

IMPLEMENTATION OF TENSOR DECOMPOSITION ON DIFFERENT AREAS OF COMPUTER VISION

IMPLEMENTATION OF TENSOR DECOMPOSITION ON DIFFERENT AREAS OF COMPUTER VISION

*A thesis submitted to Indian Institute of Technology Kharagpur for B.Tech
Project*

By

Sayantana Kirtaniya (18EC33002)

Under the guidance of

Professor Swanand Ravindra Khare

(Mathematics and Computing Department)



***Department of Electrical and Electronics Engineering
Indian Institute of Technology, Kharagpur***



Department of Electrical and Electronics Engineering, Indian Institute of Technology
Kharagpur, Kharagpur, India- 721302

CERTIFICATE

This is to certify that the work presented in this thesis entitled Implementation of **TENSOR DECOMPOSITION ON DIFFERENT AREAS OF COMPUTER VISION** has been carried out by Sayantan Kirtaniya under my supervision. This is his bonafide work and is worthy of consideration for a B.Tech term project - 1 of the Institute.

Prof.Swanand R Khare
*Department of Mathematics and
Computing
Indian Institute of Technology
Kharagpur
India - 721302.*

DECLARATION

I certify that:

- 1) the work described in this project is original and has been done by me by the support and guidance of my supervisor;
- 2) I have not submitted the work to any other institute for any other degree or diploma;
- 3) I have strictly adhered to ethical norms and guidelines while preparing the thesis;
- 4) I have strictly followed the guidelines presented by the Institute in doing the thesis;
- 5) Whenever I have used materials (data, models, figures, and text) from other sources, I have given due credit to authors and papers by citing them. Details of the citations are available in the references;

Sayantan Kirtaniya

ACKNOWLEDGEMENT

I am grateful to my supervisor ***Prof. Swanand R Khare*** for guiding me throughout this project. Without his valuable suggestions, I could not have presented this work. Special thanks to ***Tamara G. Kolda*** (Mathematical Consultant MathSci.ai) and ***Ankur Moitra*** (Department of Mathematics at MIT) for their wonderful work in this domain with their lecture videos and research papers, which helped me to understand the ideas and implementation of this area of research.

ABSTRACT

Tensor decomposition is a part of multilinear algebra under which we do the factorization of tensors depending upon the type of data and its use. In computational work, this method tremendously decreases the computational time and makes the process faster by making more efficient convolutional models.

In this project, we aim to decompose the image tensor keeping the accuracy of the image model intact.

Our project only focuses on the image tensor and does not change the algorithmic structures and model in any way so that the model accuracy or the loss should not get affected for the model. We have implemented the whole experiment in real-world datasets so that we can understand the real-world difficulties and thus making this particular project sustainable. Along with the implementation; we are also focusing on the research part by theoretical and algorithmic understanding, to implement the research to obtain the results that we are aiming for as a result i.e. less computation time with maximum accuracy.

Key points: Factor analysis, rotation problem, matrix decomposition, CP decomposition, CPRANK, CNN model, MNIST dataset, computational time,

CONTENTS

- 1. INTRODUCTION PAGE NO: 10 - 11**
The basic understanding of this particular domain can be found here, with some general concepts and ideas.
- 2. CONCEPTUAL AREAS..... PAGE NO:12 - 16**
This part is all about the concepts of how tensor decompositions and their implementation become a part of mathematics, how they become a major tool later on for computational purposes.
- 3. TOOLS AND ALGORITHMS PAGE NO:16 - 20**
This part is all about the algorithms which we are going to implement out of the proposed idea and to get the expected results.
- 4. PROPOSED IDEA PAGE NO:21 - 22**
Here we are proposing our idea of what we are going to implement and the proposal of the results which we are going to get after the implementation of the idea.
- 5. IMPLEMENTATION OF IDEA..... PAGE NO: 22 -23**
Here we are implementing our proposed ideas in a few steps. Anyone can find the implementation codes and results or output from them and can verify the results with the next part.
- 6. RESULTS..... PAGE NO:24 - 26**
Here we get the results from the implementation and from that we can check how much we have done, or is it making similarity with our proposed results, or how much is the deviation from the proposed ideas.
- 7. FUTURE WORK PAGE NO: 26**
Any person who wants to start working on the future in this project can continue from this particular part.
- 8. CONCLUSION PAGE NO: 27**
In this part, we focused on concluding the full project on a short note which can also help to get an overview of the results and future work.
- 9. REFERENCE PAGE NO: 27-28**
The figures, diagrams and the ideas which we are taking help, as well as code implementation, all the references are added in this part.

1. Introduction

Our project primarily focuses on the decomposition of the tensor of the image data to reduce the computation power in the field of computer vision, by keeping the accuracy of the image intact.

Tensors are the higher dimension matrix generalisations and can be seen as multi-dimensional fields. In the tensor product of N vector spaces; each element is a tensor of N th order. Tensors are considered more rigid than matrices, and their decomposition is more concerning as compared to matrix decomposition. Before going through some important terminologies let's visualise how tensors and their decomposition looks like, below is the diagram that demonstrates how fibres and slices can be created.

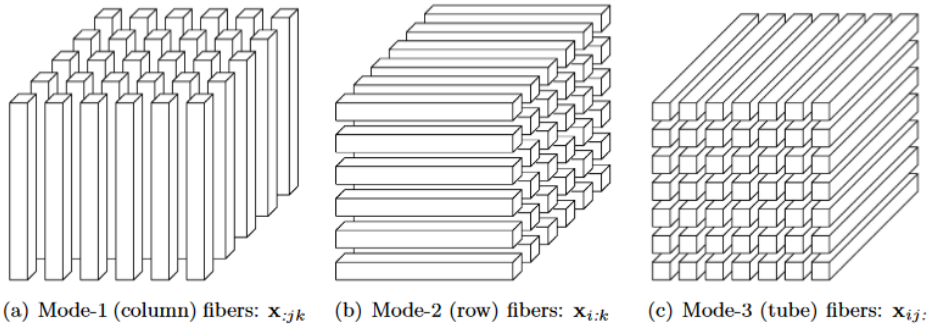


Figure 1.1: Modes of tensor(Tamara G. Kolda , Brett W. Bader 2009)

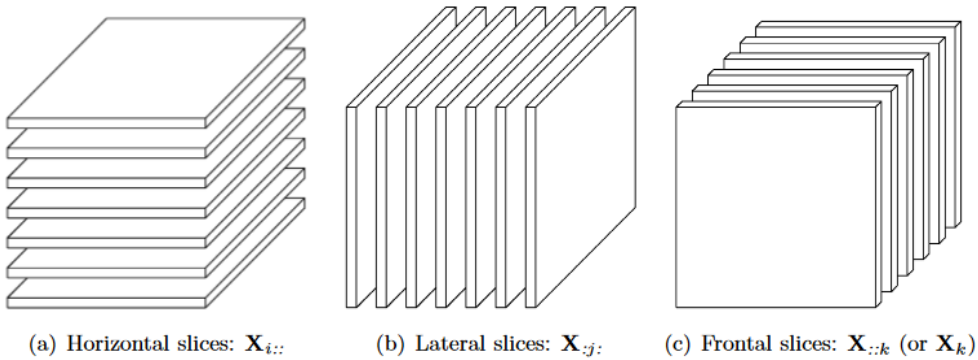


Figure 1.2: Slicing of tensor(Tamara G. Kolda, Brett W. Bader 2009)

In the **first figure**, different modes of a third-order **tensor fibre** are shown. *Mode 1* fibre shows the column segment of the tensor keeping the i component constant and varying the j and k components. *mode 2* fibre shows the row segment of the tensor keeping the j

component constant and varying i and k components. *mode 3* fibre shows the tube segment of the tensor keeping the k component constant and varying i and j components.

In the **second figure**, slicing of a third order tensor is done from three different axial positions, i.e. $x - axis(i)$, $y - axis(j)$, $z - axis(k)$.

In the later sections, we will be introducing some basic yet important concepts related to tensor decomposition, **CANDECOMP/ PARAFAC (CP)** and **Tucker**. Under the section on conceptual areas, we will be explaining spearman's hypothesis inside which we will be briefly discussing **factor analysis, rotation problem, matrix decomposition, tensor decomposition etc.** Under the **Jennrich algorithm**, we will be looking at how it can be used in slicing the matrix and **eigendecomposition**. After the conceptual areas, sections are targeting: tools and algorithms; proposed idea; making the dataset; implementation of the idea; results and hence the future work.

Since now, the use of the decomposition of higher-order tensors has been shown by the unsupervised learning domains but in recent days it is gaining popularity in temporal and multi-relational data analysis and have applications in psycho-metrics, chemometrics, signal processing, numerical linear algebra, computer vision, numerical analysis, data mining, neuroscience, graph analysis, and elsewhere.

2. Conceptual areas

2.1 Spearman's Hypothesis:

Charles Spearman, a British psychologist, speculated that human intelligence can be divided into two (hidden) factors: **educational intelligence** (the ability to understand complexity) and **reproductive intelligence** (the ability to store and reproduce information). Spearman was one of the first scientists to implement an unsupervised learning technique on a data set, which is now called **factor analysis**.

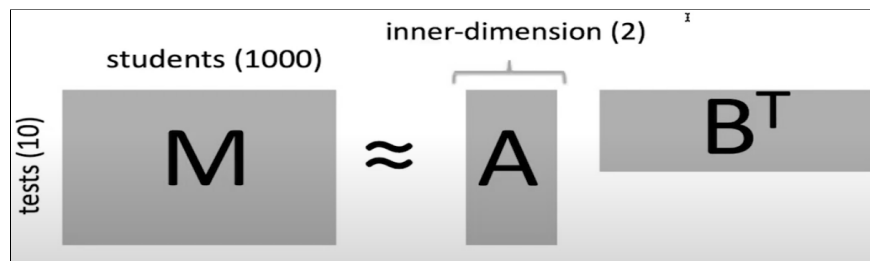


Figure 2.1.1: Spearman's Hypothesis(Ankur Moitra 2021)

S students take various tests and their scores have been stored in the matrix $M(R_{s \times t})$. Then he wondered if it was possible to just make M matrices into two smaller matrices $A(R_{s \times h})$ and $B^T(R_{h \times t})$ via $h = 2$ latent coefficients described above. According to his explanation, A will maintain a list of students with their respective academic and reproducibility and B^T will maintain a list of tests with requirements educational and regenerative needs respectively.

As a solution to this problem, a rotation(R) matrix was introduced where we can use them between the A and B^T and try to do the **factor analysis**.

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

$$\text{Given } M = \sum a_i \otimes b_i$$

Now we are introducing a matrix R between A (sum of a_i) and B^T , and rewriting the equation,

$$AB^T = (AR)(R^{-1}B^T) = (AB^T)(BR^{-1})^T = A \tilde{B} \tilde{B}^T$$

Now the issue is here that if we want to find out the factors uniquely we need to impose very strong additional conditions, which are:

- If $\{a_i\}$ and $\{b_i\}$ are orthogonal then we can find out the factorization using singular value decomposition. Or the rank of the matrix M for which we are finding the factors that have to be l , both of the points make the same sense to us.

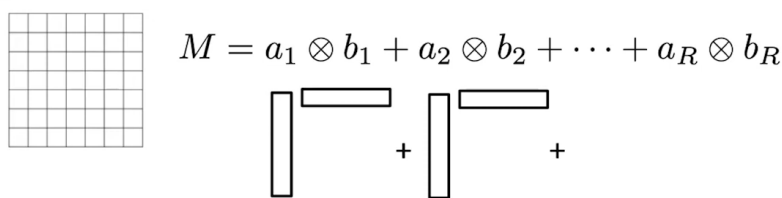
This is called the **rotation problem**, so when we do factor analysis on real data for example scientific data or any data analysis or machine learning approaches, the matrix decomposition is not unique enough for our purposes.

This particular major issue in the area of factor analysis motivated scientists to proceed with the **tensor methods**, in our project too, when we tried to decompose some image data we got better results using the tensor methods, as they are unique in much more tame conditions, as well as they, come with algorithms.

Now if we focus on the **Matrix decomposition**,

The left-hand side is the matrix M and we are presenting that as a sum of R rank 1 matrices.

MATRIX DECOMPOSITIONS

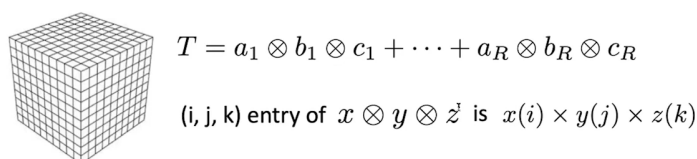


$$M = a_1 \otimes b_1 + a_2 \otimes b_2 + \cdots + a_R \otimes b_R$$

Figure 2.1.2: Matrix decomposition(Ankur Moitra 2021)

And in the case of **Tensor decomposition**, we have the 3-dimensional array of numbers, and here it is indexed by (i, j, k) .

TENSOR DECOMPOSITIONS



$$T = a_1 \otimes b_1 \otimes c_1 + \cdots + a_R \otimes b_R \otimes c_R$$

(i, j, k) entry of $x \otimes y \otimes z$ is $x(i) \times y(j) \times z(k)$

Figure 2.1.3: Tensor decomposition(Ankur Moitra 2021)

So if we take a vector a_1 , and vector b_1 and vector c_1 , and find out their tensor product that will be reconstructed as a new tensor that will be a **rank 1 tensor**. For that tensor, the i, j, k entry would be the product of the i^{th} entry of the 1^{th} vector times j^{th} entry of the 2^{nd} vector and k^{th} entry of the 3^{th} vector.

So we can see that the tensor decomposition is unique, and they are much more unique in tame conditions than matrix decompositions.

So let us discuss **JENNRICH ALGORITHM**.

2.2 Theorem[Jennrich 1970]:

Suppose we have a set of factors $\{a^i\}, \{b^i\}$ and $\{c^i\}$, and we have the following conditions,

- $\{a^i\}$ and $\{b^i\}$ are linearly **independent**.
- and there will no pair for vectors in $\{c^i\}$ will point in the same direction, means no pair of vectors in $\{c^i\}$ is a scalar multiple of each other, Then:

$$T = a^1 \otimes b^1 \otimes c^1 + a^2 \otimes b^2 \otimes c^2 + \dots + a^R \otimes b^R \otimes c^R$$

If we consider a low-rank tensor T , which is the sum of $\{a^i\}, \{b^i\}$ and $\{c^i\}$, we can say tensor decomposition is **unique** under these linear independence conditions on the factors.

By saying unique we can perform some trivial things:

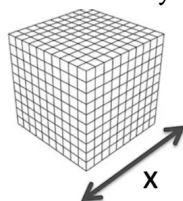
- So we can perform the rank one tensors on the right-hand side, we can swap the a^1 , b^1 and c^1 tensor with a^2 , b^2 and c^2 tensor and that will not change anything.
- Also, we can do some rescale on any of the rank one tensor factors on the right-hand side, for example, we can **double the a^1** and **half the b^1** , and that will not change that rank one tensor in the tensor decomposition.

So for those changes the tensor decomposition method is unique.

Now we are moving to the part of **Jennrich ALGORITHM**, so if we have the tensor T how can we find out these hidden factors.

So simply we can take the tensor T as $(N \times N \times N)$ so we can break it down into a matrix,

- We can choose a random vector X , so it is a dimension N vector.
- So we are going to add up the different **matrix slices**, weighted by the different entries of X . X will be chosen uniformly at random from S^{n-1} .



i.e. add up matrix slices

$$\sum_i x_i T_i$$

(x is chosen uniformly at random from S^{n-1})

Figure 2.2.1 : Jennrich ALGORITHM (Ankur Moitra 2021)

- Let's take one simple example, suppose T is a rank one tensor of factors $\{a\}, \{b\}$ and $\{c\}$, so if we hit it with a vector X , we get a rank-one matrix (not increasing the rank) and it will look like this:

$$T = a \otimes b \otimes c \text{ then } T(\cdot, \cdot, X) = \langle c, x \rangle a \otimes b$$

- Now if we think for a rank R , tensor T , if we hit it with a random vector X , then we can get a matrix rank of R because we are going to add up all the contributions from the individual rank-one matrices.

$$T(\cdot, \cdot, X) = \sum_i \langle C_i, X \rangle a_i \otimes b_i$$

- So we don't know the factors $\{a_i\}, \{b_i\}$ and $\{c_i\}$, we can actually understand what is happening inside of the Tensor as they satisfy the conditions of the **Jennrichs algorithm**.
- So now we can take the factors of $\{a_i\}$ and organize them into a matrix A , where they will be columns, we can do the same for the matrix B which columns will be $\{b_i\}$, and then we can make a diagonal matrix out of the scalars $\{c_i\}$ and X that we were supposed to multiply.

$$\text{Diag}(\{\langle C_i, X \rangle\}) \Rightarrow AD_X B^T$$

- Now we are taking another random vector Y , and the diagonal matrix will be $\{c_i\}$ and Y , and it will look like this

$$T(\cdot, \cdot, Y) \Rightarrow AD_Y B^T$$

- Now we have two matrices vector X hitting tensor T and vector Y hitting tensor T , and we can do the diagonalization of both of them, to make things simpler we can consider that the matrices are invertible,

$$\text{Diagonalize } T(\cdot, \cdot, X)(T(\cdot, \cdot, Y))^{-1} \Rightarrow AD_X B^T (B^T)^{-1} D^{-1} Y A^{-1} \Rightarrow AD_X D^{-1} Y A^{-1}$$

we can algebraically find out the form, what it's going to look like, then B^T and $(B^T)^{-1}$ can be cancelled out and we will be left with **EIGEN DECOMPOSITION**.

- Now we can construct the matrix in the above diagram, and the unknown factors we are looking for are the **Eigenvector** of this matrix and the Eigenvalues we can find out from the $D_X D_Y^{-1}$.
- For finding the $\{b_i\}$ we can do the same just by changing the form of the diagonalization, like below, and all that remains will be $\{a_i\}$ and $\{b_i\}$ and finding the $\{c_i\}$, for which we only have to solve the linear system.

$$T(\cdot, \cdot, Y)(T(\cdot, \cdot, X))^{-1}$$

- So we can match up the $\{a_i\}$ and $\{b_i\}$ because they are actually Eigenvectors whose Eigenvalues or these two Eigen decomposition problems are reciprocal to each other.

So basically this is how we are approaching the problems with tensor decompositions, as we are looking at the Jennrichs algorithms that we are studying; now we can move to some modern-day tensor decompositions algorithms and tools that will help us to get a better understanding of the domain as well as when we are going to try these methods of tensor decompositions in real-world data and do the experiment to observe how tensor decomposition can help us in the computation.

3. Tools and algorithms

➤ 3.1 MATRIX DECOMPOSITION: DETECTING THE LOW-RANK STRUCTURE

- We have a data X and we want to find a low-rank model M .

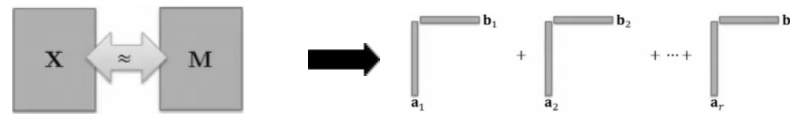


Figure 3.1.1: Matrix decomposition (Tamara G. Kolda 2018)

- Each entry of our data X is approximated by the model M , which is a sum of entries of these rank-one matrices.

$$X(i, j) \approx m(i, j) = a(i, 1)b(j, 1) + a(i, 2)b(j, 2) + \dots + a(i, r)b(j, r) = \sum_{l=1}^r a(i, l)b(j, l)$$

- We can ensemble that into a matrix notation:

$$X \approx M = \sum_{l=1}^r a_l \circ b_l = [[A, B]]$$

- Now we have a sum of **squared errors**, which is an error matrix, we try to minimize the error to get a better approximation.

$$\sum_{ij} (x(i, j) - m(i, j))^2 = ||X - M||^2$$

➤ 3.2 TENSOR DECOMPOSITION: DETECTING THE LOW-RANK 3-WAY STRUCTURE

- Now we are adding a vector $\{c\}$ for each of our rank-one tensors and we call each of the rank-one tensors as **Components**, acting as building blocks.

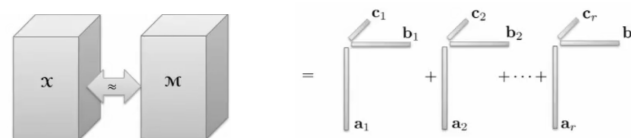


Figure 3.2.1: Tensor decomposition (Tamara G. Kolda 2018)

- Then we can write the tensor notation out of it, here we are finding the low-rank model $m(i, j, k)$ from the 3-way data $x(i, j, k)$.

$$X(i, j) \approx m(i, j) = a(i, 1)b(j, 1)c(k, 1) + a(i, 2)b(j, 2)c(k, 2) + \dots + a(i, r)b(j, r)c(k, r) = \sum_{l=1}^r a(i, l)b(j, l)c(k, l)$$

$$X \approx M = \sum_{l=1}^r a_l \circ b_l \circ c_l = [[A, B, C]]$$

- In the end, the algorithm will follow the same process of finding the **sum of squared errors (SSE)**, which will help us to give a better approximation.

$$\sum_{ijk} (x(i, j, k) - m(i, j, k))^2 = ||X - M||^2$$

➤ 3.3 SVD, HOSVD & TUCKER DECOMPOSITION

- In **Tucker Decomposition** we can try to obtain hidden information and try to obtain the reduced size of the model.
- In **SVD** we decompose the tensor into **left singular vectors, singular and right singular vectors**, where we can get the factors, **U : Left singular vectors, Σ : Singular values and V : right singular vectors**. Here is a depicted process of SVD:

$$A \approx U * \Sigma * V^T, \quad A_{[m \times n]} = U_{[m \times r]} * \Sigma_{[r \times r]} * V^T_{[n \times r]}$$

SVD (graphical representation)

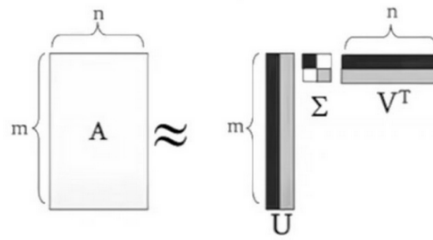


Figure 3.3.1: SVD (B.S Keyla Gonzalez 2020)

- Now basically in **Tucker Decomposition**, we get a core tensor g and along with the factor matrices.
- Basically, the **Tucker compression** extends the matrix **SVD** to multiway arrays. In this, we can get a “**core tensor**” g , “**factor matrices**” U, V, W and \times_K **tensor times matrix in mode K** , We can depict the tucker decomposition in the below diagram,

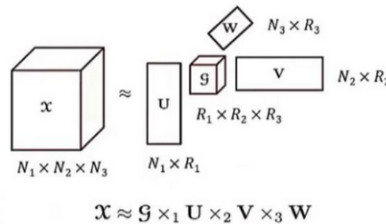


Figure 3.3.2: Factors after decomposition (B.S Keyla Gonzalez 2020)

- We also have to understand that how much the matrix should be compressed we can obtain a compression ratio from the equation below.

$$C \approx \prod_{k=1}^d \frac{N_k}{R_k}$$

- We can also find out the tucker decomposition ranks by inverting the factors vectors (U, W, V) on the tensor X side like this and for a given relative error, we can write the below equations such that the core tensor should satisfy the condition.

For a given relative error ϵ choose projection ranks R_1, R_2 and R_3 such that,

$$\|X - (g \times_1 U \times_2 V \times_3 W)\| \leq \epsilon \|X\|$$

Such that the core tensor satisfies,

$$\|g\| = \|X \times_1 U' \times_2 V' \times_3 W'\|, \|X\|^2 - \|g\|^2 \leq \epsilon \|X\|^2$$

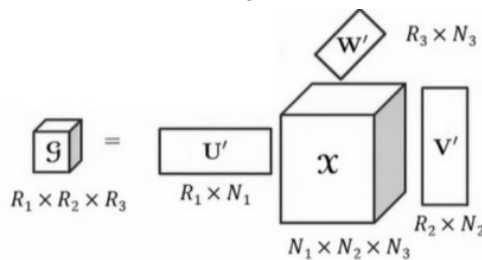


Figure 3.3.3: Determining rank for decomposition(B.S Keyla Gonzalez 2020)

- If we move forward, then we can get better solutions with **HOSVD**, in which there will be orthogonality of the core tensor and factor matrices.
- We also can implement **St-HOSVD** which is an improvement of **HOSVD**, where we shrink the tensor at each step and reduce subsequent computations at the steps.

➤ 3.4 CP-ALS (ALTERNATING LEAST SQUARES)

- So in **CP** decomposition, we do the error and approximation of the tensor T if it is the input, Now we can fit the model using **alternating least squares methods**.
- If we make an equation where we make the factors $\{b\}$ and $\{c\}$ fix and do the same for $\{b\}$ and $\{c\}$ as well for other factors, then they will be linear least-squares problems which will be really easy to resolve, then every single time the factors $\{a\}$, $\{b\}$ and $\{c\}$ will get updated, as the function's value of the error will monotonically decrease throughout and we will continue till **convergence**.

$$\text{Step1} \dots \min(A) \left(\sum_{ijk} (x_{ijk} - \sum_l a_{il} b_{jl} c_{kl}) \right)^2$$

$$\text{Step2} \dots \min(B) \left(\sum_{ijk} (x_{ijk} - \sum_l a_{il} b_{jl} c_{kl}) \right)^2$$

- *Step3* $\min(C) \left(\sum_{ijk} (x_{i,j,k} - \sum_l a_{il} b_{jl} c_{kl}) \right)^2$
..... **repeat the process till convergence**
- It is a **non-convex** problem but its sub-problems are convex sub-problems, as those are the least-squares problems that are solvable.
- Now let us look deeper inside the process, we are solving for the set of vectors in matrix A , and we put B and C together, by making a combination using **matrix Khatri-Rao Product** (its Kronecker product along the rows of a matrix, generally we use it for making combinations, symbol: \odot),

$$\min(A) \left(\sum_{ijk} (x_{i,j,k} - \sum_l a_{il} b_{jl} c_{kl}) \right)^2 \Rightarrow \min(A) \| X_{(I)} - A(C \odot B)^T \|_F^2$$

We can depict the equation in a clear view in the below diagram,

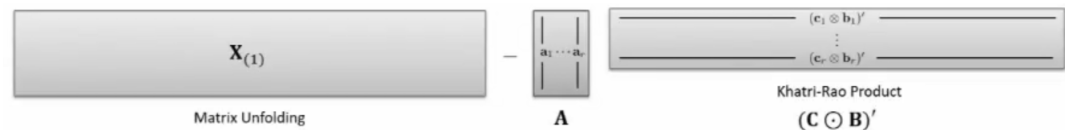


Figure 3.4.1: CP ALS (Tamara G. Kolda 2018)

- Basically in **CP-ALS**, we get the **Khatri Rao** product which is a very long and wide matrix and when it becomes a d-way tensor problem increases, so let us suppose we have an $n \times n^2$ matrix then the A will be $n \times r$ and the other matrix will be $r \times n^2$ where the r is the number of components after the decomposition, this is going to happen is the case of 3-way tensor,
 - 3 way case $n \times n^2$ $n \times r$ $r \times n^2$
 - d way case $n \times n^{d-1}$ $n \times r$ $r \times n^{d-1}$
- For the d-way tensor, the Khatri Rao product will be so big as it will be $r \times n^{d-1}$, then it becomes a very overdetermined problem.

➤ 3.5 CPRAND AND CPRAND-MIX

- As a solution to the last problem, we choose the **CPRAND** method, where we try to do some randomization.
- So we can pick a subset of the column of the matrix which we are making and also from the X which one we are decomposing we are picking out some of the **fibres** using a sampling operator, like this:



Figure 3.5.1: CP RAND (Tamara G. Kolda 2018)

- Then we solve a bit smaller problems which is much easier to compute and much more useful in this scenario because it is not just **heuristics**, it strongly proved that how good is this **randomization technique** is, it's going to look like this after the randomisation: $\|X_{(l)}S - A(C \odot B)S\|_F^2$
we can depict it below:

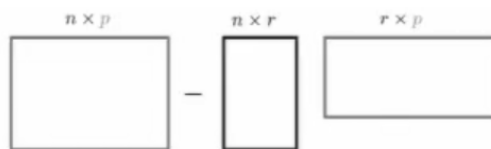


Figure 3.5.2: randomized sample for decomposition (Tamara G. Kolda 2018)

- There are two ways we try to follow in CPRAND
 - We never permute elements of tensor X into $n \times n \times n$ form, we actually in the tensor form and pull out some of the elements we need.
 - We never actually form the full Khatri Rao product before randomization, as it's going to take some computation to form it, we pick some of the fibres from there and keep that in normal form.
- CPRANDMIX:** If we say the matrix is incoherent, means there should a good mix in the data in each direction,
 - Method name **Johnson Lindenstrauss method** will make the matrix incoherent.
 - Adding that method can add some preprocessing cost, so there is **the Fast Johnson Lindenstrauss transforms** which will make the process faster

➤ 3.6 STOPPING CRITERIA

- In the **CPALS** initially, we defined that there will be convergence or till convergence, we are training the model.
- So basically we take another idea of randomisation, we just taking a poll of a few of the entries, like this:

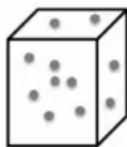


Figure 3.6.1: checking bottleneck (Tamara G. Kolda 2018)

- We estimate the convergence of the function values using a small random subset of elements in function evaluation, and we can bound how accurate our estimation is using **Chernoff Hoeffding bound accuracy**.

Now we can move forward and try something out on our own where we can try some tensor decomposition methods in real-world data and get some results out of it.

4. Proposed idea

Here is an idea we are proposing, we are going to complete our experiment in two steps:

Step 1:

For this experiment, we are going to take the **MNIST Dataset** and we are going to make a **CNN** model which should give very good accuracy in the **MNIST** dataset, and we are going to train the model until we get good accuracy in the test dataset, in this process, we are going to do the training process and observe the loss and training time in each epoch and at the end, we are going to note the test accuracy.

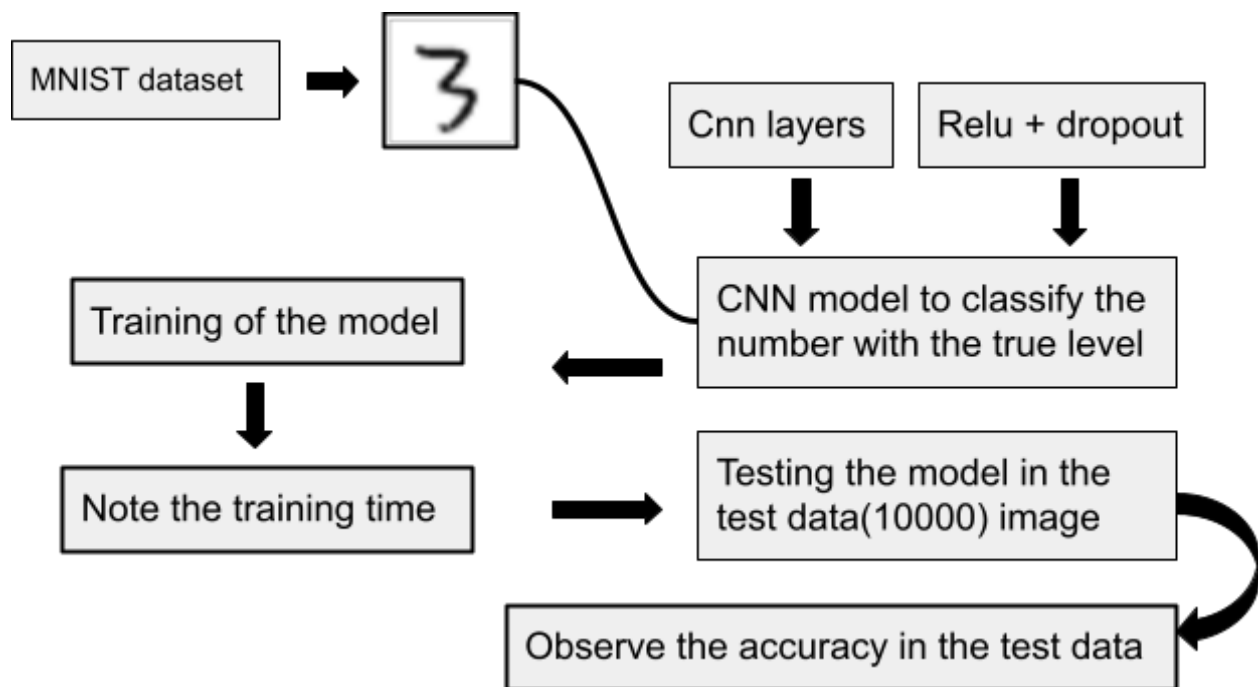


Figure 4.1

Step 2:

For the next part, we are focusing on the decomposition on the tensor decomposition part more and we are trying to decompose the whole dataset so that we can get better results in the computation of training, after saving the dataset. We are going to try **STEP 1** again in the decomposed dataset and observe the training time this time along with testing accuracy.

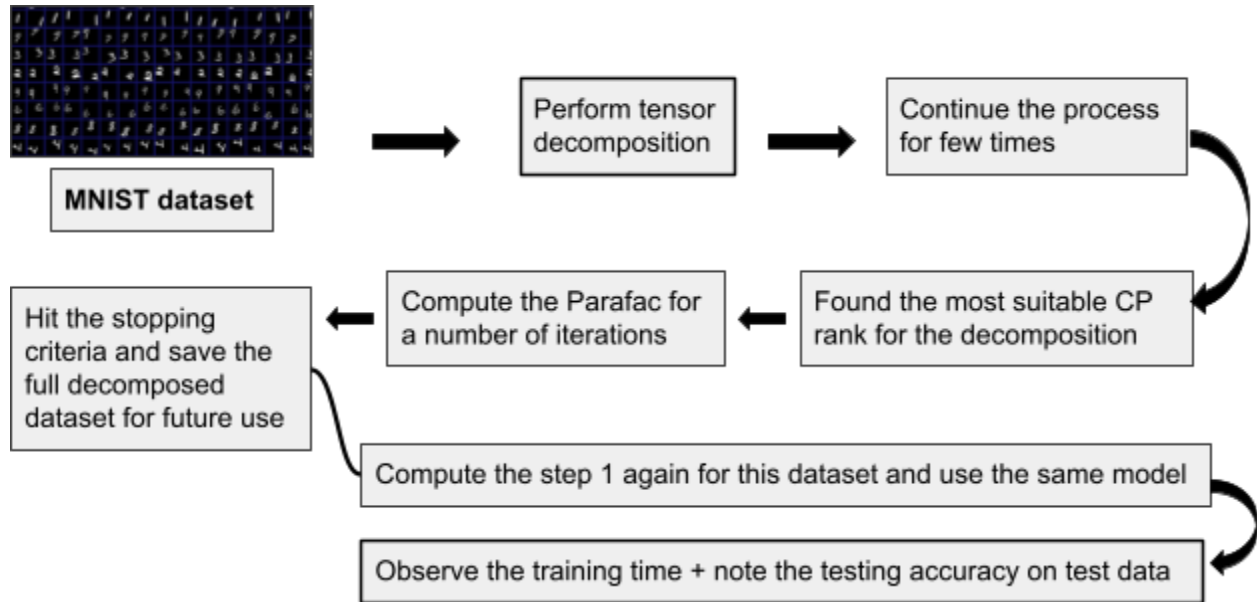


Figure 4.2

PROPOSAL

As we have understood from the theoretical part that how **CP** can help us to decrease the computation in the tensor training process, thus we can propose that:

- In **Step 2**, the time of computation will be significantly less than **step 1**, such that for long computations time it will help us a lot.
- The accuracy of **step 2** and **step 1** should not be that much different, as we are only decomposing the tensor, not changing the model or number of epochs, though there is a possibility of getting lesser accuracy in **step 2** as the data will be decomposed.

Now we can focus on the implementation of the idea, in the below we have done that.

5. Implementation of the idea

5.1 We are going to use some external libraries for completing the experiment:

- Tensorly-0.7.0
- Pytorch
- Numpy
- Pandas
- Matplotlib

5.2 The hardware which is used here to complete the experiment:

- NVIDIA-SMI 495.44
- CUDA Version: 11.2
- GPU Tesla V100-SXM2
- GPU memory 16160MiB

- RAM: 54.8 gigabytes
- VM: Google Colab pro plus

5.3 Now we are going to implement the idea and get the results after implementing the idea:

- In the first part, we can just take a high-resolution image and perform different kinds of decomposition techniques and see the output image size and resolution as well as quality. This particular step is for understating the methods and how much affects the images.
- Next, we are implementing the CNN model in step one, and we can train the model for 20 epochs and test in 10000 test images. Here is the depiction of the image model we are building:

```
ConvNet(
  (layer1): Sequential(
    (0): Conv2d(1, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (layer3): Sequential(
    (0): Conv2d(128, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc): Linear(in_features=1152, out_features=10, bias=True)
)
```

Here is the link of the code, implementation of the model along with the results:

STEP 1 CODE LINK

- Next, we have implemented **CP RAND** using the Tensorly library on the MNIST dataset and with determining the CP rank and after that, we are focusing on training the model and as well as saving the newly made dataset as a tensor so that we can use it in our further research.

Here is the newly made dataset link: **DECOMPOSED MNIST dataset**

- We are also training the model in this dataset for 20 epochs and got great results, we can discuss that more in the results part.

Here is the code link where we have implemented STEP 2 fully, from making the dataset to implementing the model on the newly made dataset: **STEP 2 CODE LINK**

6. Results

6.1 OBSERVATION ON THE TRAINING TIME

All the time is observed using **tqdm** library and noted in **GPU time**:

- **NORMAL MNIST:** OUTPUT OF A PREVIOUSLY TRAINED MODEL USING REAL-WORLD MNIST DATA SET (IN DIFFERENT BACKGROUND COLOUR)
- **DECOMPOSED MNIST:** OUTPUT OF THE SAME MODEL MADE PREVIOUSLY ,TRAINED ON NEW DECOMPOSED MNIST DATASET (IN WHITE BACKGROUND COLOUR)

DATATYPE / LEVEL OF COMPARISON	EPOCH 1/20	EPOCH 5/20	EPOCH 10/20	EPOCH 15/20	EPOCH 20/20
DECOMPOSED MNIST Time of computation	00:04 s	00:20 s	00:40 s	01:00 s	01:20 s
LOSS OF DECOMPOSED MNIST(CrossEntropyLoss)	0.1351	0.0360	0.0059	0.0670	0.0003
NORMAL MNIST Time of computation	00:06 s	00:32 s	01:05 s	01:37 s	02:09 s
LOSS OF NORMAL MNIST (CrossEntropyLoss)	0.2092	0.0122	0.0009	0.0003	0.0014

TABLE 6.1.1. comparison table of normal mnist data computation vs decomposed mnist data computation, wrt. computation time and cross-entropy loss.

Discussion on the results of TRAINING TIME:

- From the table, we can clearly understand how the different models have taken the computation time, for each of the steps we can observe that the model training on the decomposed dataset is taking very less time with respect to the model which is trained on the normal **MNIST dataset**.

- If we do a rough calculation we can predict that how much tensor decomposition is going to help us in the bigger computational cases, as we can see here there is a time gap of (Normal MNIST model training time - Decomposed MNIST training time) = **(02:09-01:20) ~ 00:49 GPU time.**
- Here are the output snippets of both of the training process last epochs:

```
Epoch [20/20], Step [200/938], Loss: 0.0001
Epoch [20/20], Step [300/938], Loss: 0.0058
Epoch [20/20], Step [400/938], Loss: 0.0002
Epoch [20/20], Step [500/938], Loss: 0.0048
Epoch [20/20], Step [600/938], Loss: 0.0195
Epoch [20/20], Step [700/938], Loss: 0.0000
Epoch [20/20], Step [800/938], Loss: 0.0297
Epoch [20/20], Step [900/938], Loss: 0.0005
100%|██████████| 20/20 [02:09<00:00, 6.48s/it]
```

CNN model training of the normal MNIST data 20 the epoch

```
Epoch [20/20], Step [100/938], Loss: 0.0052
Epoch [20/20], Step [200/938], Loss: 0.0004
Epoch [20/20], Step [300/938], Loss: 0.0045
Epoch [20/20], Step [400/938], Loss: 0.0072
Epoch [20/20], Step [500/938], Loss: 0.0388
Epoch [20/20], Step [600/938], Loss: 0.0040
Epoch [20/20], Step [700/938], Loss: 0.0021
Epoch [20/20], Step [800/938], Loss: 0.0166
100%|██████████| 20/20 [01:20<00:00, 4.02s/it]
```

CNN model training of the decomposed MNIST data 20 the epoch

6.2 OBSERVATION ON THE ACCURACY OF THE TEST DATA

THE DATASET TYPE FOR TRAINING THE MODEL	ACCURACY
ACCURACY OF THE TEST DATA FOR DECOMPOSED MNIST MODEL	98.47 %
ACCURACY OF THE TEST DATA FOR NORMAL MNIST MODEL	98.27%

TABLE 6.2.1. comparison table of normal mnist data computation vs decomposed mnist model accuracy, wrt. Test set of MNIST model

- This result tells how beautiful and efficient tensor decomposition is. We have decomposed the full dataset so that we can train the model in less computational time. We succeed in that, but also we get almost the same accuracy for both of the cases.

- There is a small change of 0.2% in the accuracy, which is really negligible as this is real-world data and we can consider that amount, we are adding the output snips here of the different models.

```
Model Accuracy on the 10000 test images: 98.47 %
```

CNN model accuracy on test data for the normal MNIST dataset

```
Model Accuracy on the 10000 test images: 98.27 %
```

CNN model accuracy on test data for the decomposed MNIST dataset

- In the end, we can say after observing the results, we can prove the point which we were discussing theoretically, that the tensor decomposition method is very unique for other factorization techniques.

We feel we can progress furthermore with this project and try to implement more ideas in this field, we can discuss that in the future work below.

7. Future work

Here we have got the improved results we can progress further with these ideas:

- **Implementing this idea on other big datasets:**
There are lots of datasets like **GOOGLE landmark** which are pretty big and take too much time to compute models like efficient net, we can try to implement our proposed idea on this kind of dataset so that we can get better results, and make a better model out the decomposed dataset.
- **Factorized convolutional layers on models:**
As we are successfully able to make the model from the decomposed dataset, we can try to use some factorized convolutional layers on state of art models like **VGG, ResNet, Rexnet, efficient net** etc; already there are lots of work in this factorized convolutional layers implementation but we can make the model further better using the decomposition of image dataset, plus adding factorized convolutional layers.
- **Try to build some new algorithms for making better tensor decompositions :**
As we were trying to implement the ideas of tensor decompositions we have observed there are lots of areas that we can improve and make the algorithms better for some techniques, which will be a great contribution to this research community.

8. Conclusion

In this project where we have implemented the proposed ideas after getting the theoretical understanding initially while doing the project. We have successfully achieved good results in the computation after decomposing the tensor which is visible from observation and results. Our target was not only to decrease the time, but we had also focused to get a good amount of model accuracy after decomposing the dataset and that gave an extraordinary result too as we can see we got only 0.2% change in the accuracy from the real data. Since in a real-world scenario it is difficult to get that much accuracy in general and that is why we can say that we have successfully achieved our goal which was to understand the basic understanding of tensor decomposition, making a problem statement as well as implementing and getting results which we proposed from our understanding of the theoretical concepts, this project has created a new perception towards the computer vision techniques and helped me to get a better understanding of this research domain.

9. Reference

Full citations information and papers information:

- Tamara G., Kolda - (Member of Technical Staff at Sandia National Laboratories) , Brett W. Bader - (Sr. Manager, Maxar Technologies), 2009, Tensor decomposition and application
- Stephan Rabanser - (Department of Informatics Technical University of Munich), Oleksandr Shchur - (Department of Informatics Technical University of Munich), Stephan Günnemann - (Department of Informatics Technical University of Munich), 2017, Introduction to Tensor Decompositions and their Applications in Machine Learning
- Casey Battaglino - (Georgia Tech), Grey Ballard - (Assistant Professor in the Computer Science Department at Wake Forest University) and Tamara G. Kolda - (Member of Technical Staff at Sandia National Laboratories), 2017, A PRACTICAL RANDOMIZED CP TENSOR DECOMPOSITION
- Ankur Moitra - (Massachusetts Institute of Technology), Tensor Decompositions and their Applications
- Tamara G. Kolda - (Sandia National Laboratories) January 11 ,2018 ,Tensor Decomposition: A Mathematical Tool for Data Analysis

- B.S Keyla Gonzalez, PETE 691 - RESEARCH SPRING 2020, Tensor Decomposition (Tucker and HOSVD) with MATLAB example
-