TP sur la réduction de dimension et auto-encodeurs (Partie 1)

Indications générales :

1. Le TP se fait impérativement en travail individuel.
2. Un compte rendu obligatoire en format PDF doit être soumis par chaque étudiant avant le 23/01/2024 à 23h55.
3. Dans le compte rendu vous présentez le code utilisé pour résoudre chaque partie (dans le cas où le code n'est pas donné) ainsi que les résultats obtenus et l'interprétation détaillée des résultats le cas échéant.
4. L'évaluation est principalement sur votre capacité d'analyser, de critiquer et d'interpréter les résultats. Ainsi, il est essentiel d'expliquer clairement vos conclusions.
5. Les codes sont donnés en Python et R-Studio. Mais si vous êtes plus à l'aise avec un autre langage, n'hésitez pas à l'utiliser.

Problème I : Reconstruction de chiffres manuscrits (utilisation Keras avec Python) :

1. On suppose que Anaconda est installé. Nous commençons par importer les librairies utiles. De plus nous allons téléchargé la base de données MNIST qui comporte les descriptions de chiffres manuscrits en format 28×28 (784 pixels). La base contient 7000 exemples de chaque chiffre (Soit 70000 exemples).

```
from keras.datasets import mnist
import numpy as np
(train_images,_),(test_images,_)=mnist.load_data()
print(train_images.shape)
print(test_images.shape)
```

2. Visualiser une des images contenus dans la base MNIST :

```
import matplotlib.pyplot as plt
plt.imshow(test_images[0])
plt.gray()
```

3. Normalisation et redimensionnement des données (apprentissage et test) afin de les utiliser comme des données d'entrée dans notre AE

```
train_images = train_images.astype('float32')/255.
test_images = test_images.astype('float32')/255.
train_images = train_images.reshape(len(train_images),
        np.prod(train_images.shape[1:]))
test_images = test_images.reshape(len(test_images),
        np.prod(test_images.shape[1:]))
print(train_images.shape)
print(test_images.shape)
```

4. Construction d'un AE avec une couche transitoire de dimension 32 et les fonctions d'activation 'relu' et 'sigmoid' respectivement

```
from keras.layers import Input, Dense
from keras.models import Model

encoding_dim = 32
#the input layer with 784 features
input_layer = Input(shape=(784,))
#the hidden or encoded layer with 32 dimensions
encoder_layer1 = Dense(encoding_dim, activation='relu')(input_layer)
#the decoded layer with 784 features
decoder_layer1 = Dense(784, activation='sigmoid')(encoder_layer1)
# this model maps an input to its reconstruction
autoencoder = Model(input_layer, decoder_layer1)
autoencoder.summary()
```

5. Pour des raisons de test, construisez séparément l'encodeur et le décodeur

```
encoder = Model(input_layer,encoder_layer1)
encoded_input = Input(shape=(encoding_dim,))
decoder_layer = autoencoder.layers[-1]
decoder = Model(encoded_input,decoder_layer(encoded_input))
```

6. Compiler l'AE de la partie (4) en considérant 'adam' comme algorithme d'optimisation (optimizer) et 'binary_crossentropy' comme fonction de perte.
Puis faire l'apprentissage et le test du modèle en utilisant les données MNIST (Fixer les paramètres epochs = 60 et batch_size = 256).

```
autoencoder.compile(optimizer='adam',loss='binary_crossentropy')

#batch: size is a number of samples
#processed before the model is updated.
#epochs: is the number of
#complete passes through the training dataset.
epochs = 60
```

```
batch_size = 256

autoencoder.fit(train_images,train_images,
                epochs=epochs,
                batch_size=batch_size,
                shuffle=True,
                validation_data=(test_images,test_images))
```

7. Utiliser l'autoencodeur des parties précédentes afin de reconstruire les images test de MNIST.
   Puis comparer les prédictions de l'AE pour les cinq premières images de l'ensemble test.

```
encoded_imgs = encoder.predict(test_images)
decoded_imgs = decoder.predict(encoded_imgs)
print(encoded_imgs.shape)
print(decoded_imgs.shape)


import matplotlib.pyplot as plt
n=5
plt.figure(figsize=(20,4))
for i in range(n):
    #original images
    ax = plt.subplot(2,n,i+1)
    plt.imshow(test_images[i].reshape(28,28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    #reconstructed images
    ax = plt.subplot(2,n,i+1+n)
    plt.imshow(decoded_imgs[i].reshape(28,28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

8. Refaire la partie (6) en considérant 4 combinaisons différentes, de votre choix, d'algorithme d'optimisation et de fonction de perte. Calculer la 'mse' de l'AE de chaque combinaison (en utilisant les données test) et sélectionner celui avec la meilleur mse. Interpréter le résultat. (Présenter vos résultats sous forme d'un tableau à deux colonnes, la première indiquant la combinaison utilisée et la deuxième la mse obtenue)
9. Maintenant, en considérant l'AE optimal de la partie (8) refaire la partie (6) mais cette fois en considérant 4 combinaisons différentes, de votre choix, d'epochs et de batch_size. Calculer la 'mse' de l'AE de chaque

combinaison (en utilisant les données test) et sélectionner celui avec la meilleur mse. Interpréter le résultat. (Présenter vos résultats sous forme d'un tableau à deux colonnes, la première indiquant la combinaison utilisée et la deuxième la mse obtenue)

10. Refaire la partie (7) en considérant l'AE optimal de la partie (9). Puis visualiser les images originales vs reconstruites des cinq images de l'ensemble test ayant les erreurs de reconstruction les moins élevées. Interpréter le résultat.

Problème II : Reconstruction d'images d'un Pianobar en 3 méthodes :

1. On suppose que Anaconda est installé. Nous commençons par importer les librairies utiles. De plus nous allons téléchargé, visualisé et traité la photo intitulée 'pianobar'.

```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
#PIL is Python Imaging Library
from PIL import Image
import math
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

img = Image.open('pianobar.pgm')
plt.figure(figsize=(10,6))
plt.imshow(img,cmap = plt.cm.gray)

image_matrix = np.array(img)
original_dimensions = image_matrix.shape
```

2. A l'aide de la méthode classique de l'ACP reconstruisez l'image après avoir réduit la largeur ('width') de l'image originale de 460 à 46. Puis calculer la 'mse' de votre reconstruction (Vous pouvez aussi en optionnel visualiser l'image reconstruite)
3. Construisez un AE avec une couche transitoire de dimension 46 et une fonction d'activation linéaire. Puis, compiler cet AE en considérant 'adam' comme algorithme d'optimisation (optimizer) ; 'MeanSquaredError' comme fonction de perte et epochs=500 **(Penser à utiliser la librairie 'StandardScaler()' pour transformer la matrice des données de l'image avant de l'intégrer dans le AE).** Enfin, calculer la 'mse' de votre reconstruction (Vous pouvez aussi en optionnel visualiser l'image reconstruite) et comparer le résultat avec celui de la partie (2). Interpréter.

4. Construisez un AE profond avec 3 couches transitoires de dimensions 128 ; 46 et 128 respectivement et des fonctions d'activation 'relu' et 'sigmoid'. Puis, compiler cet AE en considérant 'adam' comme algorithme d'optimisation (optimizer) ; 'MeanSquaredError' comme fonction de perte et epochs=500 **(Penser à utiliser la librairie 'StandardScaler()' pour transformer la matrice des données de l'image avant de l'intégrer dans le AE).** Enfin, calculer la 'mse' de votre reconstruction (Vous pouvez aussi en optionnel visualiser l'image reconstruite) et comparer le résultat avec ceux des parties (2 et 3). Interpréter.