-1}$$

$$= V^{-1} \begin{bmatrix} \frac{b_1}{a_1} & \cdots & \cdots \\ \vdots & \frac{b_2}{a_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} V$$

if all values $\frac{a_i}{b_i}$ are unique, then there is a unique solution for U and V.

Clearly, V is the eigenvector matrix of the EVD of $((T_\alpha)^{-1}(T_\beta)^T)^{-1}$ for orthogonal/ symmetric U, $U^T = U$;

So U is the eigenvector matrix of the EVD of $T_\alpha(T_\beta)^{-1}$. With U and V, we can easily find W.

Uniqueness holds because the eigenvalue of the decomposition to find U and V are set by a and b, so it can be made unique if a and b are scalings of each other, or permutations.

The linear systems to solve for W (using either T_α or T_β) have a unique solution because of the condition that every pair of w_i in W be linearly independent, so there are exactly the necessary amount of unknown variables for a unique solution.

3.3 Defining a Tensor decomposition using rank

This proof and uniqueness can be generalized up to any shape of U and V, provided that they both have a higher rank than W (i.e. number of columns of U and V are each greater than the number of columns of W). The difference is that one would use SVD instead of EVD.

In other words, decomposing a tensor to find its rank is to find U,W,V such that the loss defined by $||T - \sum U \odot W \odot V||_F^2$ is minimized.

We know that we can solve this minimization problem for any shape of U,V, so perhaps solving it for U' and V' and of rank lower than tensor T's actual U and V, we can attain an analogue matrix SVD.

By proof, we can indeed find a U' and V' for higher order T, and this represents an estimate that minimizes the norm above(close to 0, oftentimes), but a 1-component decomposition,is not an analogue for a rank 1 approximation of a matrix using its largest singular value.

The simple reason is : the optimal rank n estimate of a matrix is the sum of the rank 1 estimates, corresponding to the eigenvectors with the n highest eigenvalues. The best rank 2 estimate is the sum of the best rank 1 estimate, and the second best rank 1 estimate(by Eckart Young theorem).

For tensors, the optimal rank 2 estimate (as defined by minimizing the Frobenius norm above), will not be the sum of the best rank 1 tensor estimate and the second best rank 1 estimate. In fact, there is no "second best" estimate as the norm above can be brought arbitrarily close to 0 for any number of components [2].

4 Decomposition

4.1 CP Decomposition [1]

CP decomposition is a method used to reduce a tensor into another tensor of lower rank. This is equivalent to finding U', W', V' with an arbitrary number of components using the uniqueness proof in the section above. While Jennrich's proof does provide an algorithm for calculating a CP decomposition, practical application to large tensors can become numerically unstable, so solving the problem below for U',W',V' at the same time is computationally expensive:

$$min||T - \sum U' \odot W' \odot V'||_F^2$$

$$U' = T^{(1)}(V \odot W)(V^T V \cdot W^T W)^\dagger$$

$$W' = T^{(2)}(V \odot U)(V^T V \cdot U^T U)^\dagger$$

$$V' = T^{(3)}(U \odot W)(W^T W \cdot U^T U)^\dagger$$

We instead initialize W and V as constants, and solve for U. Then we set the calculated U and V as constants, and solve for W. Then repeat the same procedure for V, alternating between the matrices until they converge to an arbitrary standard (until the matrices only change by some small amount or until the norm is below a set threshold). This method is known as alternating least squares (ALS).

4.2 Tucker Decomposition [1]

Tucker decomposition involves decomposing a tensor into a core tensor and factor matrices. This core tensor is multiplied by the respective factor matrices along each mode in order to approximate the original tensor. For an N^{th} order tensor χ, the Tucker model is given by:

$$\chi = \chi_1 \times_1 A^{(1)} \cdots \times_N A^{(N)}$$

Here, χ_1 is the core tensor, $A^{(1)} \cdots A^{(N)}$ are the factor matrices, and $\times_i$ implies that χ_1 is being multiplied by $A^{(i)}$ along mode i.

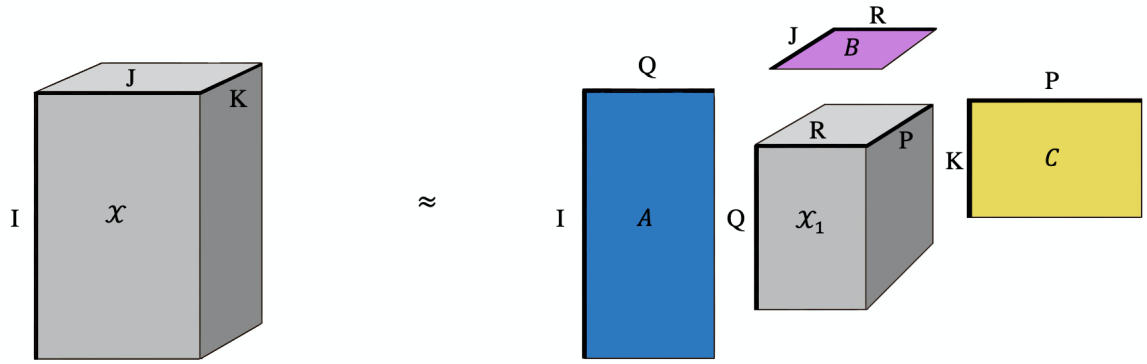


Figure 4: *Compact Tucker decomposition of a third order tensor.*

The Tucker model for a third order tensor is given by:

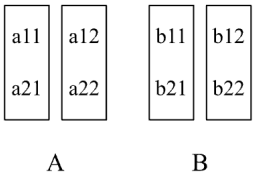
$$\chi = \chi_1 \times_1 A \times_2 B \times_3 C$$

4.2.1 Tensor n-Rank

The n-Rank (Rank_n) of a tensor χ is given by the dimension of the vector space spanned by the columns along mode-n.

Illustration:

A tensor $\chi^{2\times2\times2}$ has the following frontal slices (A and B) and mode-1 columns:



$\text{Rank}_1(\chi)$ is the dimension of the vector space spanned by the columns of A and B outlined in black.
Now, let χ be an N^{th} order tensor. R_i denotes $\text{rank}_i(\chi)$ (for i=1,$\cdots$,N). If the Tucker decomposition has rank $(R_1, \cdots, R_N)$ such that $R_i = \text{rank}_i(\chi)$, then the decomposition is exact. If $R_i \leq \text{rank}_i(\chi)$, then the decomposition is inexact or *truncated*. Generally, higher ranks lead to better approximations. However, a higher rank Tucker decomposition also requires more storage and computation.

4.2.2 Tucker/HOSVD Algorithm

If $\chi^{3\times3\times3}$ is a tensor, its mode-n *unfoldings* are given by $\chi^{(n)}$. For an N^{th} order tensor, the algorithm for a rank-$(R_1, \cdots, R_N)$ Tucker decomposition is:

1. Generate mode-n unfoldings of the tensor χ and store them as $\chi^{(n)}$ for $n = 1, \cdots, N$.
2. For $\chi^{(n)}$, compute the R_n leading left singular vectors and store them as $A^{(n)}$ (these are the factor matrices).
3. Use $A^{(n)}$ to calculate the core tensor G:

$$G = \chi \times_1 A^{(1)T} \cdots \times_N A^{(N)T}$$

4. Return the core tensor G and the factor matrices $(A^{(1)}, \cdots, A^{(N)})$.

5 Application and Results

5.1 Exploration of Data

To gain insights into the effectiveness of tensor decomposition on real-life applications and its future development prospects, we focus on the performance of tensor decomposition in image processing as our investigation direction. We utilized both PC decomposition and Tucker decomposition techniques to reduce and reconstruct some face images, carefully examining and comparing their respective performance. We leveraged a range of open-source software tools, including the machine learning library Scikit-learn (sklearn), the tensorflow library, and the tensorly Python library. We introduced fetch-olivetti-faces from the Sklearn's datasets package as our operational dataset (Figure 5).

```
import sklearn as sk
from sklearn.decomposition import PCA
from sklearn.datasets import fetch_olivetti_faces
import numpy as np
import scipy as sc

import matplotlib as plt

from pylab import *

import tensorflow as tf
import tensorly as tl
```

Figure. 5: *The libraries and packages we are using*

To demonstrate the strengths and weaknesses of tensor decomposition, we decided to apply the two most popular methods (CP and Tucker Decomposition) to a tensor dataset. We downloaded a set of 64x64 images of close-ups of human faces from the "Olivetti Faces" dataset in the Scikit-learn library, extracted one neutral expression per face, and stacked

them along the z-axis. The result is a (64, 64, 40) shape tensor (as in Figure 5) where every slice is the image of a face. This was done to appeal to a wider audience as many complex ideas can be made accessible by analogy to recognizable features. The strength of the decomposition can be described by how easily the reconstruction of the faces can be distinguished in the reconstructed tensor.

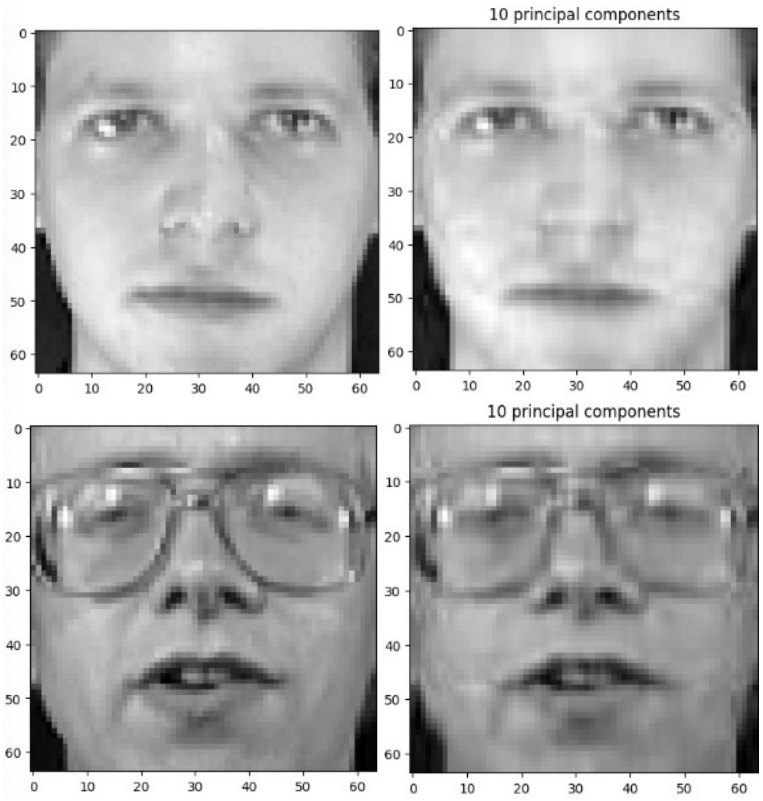


Figure 6: *PCA decomposition on the image. On the left is the orginial image. On the right is the reconstructed image after PCA for 10 principal components*

We explored the data by first applying PCA, to give an idea of how well an individual face can be constructed with less than 1/6 of the components. The faces in Figure 6 are still very recognizable.

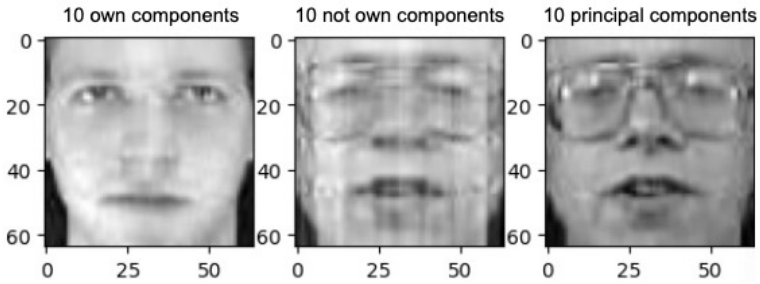


Figure 7: *Using the principal components of one image to reconstruct another image*

We also wanted to illustrate the importance of having the right principal components by attempting to reconstruct one image, with the principal components of another image. This

is essentially a compression task, where the recognizability of the image after compression then decompression reflects the idea that PCA is unique to every image. The reconstructed image, while still recognizable, is significantly worse than when using its own principal components, highlighting that while faces are fairly similar, much of what can be considered distinguishing lies in the differences between the principal components themselves.

5.2 CP Decomposition

For CP decomposition, our implementation worked well for cases of smaller tensors, but failed due to numerical instability for our dataset. This is likely because the the implementation required too many pseudoinverses of large matrices, so it became less accurate after each iteration. A positive in our implementation, however, was that when compared to the TensorLy function we used to explore the data, we were consistently able to get at least one set of reconstruction vectors similar to Tensorly. Moreover, TensorLy's CP decomposition function uses methods in both the initialization of the component matrices and the calculation of the true matrices to curb the effects of numerical error.

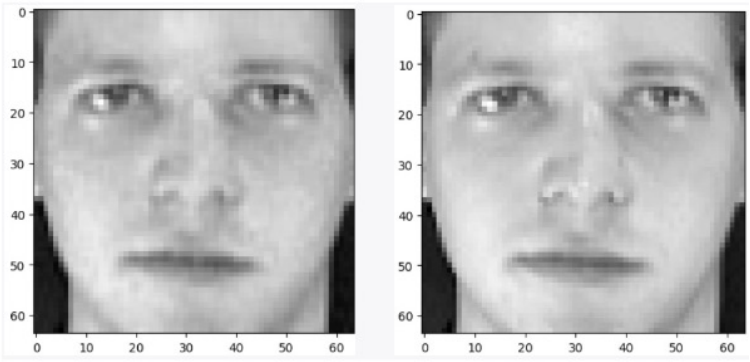


Figure 8: *Comparison between CP decomposition image of rank=400 and original image. On the left is the CP decompositon of rank=400. On the right is the original image.*

Applying CP decomposition to our dataset reveals even more connections between tensor decomposition and PCA, whereby increasing the number of components used provides diminishing returns. Here we might see the value of CP decomposition in a compression task. The image recreated using 100 CPD components is approximately as recognizable as the 10 principal components of the original image. However, the tensor components compress all images, while a similar task for PCA would be to take the first 10 principal components for every image, which would be 400 components total. Moreover, looking at the PCA for every individual image wouldn't provide the same intuition for the entire dataset as Figure 9 does. For lower number of components, the general features of the entire dataset dominate the reconstruction, with the more distinguishing features gradually appearing. In a PCA, one may not get a such a clear notion for general vs. unique features. This what is meant by tensor decomposition being better suited for multilinearity.

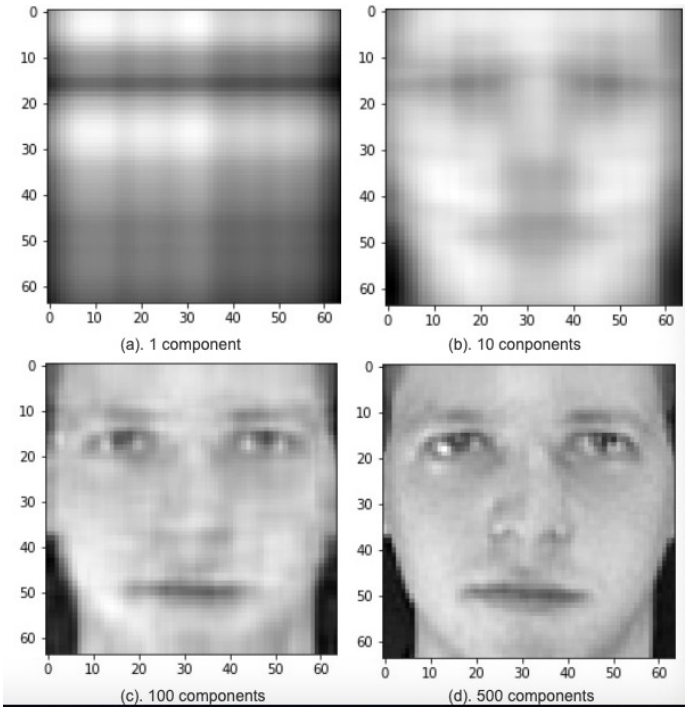


Figure 9: *Reconstructed images after CP decomposition with different components*

5.3 Tucker Decomposition

We used the same 3d TensorLy tensor for Tucker decomposition. By applying the tucker function in the tensorLy.decomposition module, we first expressed the tensor as a core tensor multiplied by a set of factor matrices, one for each mode of the tensor. We set the rank for each mode as 40. Then we converted the decomposed tensor back to its original shape by the core tensor and the factor matrices we obtained previously. At last, we computed the mean squared error between the original tensor and the reconstructed tensor, and displayed an image of the first mode of the reconstructed tensor. Below is a reconstructed image we obtained after Tucker decomposition.

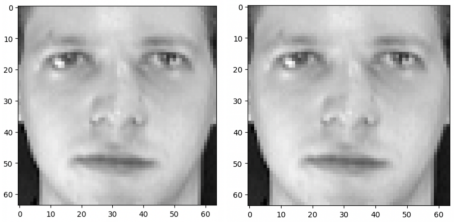


Figure 10: *Comparing the original image (left) to the reconstructed image (right) after Tucker decomposition of max rank (40,64,64)*

As we can see from Figure 10 we obtained, images undergoing Tucker decomposition demonstrate less noise and less blur than those undergoing CP decomposition when they are reconstructed. Though both of them have the rough appearance of the original image while are not as clear as the original image at the same time. Below are some reconstructed images after Tucker decomposition with different ranks (Figure 11).

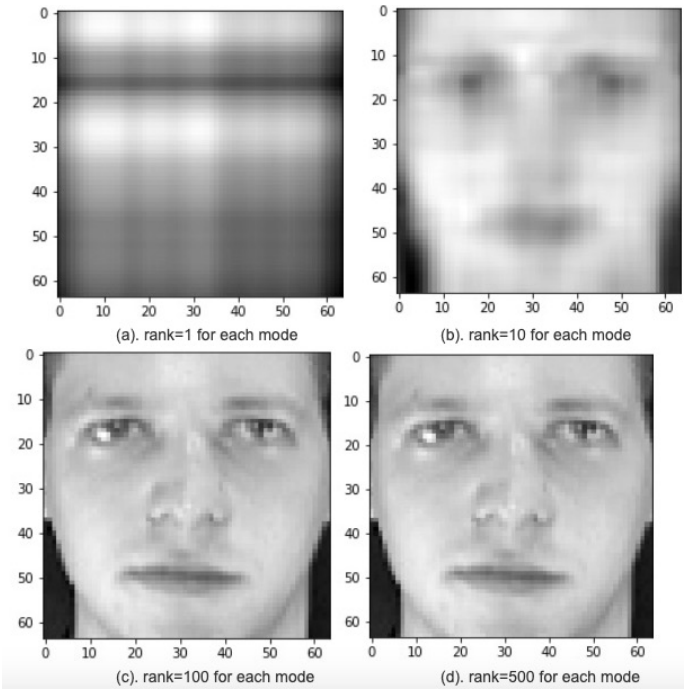


Figure 11: *Reconstructed images after Tucker decomposition with different ranks*

6 Conclusion

Tensor decomposition is an important data analysis tool that can reliably reconstruct multilinear data provide insight into the overall geometry of the data that simple PCA cannot. In our analysis, we explained why tensor problems are inherently harder than matrix problems, as well as the limitations of tensor decomposition stemming from the lack of a tensor analog for SVD. Additional difficulty comes when defining loss functions to minimize when finding a tensor decomposition. This is usually easy for matrices using the Frobenius norm or quadratic loss, but applying these functions to a tensor fails because a low rank approximation of the tensor can minimize the loss function arbitrarily close to 0. While this makes it difficult to decide on an optimal rank of approximation, the difference between a higher and lower rank decomposition can still be informative. In our example, the lower rank approximations provide a "model" for a generic human face where the distinguishing variations appear in higher rank approximations. Comparing the two methods of decomposition, we see that they are both informative with respect to the dataset. Continuing our exploration of tensor methods, we intend to define a standard to quantify the reconstruction error of the low rank tensor approximation so we can effectively compare different types of decomposition. We hope this is a useful document that can provide some intuition on the joys and toils of working with tensors to our fellow struggling undergrads.

7 References

1. T. Kolda, and B. W. Bader, "Tensor Decompositions and Applications," Society for Industrial and Applied Mathematics, Vol. 51, No. 3, pp. 455-500, 2009

2. Nicholas D. Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Member, Kejun Huang, Evangelos E. Papalexakis, and Christos Faloutsos, "Tensor Decomposition for Signal Processing and Machine Learning," IEEE Trans. on Sig. Proc., pp1, 2017.

3. M. Vasilescu and D. Terzopoulos, Multilinear Analysis of Image Ensembles: TensorFaces, pp. 447–460, Springer Berlin Heidelberg,Berlin, Heidelberg, 2002.

4. E.E. Papalexakis, C. Faloutsos, and N.D. Sidiropoulos, "Parcube: Sparse parallelizable tensor decompositions.," in ECML/PKDD (1), P.A. Flach, T.D. Bie, and N. Cristianini, Eds. 2012, vol. 7523 of Lecture Notes in Computer Science, pp. 521–536, Springer.

5. Tuncer, Yalcin, Murat M. Tanik, and David B. Allison, "An overview of statistical decomposition techniques applied to complex systems." Computational statistics data analysis 52.5 (2008): 2292–2310.

6. "Tensor Decompositions: Algorithms." Algorithmic Aspects of Machine Learning, by Ankur Moitra, Cambridge University Press, Cambridge, 2018, pp. 29–47.

8 Author Contribution

Anny : Application and Results wite-up, exploration of data, sections 5.2 and 5.3, figures:5, 6, 7, 8 , 9, 10, 11 .
Elsie : Application and Results write-up, exploration of data, sections 5.2 and 5.3, figures: 1, 2, 3, 4.
Taher : Introduction, CP Decomposition(4.1), Jennrich's proof (3.2), tensor rank (3.3), Conclusion, algorithms, data analysis coding, what is a tensor? (2.1)
Divya : Tensor n-rank, HOSVD algorithm, Tucker Decomposition (5.3 and 4.2), Matrix products (2.2: 2.2.1 and 2.2.2), Abstract, algorithms, data analysis coding

9 Corresponding Authors

If you find any mistakes/typos please email dkranth1@jh.edu and taherhaitami@gmail.com or thaitam1@jh.edu