Université de Limoges
Faculté des Sciences et Techniques
Département de Mathématiques

October 2023

MASTER DE MATHÉMATIQUES ET APPLICATIONS, PARCOURS ACSYON

APPLIED LINEAR ALGEBRA

Homework 1 .

Cleque Marlain MBOULOU

In the context of tensor decomposition, determining the rank of a tensor is a challenging problem, as there is no finite algorithm to compute it directly. This leads us to the key challenge in computing a CP decomposition: how to choose the number of rank-one components.

1- Algorithm for CP: the alternating least squares (ALS)

Assuming that the number of components (R) is fixed, there are several algorithms available for computing a CP decomposition. One of the most widely used algorithms is the Alternating Least Squares (ALS) method, originally proposed by Carroll and Chang and later by Harshman. While we present the method here for the third-order tensor, it can be extended to N-way tensors.

Let $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ be a third-order tensor that we want to decompose into R components. The objective is to find a CP decomposition $\hat{\mathcal{X}}$ that minimizes the Frobenius norm between $\mathcal{X}$ and $\hat{\mathcal{X}}$, as defined by:

$$\min_{\hat{\mathcal{X}}_k} \|\mathcal{X} - \hat{\mathcal{X}}_k\|_F \quad \text{with} \quad \hat{\mathcal{X}}_k = \sum_{r=1}^{R} \lambda_r a_r \circ b_r \circ c_r,$$

where $\circ$ denotes the outer product. The alternating least squares approach fixes B and C to solve for A, then fixes A and C to solve for B, and continues in this manner.

Having fixed all but one matrix, the problem reduces to a linear least squares problem. For example, suppose that B and C are fixed. Then we can rewrite the above minimization problem in matrix form as

$$\min_{\hat{A}} \|\mathcal{X}^{(1)} - \hat{A}(C \circ B)^T\|_F,$$

where $\hat{A} = A \cdot \text{diag}(\lambda)$. The optimal solution is then given by

$$\hat{A} = \mathcal{X}^{(1)}(C \circ B)(C^T C * B^T B)^{\dagger}.$$

- **Function** `matricialisation`**:** The `matricialisation(tensor, I,mode)` function takes a tensor and a list `I` as input and performs tensor matricialization, returning a list of matrices representing the matricialization of each tensor mode.

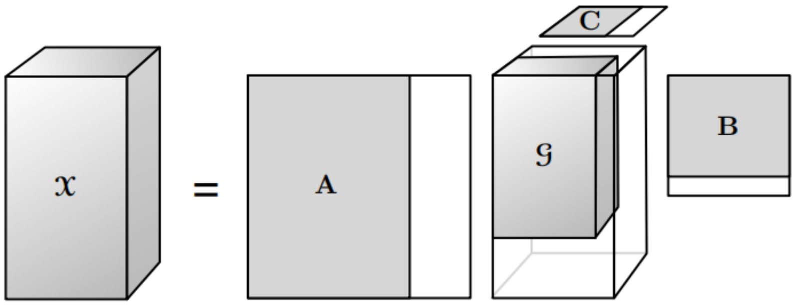
Algorithm 1: CP-ALS Algorithm

```
   Data: Tensor $\mathcal{X}$ of size $I_1 \times I_2 \times \ldots \times I_N$, Number of components $R$
   Result: Factor matrices $\Lambda$, $A^{(1)}$, $A^{(2)}$, …, $A^{(N)}$
1  Initialize random matrices $A^{(1)}, A^{(2)}, \ldots, A^{(N)}$ of sizes $I_1 \times R, I_2 \times R, \ldots, I_N \times R$;
2  Set maximum iterations $max\_iterations$;
3  Set tolerance for convergence $tolerance$;
4  repeat
5      for $n = 1$ to $N$ do
6          $V \leftarrow A^{(1)}T(A^{(1)})^T \ldots A^{(n-1)}T(A^{(n-1)})^T A^{(n+1)}T(A^{(n+1)})^T \ldots A^{(N)}T(A^{(N)})^T$;
7          $A^{(n)} \leftarrow \mathcal{X}^{(n)}(A^{(N)} \otimes \ldots \otimes A^{(n+1)} \otimes A^{(n-1)} \otimes \ldots \otimes A^{(1)})V^{\dagger}$;
8          Normalize columns of $A^{(n)}$ and store norms as $\Lambda$;
9      Calculate the fit and check for convergence;
10 until fit ceases to improve or maximum iterations exhausted;
11 return $\Lambda$, $A^{(1)}$, $A^{(2)}$, …, $A^{(N)}$
```

2- Higher-Order Singular Value Decomposition (HOSVD)

The Tucker decomposition can be described as a variant of higher-order principal component analysis, where a tensor is broken down into a core tensor that is then multiplied (or transformed) by a matrix for each mode. Thus, in the three-way case where $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ as follows:

$$\mathcal{X} = \mathcal{G} \times_1 A \times_2 B \times_3 C \tag{1}$$



Where: $A \in \mathbb{R}^{I_1 \times R_1}, B \in \mathbb{R}^{I_2 \times R_2}, C \in \mathbb{R}^{I_3 \times R_3}$ and $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ is the core tensor. And $R_i = rank(\mathcal{X}_{(i)})$, the rank of the matricialization of $\mathcal{X}$ mode i

$$\mathcal{G} = \mathcal{X} \times_1 A^T \times_2 B^T \times_3 C^T \iff \begin{cases} \mathcal{G}_{(1)} = A^T \mathcal{X}_{(1)}(C \otimes B) \\ \mathcal{G}_{(2)} = C^T \mathcal{X}_{(2)}(C \otimes A) \\ \mathcal{G}_{(3)} = C^T \mathcal{X}_{(3)}(B \otimes A) \end{cases} \tag{2}$$

In general case, when $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2, \ldots, I_N}$, the Tucker decomposition is given by:

$$\mathcal{X} = \mathcal{G} \times_1 A^{(1)T} \times_2 A^{(1)T} \times_3 \ldots \times_N A^{(N)T} \tag{3}$$

The matrices $A^{(1)T}$, $A^{(2)T}$, ..., $A^{(N)T}$ are transposes of the matrices $A^{(1)}$, $A^{(2)}$, ..., $A^{(N)}$ computed during the initialization step of the **HOSVD** procedure. These matrices $A^{(n)}$ represent the dominant left singular vectors of mode n in the tensor $\mathcal{X}$.

The characteristics of these matrices A can be summarized as follows:

1. **Factor Matrices**: The matrices $A^{(1)}$, $A^{(2)}$, ..., $A^{(N)}$ are often referred to as "factor matrices." They capture the fundamental structures of tensor $\mathcal{X}$ in their respective modes, playing a crucial role in the Tucker decomposition.
2. **Reduced Rank:** Typically, these matrices A have reduced rank, determined by the input values R_1, R_2, ..., R_N provided to the **HOSVD** procedure. The rank signifies the number of fundamental components (or dimensions) extracted from each mode of the tensor.
3. **Orthonormality :** In many applications, matrices A are computed to be orthonormal, with columns being orthogonal (perpendicular to each other) and having unit norm. This property simplifies the decomposition and enhances the interpretation of components.
4. **Dimension Reduction**: Matrices A contribute to reducing the dimension of the problem by extracting the most significant components from each mode. This results in a more compact representation of tensor $\mathcal{X}$ while retaining essential information.

When computing the core tensor $\mathcal{G}$ through the contraction of $\mathcal{X}$ with the transposed matrices $A^{(1)T}$, $A^{(2)T}$, ..., $A^{(N)T}$, the characteristics of matrices A are utilized to express tensor $\mathcal{X}$ in terms of its essential components, represented by $\mathcal{G}$. This representation serves as the foundation of the Tucker decomposition, enabling efficient data compression and multidimensional data analysis.

Algorithm 2: HOSVD($\mathcal{X}, R_1, R_2, ..., R_N$)

```
1 for $n = 1$ to $N$ do
2     $A^{(n)} \leftarrow R_n$ leading left singular vectors of $\mathcal{X}_{(n)}$.
3 $\mathcal{G} \leftarrow \mathcal{X} \times_1 A^{(1)T} \times_2 A^{(2)T} \dots \times_N A^{(N)T}$
4 Renvoyez $\mathcal{G}$ et $A^{(1)}, A^{(2)}, \dots, A^{(N)}$.
```

To compute $\mathcal{G}$, we can use equation (2). However, when the tensor's order is greater than 3, computing $\mathcal{G}$ may require a method of tensorization and matricialization of the tensor obtained through modulo i product ($\times_i$). This is why I have developed two functions: the first one to obtain the matricialization of each tensor, and the other one to obtain the initial tensor from its matricialization.

3-Higher-Order Orthogonal Iteration (HOOI)

Higher-Order Orthogonal Iteration (HOOI) is an iterative algorithm used for computing low-rank approximations to tensors. Given a tensor $\mathcal{X}$ of dimensions $I_1 \times I_2 \times \dots \times I_N$ and a set of integers $R_1, R_2, \dots, R_N$ where $1 \leq R_n \leq I_n$ for $n = 1, \dots, N$, the problem is to find a set of matrices $A^{(n)} \in \mathbb{R}^{I_n \times R_n}$, each with orthogonal columns, and a core tensor $\mathcal{G}$ of dimensions $R_1 \times \dots \times R_N$. The goal is to minimize the optimization problem:

$$\min_{A^{(1)}, A^{(2)}, \dots, A^{(T)}, \mathcal{G}} \|\mathcal{X} - \mathcal{G} \times_1 A^{(1)}) \times_2 A^{(2)} \dots \times_T A^{(N)}\|_F$$

The optimal core tensor $\mathcal{G}$ is calculated as:

$$\mathcal{G} = \mathcal{X} \times_1 A^{(1)T}) \times_2 A^{(2)T} \dots \times_T A^{(N)T}$$

Higher-order Orthogonal Iteration (HOOI) is an alternating least squares (ALS) approach to solving this problem to improve HOSVD. It solves a sequence of restricted optimization problems by updating the matrices $A^{(n)}$, where optimization is done over the n-th matrix while using the latest available values of the other $A^{(i)}$'s. For simplicity, the HOOI algorithm is stated for 3rd order tensors, but it can be extended to higher-order tensors.

Algorithm 3: HOOI Algorithm

```
1 **Input:** $X, R_1, R_2, \dots, R_N$
2 **Initialization:** Initialize $A^{(n)} \in \mathbb{R}^{I_n \times R_n}$ for $n = 1, 2, \dots, N$ using HOSVD
3 **Repeat**
      **For** $n = 1, 2, \dots, N$ **do**
          $\mathcal{Y} \leftarrow \mathcal{X} \times_1 A^{(1)T} \times_2 A^{(2)T} \dots \times_n A^{(n-1)T} \times_{n+1} A^{(n+1)T} \dots \times_N A^{(N)T}$
          $A^{(n)} \leftarrow$ Leading left singular vectors of $\mathcal{Y}^{(n)}$ with rank $R_n$
      **End for**
      **Until** fit ceases to improve or maximum iterations exhausted
      $\mathcal{G} \leftarrow \mathcal{X} \times_1 A^{(1)T} \times_2 A^{(2)T} \dots \times_N A^{(N)T}$
  **Return** $\mathcal{G}, A^{(1)}, A^{(2)}, \dots, A^{(N)}$
```

Codes

CP_ALS

```
function matrice_mode = mat_tenseur(tenseur, mode)
    % Check th dimensions
    dimensions = size(tenseur);
    if mode < 1 || mode > length(dimensions)
        error('incorrect␣mode.');
    end

    % Swap dimensions to put the chosen mode in first position
    tenseur_permute = permute(tenseur, [mode, 1:mode-1, mode+1:length(dimensions)]);

    % Matrixing the tensor
    matrice_mode = reshape(tenseur_permute, dimensions(mode), []);

end

function [X] = init(i,r)
  X = randi(5,i, r);
endfunction
```

```
function [A,B,C] = cp_als3(tensor,R,ITER = 1000)
  %Matrixing
  X_1 = mat_tenseur(tensor,1);
  X_2 = mat_tenseur(tensor,2);
  X_3 = mat_tenseur(tensor,3);
  t = 0;
  I = size(tensor);
  %Initialization
  A = init(I(1),R);%randi(5,I(1),R);
  B = init(I(2),R);%randi(5,I(2),R);
  C = init(I(3),R);%randi(5,I(3),R);
  while t<ITER
    t+=1;
    A = X_1*(khr_product(C,B))*pinv((C'*C).*(B'*B));
    B = X_2*(khr_product(C,A))*pinv((C'*C).*(A'*A));
    C = X_3*(khr_product(B,A))*pinv((B'*B).*(A'*A));

    U = A*khr_product(C, B)';
    normFr(X_1-U)
  endwhile
endfunction
```

HOSVD

```
# In this function the goal is to built a tensor given the matrixing
# mode.
# The idea is to do inverse instuctions of matrixing
# 1: Reshape
# 2 : Do the permutation of consided mode whith the position 1

function [tensor] = tenseur_mat(X,siz,mode)
  N = length(siz);
  t = 1:N;
  l = t;
  t(1) = mode;
  t(mode) = 1;
  new_size = [];
  if mode==N
    tensor = permute(reshape(X',siz), l);
  else
    for i = 1: N
      new_size = [new_size siz(t(i))];
    endfor
    tensor = reshape(X,new_size);
    tensor = permute(tensor, t);

  end

endfunction
```

```
function norme = check(tensor,G,A)
  I1 = size(tensor);
  I2 = size(G);
  N = length(I1);
  G1 = mat_tenseur(G,1);
  x = G1;
  siz = I2;
  for i = 1:N-1
    siz(i) = I1(i);
    x = mat_tenseur(tenseur_mat(A{i}*x,siz ,i),i+1);
  endfor
  norme = normFr(mat_tenseur(tensor,N)-A{N}*x);
endfunction
```

HOOI

```

function [G,Q] = HOOI (tensor, R,ITER=100)
  [G1 A1] = HOSVD(tensor,R);

  A = A1{1};
  B = A1{2};
  C = A1{3};
  %tenseur_mat(A*G1*(kron(C,B))',size(tensor),1)
  I = size(tensor);
  N = 3;
  X = [];
    %Compute all matricialization of tensor

  X_1= mat_tenseur(tensor,1);
  X_2= mat_tenseur(tensor,2);
  X_3= mat_tenseur(tensor,3);
  tenseur_mat(X_3,size(tensor),3);


  for it=1:ITER

    %We take the matricialization of the first element the list l whithout i
    K_1 = C'*mat_tenseur(tenseur_mat(B'*X_2,[I(1) R(2) I(3)],2),3);
    Y_1 = tenseur_mat(K_1,[I(1) R(2) R(3)],3);
    [U1, D1, V1] = svd(mat_tenseur(Y_1,1));
    A = U1(:,1:R(1));
    K_2 = C'*mat_tenseur(tenseur_mat(A'*X_1,[R(1) I(2) I(3)],1),3);
    Y_2 = tenseur_mat(K_2,[R(1) I(2) R(3)],3);
    [U2, D2, V2] = svd(mat_tenseur(Y_2,2));
    B = U2(:,1:R(2));

    K_3 = B'*mat_tenseur(tenseur_mat(A'*X_1,[R(1) I(2) I(3)],1),2);
    Y_3 = tenseur_mat(K_3,[R(1) R(2) I(3)],2);
    [U3, D3, V3] = svd(mat_tenseur(Y_3,3));
    C = U3(:,1:R(3));
    %G = A'*X_1*(kron(C,B));
    %normFr(X_1 - A*G*(kron(C,B))')
  endfor
  %Like HOSVD
  G = A'*X_1*(kron(C,B));
  G = tenseur_mat(G,R,1);
  Q{1} = A;
  Q{2} = B;
  Q{3} = C;

  %tenseur_mat(A*G*(kron(C,B))',size(tensor),1);

endfunction
```

Examples

```

function result  = Examples ()
  max_size = 10;
  min_size = 2;
  %l = randi([min_size,max_size],1,3) % Creat a list of 3 integers between 2 and 10
  %tensor = randi(20,l);
  l = [3 4 2];
  tensor =[1 2 3 4 ;5 6 7 8; 9 10 11 12];
  tensor(:,:,2) = [13 14 15 16; 17 18 19 20;21 22 23 24];

  ranks = [];
  for i=1:length(l)
    rank(mat_tenseur(tensor,i));
    ranks = [ranks rank(mat_tenseur(tensor,i))];
  endfor
    disp("CP_ALS␣result");
   [A,B,C] =cp_als3(tensor,ranks)
   disp("HOSVD␣result\n");
   [G, A] = HOSVD(tensor,ranks)
   result = check(tensor,G, A)
   disp("HOOI␣result\n");
   [G, A] = HOOI(tensor,ranks)
   result = check(tensor,G, A);
endfunction


>> Examples

CP_ALS result
A =

   2.36577  -0.77899
   2.67366  -0.66116
   2.98154  -0.54332

B =

   1.9837   6.5295
   2.0435   6.2578
   2.1033   5.9861
   2.1631   5.7144

C =

   3.1718   2.7381
   4.4871   1.5749

HOSVD result

G =

ans(:,:,1) =

  -69.627139    0.091390
   -0.033009   -1.045320

ans(:,:,2) =

   0.020124   2.211762
  -6.722591  -0.934788

A =
{
  [1,1] =

    -0.41726   0.81193
    -0.56467   0.12036
    -0.71207  -0.57122

  [1,2] =

    -0.45266  -0.70363
    -0.48343  -0.25748
    -0.51420   0.18867
    -0.54497   0.63483

  [1,3] =

    -0.35334  -0.93549
    -0.93549   0.35334

}

norm(X_(1)-X^_(1))
result =    2.6291e-14
HOOI result
G =

ans(:,:,1) =

  -69.627139    0.091390
    0.033009    1.045320

ans(:,:,2) =

   0.020124   2.211762
   6.722591   0.934788

A =
{
  [1,1] =

    -0.41726  -0.81193
    -0.56467  -0.12036
    -0.71207   0.57122

  [1,2] =

    -0.45266  -0.70363
    -0.48343  -0.25748
    -0.51420   0.18867
    -0.54497   0.63483

  [1,3] =

    -0.35334  -0.93549
    -0.93549   0.35334

}

norm(X_(1)-X^_(1))
ans =    3.1102e-14
>>
```

Conclusion

The Tucker decomposition is a versatile technique with various applications in data analysis, compression, and dimensionality reduction. It allows us to uncover the underlying structure of multi-dimensional data and can lead to significant storage savings when applied to real-world problems.