M24-DS-01 HDDA: Final Exam	Marks obtained ↓
Date: 19.12.2024, Total questions: **4** Total points: **70**	
Name:	Time: 26 hrs

Question:	1	2	3	4	Total
Points:	25	25	10	10	70
Score:					

Instructions:

1. **You receive now the questions for the final exam: you have 26 hours to answer them, then you are requested to join the Professor in room 321, December 19th to present and defend your answers during 15-20 minutes. The running order has been provided in moodle.**

2. **Question 4 is only for students who were present less than 50% of the time (no exceptions).**

3. **Create a colab or local Jupyter notebook for solving the tasks. Write the code and/or text of your answers in the cells specifying the ID of the question. (e.g., Q1.(b), Q2.Task 1-(a), etc).**

4. **Explain all steps clearly with proper comments**

5. **Evidences of communication between students, evidences of plagiarism and the use of external resources (automated, such as GPT or not) is automatically sanctioned with a D.**

1. **Q1: Sparse Signal Recovery** - Recover a grayscale image

   An **image** (in black and white) is in fact a **matrix** whose values represent the intensity of a pixel.

   The goal of this exercise is to recover an unknown grayscale image (called the ”signal”) based on linear measurements using the sparsity of the signal in the discrete cosine transform domain.

   2 (a) First load the image (in color: RGB = red, green, blue) and convert it into a black & white image, then into an A matrix with *numpy*, and check visually.

```python
# Import useful modules
import matplotlib.pyplot as plt
import numpy as np
from skimage.color import rgb2gray
from skimage import data
from scipy.fft import dct

# We select the "binary_blobs" image
caller = getattr(data, 'binary_blobs')
original = caller()
# Then we convert it in grayscale
grayscale = original # it is already grayscale
# Plot
fig, axes = plt.subplots(1, 2, figsize=(8, 4))
ax = axes.ravel()
ax[0].imshow(original)
ax[0].set_title("Original")
ax[1].imshow(grayscale, cmap=plt.cm.gray)
```

```
ax[1].set_title("Grayscale")
fig.tight_layout()
plt.show()

# We now convert the image into a matrix for processing with numpy
X_o = np.matrix(grayscale)/ 1.0
```

$\boxed{2}$ (b) Compute the (left) Discrete Cosine Transform (DCT) of $X_o \in \mathbb{R}^{n_1 \times n_2}$, denoted $X_{dct,o}$, such that $X_{dct,o} = \Psi X_o$ with $\Psi \in O(n_1)$ denoting the (left) DCT matrix (orthogonal matrix of size n_1). We recommend to use the function *dct* (type II) from *scipy.fft* Python package:

```
X_dct_o = dct(X_o, axis=0, norm='ortho')
```

We also recommend to explicitly build the matrix Ψ for further tasks:

```
Psi = dct(np.eye(n_1), axis=0, norm='ortho')
```

Display the entries of $X_{dct,o}$ and plot the Ordered Histogram of DCT Coefficients. Comment the results. What is the *structure* of $X_{dct,o}$?

$\boxed{4}$ (c) Image now we are interested in recovering X_o solely based on some linear measurements $Y \in \mathbb{R}^{m \times n_2}$ obtained through some random linear operators as follows: $Y = \Phi X_o$, with Φ a random matrix ($\Phi_{i,j} \sim \mathcal{U}(0, 1)$ for all i, j) of size $m \times n_1$. Explain the general approach to recover X_o based on Y (and the conclusions of part (b) ![1]), and formulate an optimization problem based on the ℓ^0 norm minimization. Then explain the main challenge with such a formulation. Is it a tractable problem ?

$\boxed{5}$ (d) Now, propose a convex relaxation of this first optimization problem and derive an algorithm to solve it relying on the subgradient method.

$\boxed{8}$ (e) It is time to test your algorithm ! Implement the algorithm in Python and test it with the following testing procedure:

1. Build the matrices Φ and Ψ, and the measurements Y.
2. Set $m = 150$, run your algorithm and check its numerical convergence.
3. Build your estimate of X_o, denoted $\tilde{X}_o$, and compare both visually and by computing some classical metrics for image comparison such that the Peak Signal-to-Noise Ratio (PSNR) (higher the better) and the Structural similarity index (close to one the best). The PSNR can be computed using the function *PSNR* from *cv2* Python library, and the Structural similarity index can be computed from

   ```
   from skimage.metrics import structural_similarity as ssim
   ```

   Discuss the results.

$\boxed{4}$ (f) To improve the results: increase value of m and repeat the procedure, discuss the results.

[1] hint: is X_o what you're going to actually compute?

2. **Q2: Nonnegative Matrix Factorization (NMF)**

The low-rank approximation of a matrix is a key problem in data analysis, and is widely used for Linear Dimensionality Reduction (LDR). LDR techniques such as principal component analysis are powerful tools for the analysis of high-dimensional data. In this exercise, we explore a popular variant of LDR, namely Nonnegative Matrix Factorization (NMF), which consists in a low-rank matrix approximation problem with nonnegativity constraints. More precisely, we seek to approximate a given nonnegative matrix X with the product of two nonnegative matrices, W and H, of smaller size. The nonnegativity constraints allow to extract easily interpretable and meaningful information from the input data.

The goal of this question is to implement a few simple algorithms for NMF, and then use them for two applications, namely facial feature extraction and Blind hyperspectral unmixing. We consider the following NMF optimization model

$$\begin{aligned} \min_{W,H} \quad & \frac{1}{2}\|X - WH\|_F^2 \\ \text{s.t.} \quad & W \geq 0, H \geq 0 \end{aligned} \tag{1}$$

where $X \in \mathbb{R}_+^{m\times n}$ is the input matrix, $W \in \mathbb{R}_+^{m\times r}$ and $H \in \mathbb{R}_+^{r\times n}$ are the variables with r being the given factorization rank, and $\|.\|_F^2$ is the squared Frobenius norm defined as $\|X - WH\|_F^2 = \sum_{i,j}[X - WH]_{i,j}^2$.

Algorithm: Multiplicative updates

Most NMF algorithms rely on alternating optimization; see Algorithm 1. The reason is that the subproblem in W and H separately are convex nonnegative least squares problems.

```
Algorithm 1 Alternating optimization framework for NMF
Input: $X \in \mathbb{R}^{m\times n}$, $W^{(0)} \in \mathbb{R}_+^{m\times r}$, $H^{(0)} \in \mathbb{R}_+^{r\times n}$, maximum number of iterations $T$
Output: An NMF solution $(W^{(T)}, H^{(T)}) \geq 0$ such that $\|X - W^{(T)}H^{(T)}\|_F^2$ is minimized.
for $i = 1, 2, \ldots, T$ do
    Update $W^{(i)}$ by solving exactly or approximately $\min_{W\geq 0} \|X - WH^{(i-1)}\|_F^2$.
    Update $H^{(i)}$ by solving exactly or approximately $\min_{H\geq 0} \|X - W^{(i)}H\|_F^2$.
end for
```

In the original paper that started NMF by Lee and Seung, the authors used the multiplicative updates (MU) that are guaranteed to decrease the objective function; see Algorithm 2. The notation $A \odot B$ means the component-wise product between matrices A and B of the same size, and $\frac{[A]}{[B]}$the component-wise division, and A^T denotes the transpose of A.

```
Algorithm 2 Multiplicative Updates (MU)
Input: $X \in \mathbb{R}^{m\times n}$, $W^{(0)} \in \mathbb{R}^{m\times r}$, $H^{(0)} \in \mathbb{R}^{r\times n}$
Output: An NMF solution $(W, H) \geq 0$ such that $\|X - WH\|_F^2$ is minimized.
for $i = 1, 2, \ldots$ do
    $W \leftarrow W \circ \frac{[XH^\top]}{[W(HH^\top)]}$
    $H \leftarrow H \circ \frac{[W^\top X]}{[(W^\top W)H]}$
end for
```

Task 1

3 (a) Implement the MU for NMF. Try it on a simple problem by generating

```
import numpy as np
X = np.dot(np.random.rand(m, r), np.random.rand(r, n))
```

and display the evolution of the relative error, $\frac{\|X-WH\|_F}{\|X\|_F}$.

3 (b) What is the computational cost of one iteration of the MU (in number of flops)? When computing $W(HH^T)$ in the update of W and $(W^TW)H$ in the update of H, why are the parenthesis crucial, especially for sparse matrices?

3 (c) Computing the error $\|X-WH\|_F$ naively by forming WH explicitly will require $O(mnr)$ operations and $O(mn)$ memory to store WH (which is typically dense even when X is sparse). This can be computed much more efficiently (especially for sparse matrices):

$$\begin{aligned}\|X-WH\|_F^2 &= \langle X-WH, X-WH\rangle \\ &= \langle X, X\rangle - 2\langle X, WH\rangle + \langle WH, WH\rangle \\ &= \|X\|_F^2 - \langle W^TX, H\rangle + \langle W^TW, HH^T\rangle\end{aligned}$$

where $\langle A,B\rangle = \sum_{i,j} A[i,j]B[i,j]$ is the inner product between two matrices of the same size. Implement the computation of the error in this way. Note that $\|X\|_F^2$ can be computed only once, while W^TX and W^TW are already computed when updating H with the MU.

Applications

The two data sets *CBCL.mat* and *Urban.mat* are available from this link (in sub-directory "data sets").

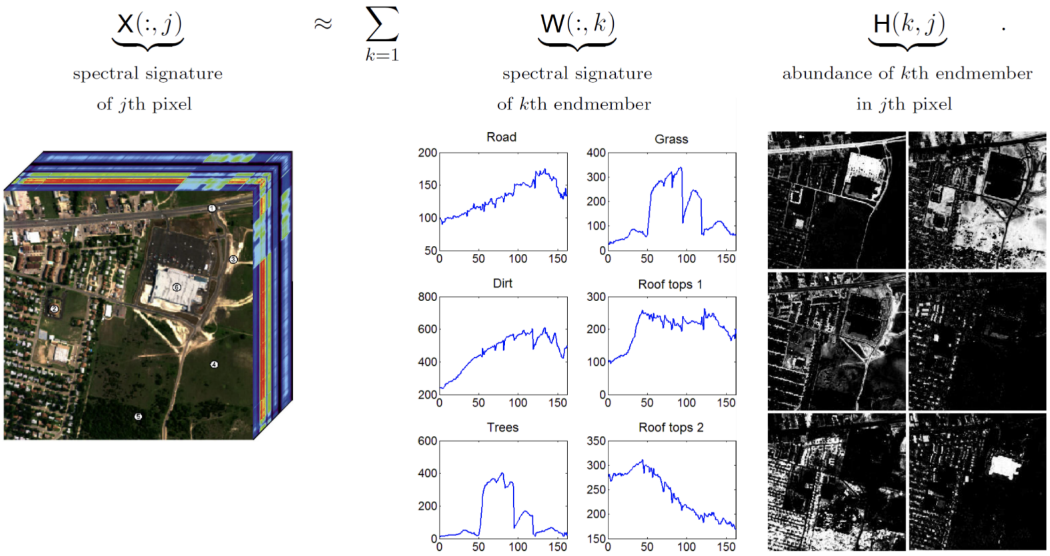
Task 2 - Facial feature extraction: CBCL

5 (a) Load the CBCL data set, and compute an NMF of rank 49 using the MU. Then display the facial features extracted. (you can simply reshape the columns of $W \in \mathbb{R}^{361\times49}$ as a 19-by-19 image, and display them.)

Task 3 - Blind hyperspectral unmixing: Urban data set

The hyperspectral image consists of 307x307 pixels, with each pixel having 162 spectral bands. These spectral bands represent different wavelengths across the electromagnetic spectrum, allowing for detailed analysis of the scene beyond the visible range.

Each pixel corresponds to one of the following materials present in the scenery (referred to as *endmembers*): Asphalt Road, Grass, Trees, Roofs and Dirt. Below is a reference decomposition for the urban dataset to help you assessing your results:



1 (a) Load the Urban data set. For convenience, the original order 3 tensor of size 307 × 307 × 162 has been reshaped into a matrix X of size 162 × 94249, such that each column of X corresponds to the spectral signature of one pixel.

5 (b) compute an NMF of rank 6 using the MU. Then interpret the result by identifying displaying each column of W, and the the row of H (which have to be reshaped as a 307-by-307 images).

Task 4 - Sensitivity to initialization

$\boxed{2}$ (a) Repeat tasks 2 for different initial factors $W^{(0)}$ and $H^{(0)}$. Do you get the same results?

$\boxed{3}$ (b) A very powerful method to initialize factor $W^{(0)}$ for the task of blind hyperspectral unmixing is called *Vertex Component Analysis* (VCA) [Nascimento et al., 2004]. You can find a simple Python implementation by clicking here. To make the most of this initialization method, we recommend reversing the order in which the W and H matrices are updated in Algorithm 2, i.e. updating the H factor first (one or more ?), then the W factor (why?). Repeat tasks 3 with initial $W^{(0)}$ computed with VCA with "snr input" input parameter of VCA routine set to 15dB. Do you get the same results?

In the following, you have the choice between two bonus questions. You can choose one of them, not both.

Bonus 1 - Going further - HALS

An algorithm that typically converges much faster than the MU is the hierarchical alternating least squares (HALS); see Algorithm 3. HALS was introduced by Cichocki, Zdunek and Amari.

Algorithm 3 Hierarchical Alternating Least Squares (HALS)

```
Input: $X \in \mathbb{R}^{m\times n}$, $W \in \mathbb{R}^{m\times r}$, $H \in \mathbb{R}^{r\times n}$
Output: An NMF solution $(W, H) \geq 0$ such that $\|X - WH\|_F^2$ is minimized.
for $i = 1, 2, \dots, T$ do
    Compute $XH^\top$ and $HH^\top$
    for $k = 1 \to r$ do
        $W_{:k} \leftarrow \max\left(0, \frac{(XH^\top)_{:k} - \sum_{l=1, l\neq k} W_{:l}(HH^\top)_{lk}}{(HH^\top)_{kk}}\right)$
    end for
    Compute $W^\top X$ and $W^\top W$
    for $k = 1 \to r$ do
        $H_{k:} \leftarrow \max\left(0, \frac{(W^\top X)_{k:} - \sum_{l=1, l\neq k} (W^\top W)_{kl} H_{l:}}{(W^\top W)_{kk}}\right)$
    end for
end for
```

The idea behind HALS is to update each column of W (resp. row of H) sequentially.

Task 5 [3]
Show that the formula used in HALS is the optimal solution of the NMF problem when optimizing only the kth column of W (that is, $W_{:k}$) in NMF (1), that is, show that the update in HALS corresponds to solving $\min_{W_{:k}\geq 0} \|X - WH\|_F^2$ exactly.

Task 6 [2]
Implement HALS. What is the computational cost of one iteration of HALS (in number of flops) ? (for dense matrices only).

Task 7 [5]
Compare the evolution of the relative error $\frac{\|X-WH\|_F}{\|X\|_F}$ with the MU on the two datasets (w.r.t. time and iteration). How much faster is HALS?

Bonus 2 - Going further - matrix completion with NMF

NMF may be a powerful method for estimating missing entries in a given matrix $X \in \mathbb{R}_+^{m \times n}$, provided the matrix X has a low-rank structure. This problem is usually referred to as *matrix completion*. The goal of this question is to develop a variant of Algorithm 2 to solve the following problem

$$\min_{W \in \mathbb{R}_+^{m \times r}, H \in \mathbb{R}_+^{r \times n}} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} M_{ij} \, (X - WH)_{ij}^2 \, , \tag{2}$$

where the mask matrix M is defined as follows

$$M_{ij} = \begin{cases} 1 & \text{if } (i,j)\text{-th entry of } X \text{ is known} \\ 0 & \text{otherwise} \end{cases} \quad ,$$

and the unknown inputs of X can be given any value (e.g. 0).

Task 5 [4]
Develop the expression of the MU for solving problem (2).

Help: say we are interesting in updating factor W while H is kept fixed, the first step consists in deriving the expression of the gradient of the objective function w.r.t. W, denoted $\nabla_W f(W,H)$. The next step consists in using a simple heuristic to formulate the MU as follows:

$$W^{(k+1)} := W^{(k)} \odot \frac{[\nabla_W^- f(W^{(k)}, H^{(k)})]}{[\nabla_W^+ f(W^{(k)}, H^{(k)})]} \tag{3}$$

where $\odot$ denotes the element-wise matrix multiplication, $\frac{[.]}{[.]}$ denotes the element-wise matrix division, $\nabla_W^+ f(.,.)$ and $\nabla_W^- f(.,.)$ respectively denote the positive and negative part of the gradient $\nabla_W f(.,.)$.

Task 6 [6]
Test your algorithm on a synthetic case, i.e. construct a non-negative random matrix Y or of rank r (the truth matrix we'll try to find). Next, generate a binary matrix M where each entry of one corresponds to the input we know and entries of zero correspond to unknown inputs. Construct $X = M \odot Y$. Now apply your new MU algorithm to try to recover the low rank X on the basis of Y and M. Plot the evolution of the objective function along the iterations and the evolution of the reconstruction error "$Y - X$" along the iterations. Repeat this operation for values of $r = [2, 5, 10]$ and with different percentages "p" of known inputs (percentage of ones in M) in $[10\%, 20\%, 30\%]$. Discuss the results.

3. **Q3: Compression of Convolutional Kernel in CNN with Canonical Polyadic Decomposition (CPD)**

Consider the convolutional kernel, $\mathcal{W}$, in layer 4.1conv1 of the pretrained Resnet18. The kernel has size of $512 \times 521 \times 3 \times 3$.

5 (a) Reshape the tensor to order-3 tensor of size $512 \times 512 \times 9$, then decompose it by CPD with different ranks $R = 30, ..., 300$.

Plot the compression ratios vs Rank and the relative approximation error vs Rank.

3 (b) Factorizing weights in convolutional and fully connected layers allows for the replacement of dense, large kernels with a network of smaller kernels. Specifically, employing a low-rank approximation (such as a low-rank CPD) of these kernels reduces the number of parameters in convolutional layers, thereby accelerating the network's inference. However, as noted by Lebedev et al. (2015), fine-tuning the entire network can be unstable ! Tensor decomposition may not be very effective if neural networks with factorized layers exhibit instability and are sensitive to parameter changes. Additionally, applying tensor decompositions to compress convolutional neural networks (CNNs) is not straightforward, primarily due to the sensitivity of the Canonical Polyadic Decomposition (CPD), which is also referred to as the degree of degeneracy.

There are two possible measures for the sensitivity degree, or degeneracy degree, in the CPD $[\![A, B, C]\!]$ of a given tensor $\mathcal{T}$ of size $I \times J \times K$:

1. Measure 1 - Norms of the rank-1 tensors [Phan et al., 2019]:

$$sn[\![A, B, C]\!] = \sum_{r=1}^{R} \|a_r \circ b_r \circ c_r\|_F^2 \tag{4}$$

2. Measure 2 - Sensitivity in the sense of [Tichavský et al., 2019]:

$$ss[\![A, B, C]\!] = \lim_{\sigma^2 \to 0} \frac{1}{\sigma^2 R} \mathbb{E}\{\|\mathcal{T} - [\![A + \delta A, B + \delta B, C + \delta C]\!]\|_F^2\} \tag{5}$$

where $\delta A, \delta B, \delta C$ have random i.i.d. elements from $\mathcal{N}(0, \sigma^2)$. It is a measure with respect to perturbations in individual factor matrices. Hopefully for us, Equation (5) can be simplified as follows:

$$ss[\![A, B, C]\!] = K\text{trace}\{(A^T A) \odot (B^T B)\} + I\text{trace}\{(B^T B) \odot (C^T C)\} + J\text{trace}\{(A^T A) \odot (C^T C)\}. \tag{6}$$

with $\odot$ the Hadamard product (element-wise matrix product).

Decompositions with high sensitivity are usually useless, for instance when it is around $10^4 - 10^5$. Compute the sensitivity of the decompositions obtained at point (a) with Equation (6), what is your conclusion ? Is the sensitivity measure related to the unbalancing of rank-1 tensor norms ?

2 (c) There are various methods available to address degeneracy, which can enhance stability and convergence. Notable approaches include adding orthogonality constraints to the factors of the Canonical Polyadic Decomposition (CPD), incorporating Tikhonov regularization on the factors, and more recently, the Error Preserving Correction (EPC) method. In this context, we will explore Tikhonov regularization by setting the Tikhonov matrices to identity matrices for simplicity. This approach is equivalent to applying squared Frobenius norm regularization to each factor A, B, and C. With a weight μ, CPD with such regularizations can be computed using the *constrained_parafa* function from *Tensorly*. Repeat steps (a) and (b) with $\mu = [0.1, 1, 10]$. What observations do you make? Would you recommend this method for addressing degeneracy?

4. **Q4: Additional question - Minimization of the Rosenbrock function with CPD**

The multidimensional Rosenbrock function is defined as

$$f(x_1, ..., x_N) = \sum_{i=1}^{N-1} b(x_{i+1} - x_i^2)^2 + (a - x_i)^2. \tag{7}$$

where $a = 1$ and $b = 100$, $N = 3, 4, ...$ in the grid $[-2, 2] \times \cdots \times [-2, 2] \times [-1, 3]$.

$\boxed{5}$ (a) Create a tensor $\mathcal{T}$ by randomly sampling the Rosenbrock function of order $N = 3$ in the grid $[-2, 2] \times [-2, 2] \times [-1, 3]$ with the step 0.05, so that the tensor has the size $81 \times 81 \times 81$ and only 10% of tensor elements are available.
Bonus (3) Show that the incomplete tensor $\mathcal{T}$ can be approximated by CP for $N = 3$ and $N = 4$. What should be the CP rank for both cases ?

$\boxed{5}$ (b) Find the minimum of $f(x_1, ..., x_N)$ for $N = 3, 4$ via its low-rank tensor approximation.
Hint: use the *tensorly* Python library, in which you can find a routine called *parafac* to compute the CP of the tensor, which allows to input mask matrices (binary matrices).