Application of CP Decomposition

Submitted by

Ranjan Sarkar

(Roll No. 21MA40022)

Under the Supervision of

Prof. Swanand R. Khare

Department of Mathematics
Indian Institute of Technology Kharagpur
West Bengal, India - 721302

Signature of the Student
Date:

Signature of the Supervisor
Date:

Acknowledgement

I am heartily thankful to my supervisor, **Prof. Swanand R. Khare**, whose encouragement, guidance and support enabled me to develop an understanding of this topic. Finally I thank to Mathematics Department which provides all facilities required to do this project.

Ranjan Sarkar

Abstract

In this project, I have learned about Tensor, various types of tensor specially rank-one tensors; it's operations and some of the decomposition techniques of the higher order tensors. By decomposing a higher dimensional tensor in a efficient way, we can optimize the time to calculate tensor operations. Decompositions of higher-order tensors (i.e., N way arrays with N 3) have a lot of applications including signal processing, numerical linear algebra, computer vision, image processing, data mining, graph analysis, and elsewhere.

CP decomposition algorithm is implemented on Image Data (3 way tensor) to compress the size of the data. It is an iterative process where I've used Alternative Least Square (ALS) method to reach the solution. An example from MNIST Dataset has been shown and also the comparison of compression quality is there. And finally the Computational Complexity of this algorithm has been discussed.

Keywords: Tensor, Tensor Decomposition, multiway arrays, multilinear algebra, canonical decomposition (CANDECOMP), Alternative Least Square (ALS) method, Data Processing, Image Compression, Approximation, Optimization, Higher Dimensional Data Handling

Contents

1 Introduction **5**
- 1.1 What is Tensor? . . . 5
  - 1.1.1 Subarray . . . 5
  - 1.1.2 Fibers and Slices . . . 5
  - 1.1.3 Norm . . . 6
  - 1.1.4 Rank-One Tensor . . . 6
- 1.2 Matricization (Transforming a Tensor into a Matrix) . . . 6
- 1.3 Tensor Product / Multiplication . . . 7
  - 1.3.1 n-mode Product . . . 7
  - 1.3.2 Kronecker Product of 2 Matrices . . . 8
  - 1.3.3 Khatri-Rao Product . . . 9
- 1.4 Tensor Rank and the CP Decomposition . . . 9
- 1.5 Low-Rank Approximations . . . 10

2 ALS: Alternative Least Square **12**
- 2.1 Least squares problem . . . 12
- 2.2 Solution of Least Square . . . 12
- 2.3 Matrix Least Square . . . 13
- 2.4 ALS Algorithm to find Factors of CP Decomposition . . . 14
- 2.5 Python Code for CP Decomposition and Image Compression . . . 15
  - 2.5.1 Input Section . . . 15
  - 2.5.2 ALS: Alternating Least Square . . . 16
- 2.6 Image Data Compression . . . 17
- 2.7 Computational Cost . . . 19

References **20**

Chapter 1

Introduction

1.1 What is Tensor?

A Tensor is a multi-dimensional array, a general form of a Matrix (which is a 2-dim array). More formally, an N-way or Nth-order tensor is an element of the tensor product of N vector spaces, each of which has its own coordinate system.

A first-order tensor is a vector, a second-order tensor is a matrix, and tensors of order three or higher are called higher-order tensors.

1-dim array - $(1, 2, 3) - \mathbf{a}$ (Vector)

2 -dim array - $\begin{pmatrix} 3 & 3 & 9 \\ 3 & 3 & 2 \end{pmatrix} - \mathbf{A}(2 \times 3$ Matrix)

3-way array - $\left(\begin{pmatrix} 2 & 8 & 2 & 9 \\ 3 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 6 & 8 & 1 & 1 \\ 2 & 2 & 3 & 3 \end{pmatrix}\right) - \boldsymbol{\chi}(2 \times 2 \times 4$ Tensor)

Image is a 2D object that's why we need 2-way array/tensor for black white image. But for color image we need 1 more dimension having 3 channels: RGB (Red, Green, Blue). So, we'll be dealing **3-way tensors** for color image data.

1.1.1 Subarray

Subarrays are formed when a subset of the indices is fixed. For matrices, these are the rows and columns. A colon is used to indicate all elements of a mode. Thus, the jth column of A is denoted by $a_{:j}$, and the ith row of a matrix A is denoted by a_i :

The followings are the subarrays of Matrix A.

$$\text{Rows of A:} \quad \begin{matrix} \mathrm{a}_{1:} \\ \mathrm{a}_{2:} \end{matrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$$

$$\text{Columns of A:} \quad \mathrm{a}_{:1} = \begin{pmatrix} a_{11} \\ a_{21} \end{pmatrix}, \quad a_{:2} = \begin{pmatrix} a_{12} \\ a_{22} \end{pmatrix}, \quad a_{:3} = \begin{pmatrix} a_{13} \\ a_{23} \end{pmatrix}$$

1.1.2 Fibers and Slices

Fibers are the higher-order analogue of matrix rows and columns. A fiber is defined by fixing every index but one. A matrix column is a mode-1 fiber and a matrix row is a mode-2 fiber. Third-order tensors have column, row, and tube fibers, denoted by $x_{:jk}, x_{i:k}$, and $x_{ij:}$, respectively.

Slices are two-dimensional sections of a tensor, defined by fixing all but two indices. The horizontal, lateral, and frontal slides of a third-order tensor X, denoted by $X_{i::}, X_{:j:}$, and $X_{::k}$, respectively.

1.1.3 Norm

The norm of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the square root of the sum of the squares of all its elements, i.e.,

$$\|\mathcal{X}\| = \sqrt{\sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \cdots i_N}^2}$$

This is analogous to the matrix Frobenius norm, which is denoted $\|A\|$ for a matrix.

The inner product of two same-sized tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the sum of the products of their entries, i.e.,

$$\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \cdots i_N} y_{i_1 i_2 \cdots i_N}$$

It follows immediately that $\langle \mathcal{X}, \mathcal{X} \rangle = \|\mathcal{X}\|^2$.

1.1.4 Rank-One Tensor

An N-way tensor $\mathfrak{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is rank one if it can be written as the outer product of N vectors, i.e.,

$$\mathcal{X} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \cdots \circ \mathbf{a}^{(N)}.$$

The symbol "o" represents the vector outer product. This means that each element of the tensor is the product of the corresponding vector elements:

$$x_{i_1 i_2 \cdots i_N} = a_{i_1}^{(1)} a_{i_2}^{(2)} \cdots a_{i_N}^{(N)} \text{ for all } 1 \leq i_n \leq I_n$$

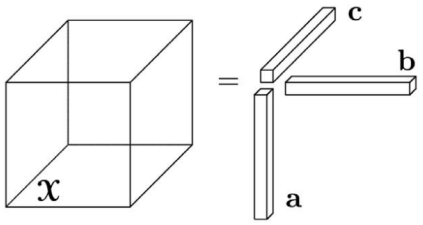

Figure 1.1: A third-order rank-one tensor
(reprinted from [**1**])

Example of 3rd order Rank-One Tensor -

$$\begin{aligned} X &= \begin{pmatrix} 1 & 2 \end{pmatrix} \circ \begin{pmatrix} -1 & 1 \end{pmatrix} \circ \begin{pmatrix} 3 & 2 \end{pmatrix} \\ &= \left[\begin{bmatrix} -3 & -2 \\ 3 & 2 \end{bmatrix}, \begin{bmatrix} -6 & -4 \\ 6 & 4 \end{bmatrix} \right] \end{aligned}$$

1.2 Matricization (Transforming a Tensor into a Matrix)

Matricization, also known as unfolding or flattening, is the process of reordering the elements of an N-way array into a matrix. For instance, a $2 \times 3 \times 4$ tensor can be arranged as a 6×4 matrix or a 3×8 matrix, and so on. The mode- n matricization of a tensor $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is denoted by $\mathbf{X}_{(n)}$ and arranges the mode- n fibers to be the columns of the resulting matrix. Though conceptually simple, the formal notation is clunky. Tensor element $(i_1, i_2, \ldots, i_N)$ maps to matrix element (i_n, j), where

$$j = 1 + \sum_{\substack{k=1 \\ k \neq n}}^{N} (i_k - 1) J_k \quad \text{with} \quad J_k = \prod_{\substack{m=1 \\ m \neq n}}^{k-1} I_m$$

The concept is easier to understand using an example. Let $\mathcal{X} \in \mathbb{R}^{2 \times 3 \times 4}$ be

$$\mathcal{X} = \left[\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{array} \right], \left[\begin{array}{cccc} 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 \end{array} \right]$$

Then the three mode- n unfoldings are

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

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

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

it is also possible to vectorize a tensor. In the example above, the vectorized version is,

$$\text{vec}(\boldsymbol{X}) = \left[\begin{array}{c} 1 \\ 2 \\ \vdots \\ 24 \end{array} \right]$$

1.3 Tensor Product / Multiplication

1.3.1 n-mode Product

Tensors can be multiplied together, though obviously the notation and symbols for this are much more complex than for matrices. Here we consider only the tensor n-mode product, i.e., multiplying a tensor by a matrix (or a vector) in mode n.

The n-mode (matrix) product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a matrix $\mathbf{U} \in \mathbb{R}^{J \times I_n}$ is denoted by $\mathcal{X} \times_n \mathbf{U}$ and is of size $I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$. Elementwise, we have

$$(\mathcal{X} \times_n \mathbf{U})_{i_1 \cdots i_{n-1} j i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \cdots i_N} u_{j i_n}.$$

Each mode- n fiber is multiplied by the matrix $\mathbf{U}$. The idea can also be expressed in terms of unfolded tensors:

$$\boldsymbol{y} = \boldsymbol{X} \times_n \mathbf{U} \quad \Leftrightarrow \quad \mathbf{Y}_{(n)} = \mathbf{U}\mathbf{X}_{(n)}$$

The n-mode product of a tensor with a matrix is related to a change of basis in the case when a tensor defines a multilinear operator.

Example of n-mode matrix product.

$$\mathcal{X}_{2\times 2\times 3} = \left[\begin{bmatrix} 1 & 0 & 2 \\ 1 & -1 & -2 \end{bmatrix}, \begin{bmatrix} 2 & 1 & 3 \\ 1 & 2 & -3 \end{bmatrix}\right]$$

$$U_{2\times 3} = \begin{bmatrix} 1 & -1 \\ 2 & 0 \\ 1 & 1 \end{bmatrix}$$

$$X_{(1)} = \begin{bmatrix} 1 & 0 & 2 & 1 & -1 & -2 \\ 2 & 1 & 3 & 1 & 2 & -3 \end{bmatrix}$$

$$U_{2\times 3} \times X_{(1)} = \begin{bmatrix} -1 & -1 & -1 & 0 & -3 & 1 \\ 2 & 0 & 4 & 2 & -2 & -4 \\ 3 & 1 & 5 & 2 & 1 & -5 \end{bmatrix}$$

$$\mathcal{X} \times_1 U = \left[\begin{bmatrix} -1 & -1 & -1 \\ 0 & -3 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 0 & 4 \\ 2 & -2 & -4 \end{bmatrix}, \begin{bmatrix} 3 & 1 & 5 \\ 2 & 1 & -5 \end{bmatrix}\right]$$

A few facts regarding n-mode matrix products are in order. For distinct modes in a series of multiplications, the order of the multiplication is irrelevant, i.e.,

$$\mathcal{X} \times_m \mathbf{A} \times_n \mathbf{B} = \mathcal{X} \times_n \mathbf{B} \times_m \mathbf{A} \quad (m \neq n).$$

If the modes are the same, then

$$\mathcal{X} \times {}_n\mathbf{A} \times {}_n\mathbf{B} = \boldsymbol{X} \times {}_n(\mathbf{BA})$$

The n-mode (vector) product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a vector $\mathbf{v} \in \mathbb{R}^{I_n}$ is denoted by $\mathcal{X} \overline{\times}_n \mathbf{v}$. The result is of order $N-1$, i.e., the size is $I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N$. Elementwise,

$$(\boldsymbol{X} \overline{\times}_n \mathbf{v})_{i_1 \cdots i_{n-1} i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \cdots i_N} v_{i_n}$$

The idea is to compute the inner product of each mode- n fiber with the vector $\mathbf{v}$.

For example, let $\boldsymbol{X}$ be as given and define $\mathbf{v} = \begin{bmatrix} 1 & 2 \end{bmatrix}^\top$. Then

$$\mathcal{X} \overline{\times}_1 \mathbf{v} = \begin{bmatrix} 5 & 2 & 8 \\ 3 & 3 & -8 \end{bmatrix}$$

When it comes to mode- n vector multiplication, precedence matters because the order of the intermediate results changes. In other words,

$$\mathcal{X} \overline{\times}_m \mathbf{a} \overline{\times}_n \mathbf{b} = (\mathcal{X} \overline{\times}_m \mathbf{a}) \overline{\times}_{n-1} \mathbf{b} = (\mathcal{X} \bar{\times}_n \mathbf{b}) \overline{\times}_m \mathbf{a} \quad \text{for} \quad m < n$$

1.3.2 Kronecker Product of 2 Matrices

The Kronecker product of matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$ is denoted by $\mathbf{A} \otimes \mathbf{B}$. The result is a matrix of size $(IK) \times (JL)$ and defined by

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

Examples:

$$A \in \mathbb{R}^{2\times 2}, B \in \mathbb{R}^{2\times 2}, \text{ then } A \otimes B \in \mathbb{R}^{4\times 4}$$

$$A = \begin{bmatrix} 1 & 2 \\ -1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & -1 \\ -2 & 3 \end{bmatrix}, \text{ then } A \otimes B = \begin{bmatrix} 1 & -1 & 2 & -2 \\ -2 & 3 & -4 & 6 \\ -1 & 1 & 0 & 0 \\ 2 & -3 & 0 & 0 \end{bmatrix}$$

1.3.3 Khatri-Rao Product

The Khatri-Rao product is the "matching columnwise" Kronecker product. Given matrices $\mathbf{A} \in \mathbb{R}^{I\times K}$ and $\mathbf{B} \in \mathbb{R}^{J\times K}$, their Khatri-Rao product is denoted by $\mathbf{A} \odot \mathbf{B}$. The result is a matrix of size $(IJ) \times K$ defined by

If $\mathbf{a}$ and $\mathbf{b}$ are vectors, then the Khatri-Rao and Kronecker products are identical, i.e., $\mathbf{a} \otimes \mathbf{b} = \mathbf{a} \odot \mathbf{b}$.

Examples:

$$A \in \mathbb{R}^{2\times 2}, B \in \mathbb{R}^{3\times 2}, \text{ then } A \odot B \in \mathbb{R}^{6\times 2}$$

$$A = \begin{pmatrix} 2 & 1 \\ -1 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & -2 \\ 2 & 1 \\ 3 & 0 \end{pmatrix}, \text{ then } A \odot B = \begin{pmatrix} 2 & -2 \\ 4 & 1 \\ 6 & 0 \\ -1 & 0 \\ -2 & 0 \\ -3 & 0 \end{pmatrix}$$

1.4 Tensor Rank and the CP Decomposition

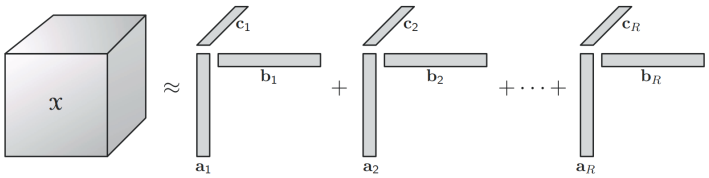

Figure 1.2: *CP* decomposition of a three-way array.

(reprinted from [**1**])

The CANDECOMP/PARAFAC (CP) decomposition factorizes a tensor into a sum of component rank-one tensors. For example, given a third-order tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$, we wish to write it as

$$\boldsymbol{X} \approx \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r, \tag{1.1}$$

where R is a positive integer and $\mathbf{a}_r \in \mathbb{R}^I, \mathbf{b}_r \in \mathbb{R}^J$, and $\mathbf{c}_r \in \mathbb{R}^K$ for $r = 1, \ldots, R$. Elementwise, Eq. 1.1 is written as

$$x_{ijk} \approx \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr} \text{ for } i = 1, \ldots, I, j = 1, \ldots, J, k = 1, \ldots, K. \tag{1.2}$$

This is illustrated in Figure: 1.2. When Eq. 1.1 is perfect equals to then R = ***Rank of the Tensor X***

The rank of a tensor $\boldsymbol{X}$, denoted $\text{rank}(\boldsymbol{X})$, is defined as the smallest number of rank-one tensors (see section 2.1) that generate $\mathcal{X}$ as their sum. In other words, this is the smallest number of components in an exact CP decomposition, where "exact" means that there is equality in 1.1. An exact CP decomposition with $R = \text{rank}(\mathcal{X})$ components is called the rank decomposition.

The factor matrices refer to the combination of the vectors from the rank-one components, i.e., $\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_R \end{bmatrix}$ and likewise for $\mathbf{B}$ and $\mathbf{C}$. Using these definitions, 1.1 may be written in matricized form:

$$\begin{aligned} \mathbf{X}_{(1)} &\approx \mathbf{A}(\mathbf{B} \odot \mathbf{C})^\top, \\ \mathbf{X}_{(2)} &\approx \mathbf{B}(\mathbf{A} \odot \mathbf{C})^\top, \\ \mathbf{X}_{(3)} &\approx \mathbf{C}(\mathbf{A} \odot \mathbf{B})^\top. \end{aligned} \tag{1.3}$$

$\odot$ denotes the Khatri-Rao product. The three-way model is sometimes written in terms of the frontal slices of $\mathcal{X}$:

$$\mathbf{X}_k \approx \mathbf{A}\mathbf{D}^{(k)}\mathbf{B}^\top, \text{ where } \mathbf{D}^{(k)} \equiv \text{diag}\left(\mathbf{c}_{k:}\right) \text{ for } k = 1, \ldots, K.$$

Analogous equations can be written for the horizontal and lateral slices. In general, though, slicewise expressions do not easily extend beyond three dimensions. The CP model can be concisely expressed as

$$\mathcal{X} \approx \mathbf{A}, \mathbf{B}, \mathbf{C} \equiv \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r.$$

It is often useful to assume that the columns of $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$ are normalized to length one with the weights absorbed into the vector $\boldsymbol{\lambda} \in \mathbb{R}^R$ so that

$$\mathcal{X} \approx \boldsymbol{\lambda}; \mathbf{A}, \mathbf{B}, \mathbf{C} \equiv \sum_{r=1}^{R} \lambda_r \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r.$$

We have focused on the three-way case because it is widely applicable and sufficient for many needs. For a general Nth-order tensor, $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, the CP decomposition is

$$X \approx \boldsymbol{\lambda}; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)} \equiv \sum_{r=1}^{R} \lambda_r \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(N)},$$

where $\boldsymbol{\lambda} \in \mathbb{R}^R$ and $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$ for $n = 1, \ldots, N$. In this case, the mode- n matricized version is given by

$$\mathbf{X}_{(n)} \approx \mathbf{A}^{(n)} \boldsymbol{\Lambda} \left(\mathbf{A}^{(1)} \odot \cdots \odot \mathbf{A}^{(n-1)} \odot \mathbf{A}^{(n+1)} \odot \cdots \odot \mathbf{A}^{(N)} \right)^\top,$$

where $\boldsymbol{\Lambda} = \text{diag}(\boldsymbol{\lambda})$.

1.5 Low-Rank Approximations

For matrices, Eckart and Young showed that a best rank- k approximation is given by the leading k factors of the SVD. In other words, let R be the rank of a matrix $\mathbf{A}$ and assume its SVD is given by

$$\mathbf{A} = \sum_{r=1}^{R} \sigma_r \mathbf{u}_r \circ \mathbf{v}_r \quad \text{with } \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_R > 0.$$

Then a rank- k approximation that minimizes $\|\mathbf{A} - \mathbf{B}\|$ is given by

$$\mathbf{B} = \sum_{r=1}^{k} \sigma_r \mathbf{u}_r \circ \mathbf{v}_r.$$

This type of result does not hold true for higher-order tensors. For instance, consider a third-order tensor of rank R with the following CP decomposition:

$$\boldsymbol{X} = \sum_{r=1}^{R} \lambda_r \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r.$$

Ideally, summing k of the factors would yield a best rank-k approximation, but that is not the case. It is possible that the best rank-k approximation may not even exist. The problem is referred to as one of degeneracy. A tensor is degenerate if it may be approximated arbitrarily well by a factorization of lower rank. Let $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ be a rank-three tensor defined by

$$\mathcal{X} = \mathbf{a}_1 \circ \mathbf{b}_1 \circ \mathbf{c}_2 + \mathbf{a}_1 \circ \mathbf{b}_2 \circ \mathbf{c}_1 + \mathbf{a}_2 \circ \mathbf{b}_1 \circ \mathbf{c}_1,$$

where $\mathbf{A} \in \mathbb{R}^{I \times 2}, \mathbf{B} \in \mathbb{R}^{J \times 2}$, and $\mathbf{C} \in \mathbb{R}^{K \times 2}$, and each has linearly independent columns. This tensor can be approximated arbitrarily closely by a rank-two tensor of the following form:

$$\boldsymbol{y} = \alpha \left(\mathbf{a}_1 + \frac{1}{\alpha} \mathbf{a}_2 \right) \circ \left(\mathbf{b}_1 + \frac{1}{\alpha} \mathbf{b}_2 \right) \circ \left(\mathbf{c}_1 + \frac{1}{\alpha} \mathbf{c}_2 \right) - \alpha \mathbf{a}_1 \circ \mathbf{b}_1 \circ \mathbf{c}_1.$$

Chapter 2

ALS: Alternative Least Square

This is nothing but a special type of **Matrix Least Square/ Ordinary Least Square (OLS)** method where while finding unknowns through least square we fix some variables, and then we fix other variables and vary previous ones. In such alternative way, we apply least square. So, first we will discuss ordinary least square problem followed by matrix least square. Then we will go to the Alternative least square.

2.1 Least squares problem

Suppose that the $m \times n$ matrix A is tall, so the system of linear equations $Ax = b$, where b is an m-vector, is over-determined, i.e., there are more equations (m) than variables to choose (n). These equations have a solution only if b is a linear combination of the columns of A.

For most choices of b, however, there is no n-vector x for which $Ax = b$. As a compromise, we seek an x for which $r = Ax - b$, which we call the residual (for the equations $Ax = b$), is as small as possible. This suggests that we should choose x so as to minimize the norm of the residual, $\|Ax - b\|$. If we find an x for which the residual vector is small, we have $Ax \approx b$, i.e., x almost satisfies the linear equations $Ax = b$. (Some authors define the residual as $b - Ax$, which will not affect us since $\|Ax - b\| = \|b - Ax\|$.

Minimizing the norm of the residual and its square are the same, so we can just as well minimize

$$\|Ax - b\|^2 = \|r\|^2 = r_1^2 + \cdots + r_m^2,$$

the sum of squares of the residuals. The problem of finding an n-vector $\hat{x}$ that minimizes $\|Ax - b\|^2$, over all possible choices of x, is called the least squares problem. It is denoted using the notation

$$\text{minimize } \|Ax - b\|^2 \tag{2.1}$$

where we should specify that the variable is x (meaning that we should choose x). The matrix A and the vector b are called the data for the problem (2.1), which means that they are given to us when we are asked to choose x. The quantity to be minimized, $\|Ax - b\|^2$, is called the objective function (or just objective) of the least squares problem (2.1).

2.2 Solution of Least Square

In this section we find the solution of the least squares problem using some basic results from calculus. We know that any minimizer $\hat{x}$ of the function $f(x) = \|Ax - b\|^2$ must satisfy

$$\frac{\partial f}{\partial x_i}(\hat{x}) = 0, \quad i = 1, \ldots, n$$

which we can express as the vector equation

$$\nabla f(\hat{x}) = 0$$

where $\nabla f(\hat{x})$ is the gradient of f evaluated at $\hat{x}$. The gradient can be expressed in matrix form as

$$\nabla f(x) = 2A^T(Ax - b) \tag{2.2}$$

This formula can be derived from the chain rule, and the gradient of the sum of squares function. For completeness, we will derive the formula (2.2) from scratch here. Writing the least squares objective out as a sum, we get

$$f(x) = \|Ax - b\|^2 = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} A_{ij}x_j - b_i \right)^2 .$$

To find $\nabla f(x)_k$ we take the partial derivative of f with respect to x_k. Differentiating the sum term by term, we get

$$\begin{aligned}
\nabla f(x)_k &= \frac{\partial f}{\partial x_k}(x) \\
&= \sum_{i=1}^{m} 2 \left(\sum_{j=1}^{n} A_{ij}x_j - b_i \right) (A_{ik}) \\
&= \sum_{i=1}^{m} 2 \left(A^T\right)_{ki} (Ax - b)_i \\
&= \left(2A^T(Ax - b)\right)_k
\end{aligned}$$

This is our formula (2.2), written out in terms of its components.

Now we continue the derivation of the solution of the least squares problem. Any minimizer $\hat{x}$ of $\|Ax-b\|^2$ must satisfy

$$\nabla f(\hat{x}) = 2A^T(A\hat{x} - b) = 0,$$

which can be written as

$$A^T A\hat{x} = A^T b \tag{2.3}$$

These equations are called the normal equations. The coefficient matrix $A^T A$ is the Gram matrix associated with A; its entries are inner products of columns of A.

Our assumption that the columns of A are linearly independent implies that the Gram matrix $A^T A$ is invertible. This implies that

$$\hat{x} = \left(A^T A\right)^{-1} A^T b \tag{2.4}$$

is the only solution of the normal equations (2.3). So this must be the unique solution of the least squares problem (2.1).

We have already encountered the matrix $\left(A^T A\right)^{-1} A^T$ that appears in (2.4): It is the pseudo-inverse of the matrix A. So we can write the solution of the least squares problem in the simple form

$$\hat{x} = A^{\dagger} b \tag{2.5}$$

2.3 Matrix Least Square

A simple extension of the least squares problem is to choose the $n \times k$ matrix X so as to minimize $\|AX - B\|^2$. Here A is an $m \times n$ matrix and B is an $m \times k$ matrix, and the norm is the matrix norm. This is sometimes called the matrix least squares problem. When $k = 1, x$ and b are vectors, and the matrix least squares problem reduces to the usual least squares problem.

The matrix least squares problem is in fact nothing but a set of k ordinary least squares problems. To see this, we note that

$$\|AX - B\|^2 = \|Ax_1 - b_1\|^2 + \cdots + \|Ax_k - b_k\|^2$$

where x_j is the j th column of X and b_j is the j th column of B. (Here we use the property that the square of the matrix norm is the sum of the squared norms of the columns of the matrix.) So the objective is a sum of k terms, with each term depending on only one column of X. It follows that we can choose the columns x_j independently, each one by minimizing its associated term $\|Ax_j - b_j\|^2$. Assuming that A has linearly independent columns, the solution is $\hat{x}_j = A^\dagger b_j$. The solution of the matrix least squares problem is therefore

$$\begin{aligned}\hat{X} &= \begin{bmatrix} \hat{x}_1 & \cdots & \hat{x}_k \end{bmatrix} \\ &= \begin{bmatrix} A^\dagger b_1 & \cdots & A^\dagger b_k \end{bmatrix} \\ &= A^\dagger \begin{bmatrix} b_1 & \cdots & b_k \end{bmatrix} \\ &= A^\dagger B.\end{aligned}$$

The very simple solution

$$\hat{X} = A^\dagger B \tag{2.6}$$

the matrix least squares problem agrees with the solution of the ordinary least squares problem when $k = 1$ (as it must). Many software packages for linear algebra use the backslash operator $A\backslash B$ to denote $A^\dagger B$, but this is not standard mathematical notation.

2.4 ALS Algorithm to find Factors of CP Decomposition

Mostly we will be dealing with 3-way Tensors. If it is a 32x32 pixel color image, then the shape of the Image tensor is (32, 32, 3).

For this 3-way data (say, X), we have 3 types of matricization techniques and thus we will get 3 matrix $X_{(1)}, X_{(2)}$ and $X_{(3)}$ and suppose our approximated Tensor is going to be A (we will start with a random initial A)

$$\begin{aligned}\mathbf{X}_{(1)} &\approx \mathbf{A_1}(\mathbf{A_2} \odot \mathbf{A_3})^\top, \\ \mathbf{X}_{(2)} &\approx \mathbf{A_2}(\mathbf{A_1} \odot \mathbf{A_3})^\top, \\ \mathbf{X}_{(3)} &\approx \mathbf{A_3}(\mathbf{A_1} \odot \mathbf{A_2})^\top.\end{aligned}$$

Here, A_1, A_2, A_3 are the new symbols of A, B, C in the eq.(1.3) respectively.

Now, we have 3 Least Square Minimization Problem,

$$\begin{aligned}&\text{Min } \|\mathbf{A_1}(\mathbf{A_2} \odot \mathbf{A_3})^\top - \mathbf{X}_{(1)}\|, \\ &\text{Min } \|\mathbf{A_2}(\mathbf{A_1} \odot \mathbf{A_3})^\top - \mathbf{X}_{(2)}\|, \\ &\text{Min } \|\mathbf{A_3}(\mathbf{A_1} \odot \mathbf{A_2})^\top - \mathbf{X}_{(3)}\|.\end{aligned} \tag{2.7}$$

The first one of Eq (2.7) can be written as,

$$\text{Min } \|(\mathbf{A_2} \odot \mathbf{A_3})\mathbf{A_1}^\top - \mathbf{X}_{(1)}^\top\|$$

Here, we will fix the co-efficient of $A_1^\top$ i.e. $(\mathbf{A_2} \odot \mathbf{A_3}) = \mathbf{B}$ (say).

$$\text{Min } \|\mathbf{B}\mathbf{A_1}^\top - \mathbf{X}_{(1)}^\top\|$$

So, here A_1 is the unknown and the Matrix Least Square solution is,

$$\hat{\mathbf{A}}_1^\top = \left(\mathbf{B}^\top\mathbf{B}\right)^{-1}\mathbf{B}^\top\mathbf{X}_{(1)}^\top$$

i.e.

$$\hat{\mathbf{A}}_1 = \mathbf{X}_{(1)}\mathbf{B}\left(\mathbf{B}^\top\mathbf{B}\right)^{-1} \tag{2.8}$$

Similar formula can be derived for the other two Min. problems. From (2.8), we can get modified A_1, then we will fix A_1 and A_3 and get modified A_2 from the corresponding formula. And at last we will do the same to get modified A_3. Thus, one iteration will be completed and we will get modified A_1, A_2, A_3.

2.5 Python Code for CP Decomposition and Image Compression

Used Libraries

[1]:
```
import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt
import tensorly as tl
from tensorflow.keras import datasets
```

2.5.1 Input Section

[2]:
```
n = int(input("Enter the order of the Tensor: "))

print("Enter the shape of the tensor:")
shape = []
for i in range(n):
    shape.append(int(input()))
```

```
Enter the order of the Tensor: 2
Enter the shape of the tensor:
28
28
```

MNIST Image Input

Taking image input from MNIST dataset in python. This is the collection of 28x28 pixel images (black & white)

[3]:
```
from keras.datasets import mnist
```

[4]:
```
(img_train, _), (img_test, _) = mnist.load_data()
```

[5]:
```
plt.figure(figsize = (3,3))
plt.imshow(img_train[22], cmap = 'binary')
plt.show()
```

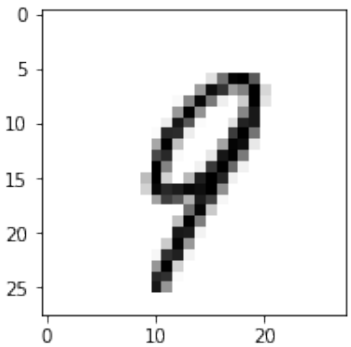

Figure 2.1: MNIST Data: Handwritten image of 9

[6]:
```
val = img_train[22]
val.shape
```

[6]:
```
(28, 28)
```

Setting up the other paramaters:

- eps - error limit
- k - order of approximation
- iter - Max. no. of iteration

[7]:
```
eps = float(input('Enter the value of epsilon: '))

print('\nWe are going to find K-th rank approximation: ')
k = int(input('Enter the value of k: '))

iter = int(input('\nEnter max. number of iteration: '))
```

```
Enter the value of epsilon: 0.0001

We are going to find K-th rank approximation:
Enter the value of k: 8

Enter max. number of iteration: 10000
```

[8]:
```
k, eps, iter
```

[8]:
```
(8, 0.0001, 10000)
```

Setting Initial Factor Matrices

[9]:
```
A = []
for i in range(n):
    A.append(np.random.normal(size = (shape[i],k), loc = 0, scale = 1 / k))
```

Finding the m-modes matrices

[10]:
```
val_m = []

for i in range(n):
    val_m.append(tl.unfold(val,i))

# val_m
```

2.5.2 ALS: Alternating Least Square

Matrix Method for Approximation

[11]:
```
for a in range(iter):
    # iteration

    for k_mode in range(n):

        temp = A.copy()
        temp.pop(k_mode)

        B = temp[0]
```

```
        for ele in temp[1:]:
            B = linalg.khatri_rao(B, ele)

        A[k_mode] = np.matmul(val_m[k_mode], np.matmul(B, linalg.inv(np.matmul(B.T,␣
↪B))))

        err = tl.norm(val_m[k_mode] - np.matmul(A[k_mode], B.T))
        if err < eps:
            break
    if err < eps:
        break
```

Approximated Tensor

[12]:
```
result = []
for i in range(n):
    value = []
    for col in range(A[i].shape[1]):
        value.append(A[i][:,col])

    result.append(value)
# print(result)
```

[13]:
```
final_val = 0
for j in range(k):
    prod_val = []
    for i in range(n):
        prod_val.append(result[i][j])
    final_val += tl.tenalg.outer(prod_val)

# display(final_val, val)
```

[14]:
```
tl.norm(final_val), tl.norm(val), tl.norm(final_val - val)
```

[14]:
```
(1772.0521072339723, 103.82196299434914, 122.71238424735284)
```

Relative Error!

We are taking the Frobenius norm of tensor to get the idea of how good we can approximate the actual tensor by the iterated one by using the difference tensor.

[15]:
```
tl.norm(final_val - val) / tl.norm(val)
```

[15]:
```
1.1819501453081935
```

2.6 Image Data Compression

I have applied this algorithm on 2 image data. First one is hand wriiten data from MNIST Dataset, second one is a png file of our IIT Kgp logo. Other than this, I also worked on several types of data including CIFAR-10 dataset.

Output 1

[16]:
```
import cv2
```

[17]:
```
cv2.imwrite("img.jpg", final_val)
```

[17]:
```
True
```

[18]:
```
cv2.imwrite('ac_img.jpg', val)
```

[18]:
```
True
```

[19]:
```
final_val = np.floor(final_val).astype(int)

fig = plt.figure(figsize = (4,4))

fig.add_subplot(1,2,2)
plt.imshow(val, cmap = 'binary')
plt.axis('off')
plt.title('Actual')

fig.add_subplot(1,2,1)
plt.imshow(final_val, cmap = 'binary')
plt.axis('off')
plt.title('Compressed')

plt.show()
```

Figure 2.2: Comparison between Compressed and Actual Image

No. of Pixels Compressed

[20]:
```
print("Actual:       ", end = '')
print('   28 * 28   =', 28 * 28)

print("Compressed:", end = ' ')
print('(28 + 28) * 8 =', (28 + 28) * 8)
```

```
Actual:         28 * 28   = 784
Compressed: (28 + 28) * 8 = 448
```

Instead of using **784 bytes** data, we can replace it with **448 bytes** data without losing too much photo quality. Thus, we compressed the given photo from 784 to 448 bytes.

Output 2

In this example, input data is a color photo of shape = (896, 800, 3)

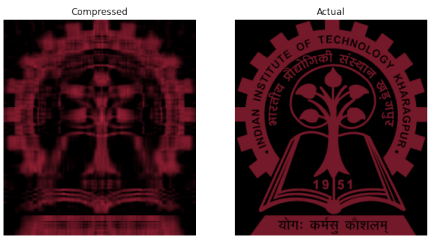

Figure 2.3: IIT Kgp Logo

So, Size of Actual Data = 896 * 800 * 3 = **2150400 bytes**.

Size of Approx. Data = (896 + 800 + 3) * 8 = **13592 bytes**. So, the difference is too much.
While transferring this large type of file through any network we need this type of compression to reduce the time while maintaining the transferring time. Though in this example we have taken a lower rank approximation that's why we have got pixelated image. Otherwise, there are lot of good methods which are too much popular in our daily life.

2.7 Computational Cost

Parameters we used are as follows:

iter = no. of iteration, order of tensor = n (= 3 for color image)

shape of tensor = $(i_1, i_2, \cdots, i_n)$, m = $\max(i_1, i_2, \cdots, i_n)$, Rank approximation = k

Considering the operations and loops in the code,

- For each iteration, 2 nested for loop is working, in a range of n and m.
- For matrix inversion of $B^\top B$, it is $O(k^3)$ (Gauss-Jordan Elimination)

So, Computational Complexity = $O(nmk^3)$

References

[1] Kolda, Tamara G., and Brett W. Bader. *Tensor decompositions and applications* SIAM review 51.3 (2009): 455-500.

[2] Stephen Boyd, Lieven Vandenberghe. *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares* Cambridge University Press 2018, pp. 225 - 239

[3] J. B. Kruskal, *Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics*, Linear Algebra Appl., 18 (1977), pp. 95–138.

[4] J. B. Kruskal, *Rank, decomposition, and uniqueness for 3-way and N-way arrays, in Multiway Data Analysis*, R. Coppi and S. Bolasco, eds., North-Holland, Amsterdam, 1989, pp. 7–18.