TP sur la réduction de dimension et auto-encodeurs (Partie 2)

Problème III : Reconstruction de chiffres manuscrits et détection des anomalies (Indication : utilisation h2o avec R-studio) :

1. On suppose que R et R-studio sont installés. Nous commençons par importer les librairies utiles + Initialisation de la session sous h2o. 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).

```
library(dplyr)    # for data manipulation
library(ggplot2)  # for data visualization
library(h2o)  # for fitting autoencoders

mnist <- dslabs::read_mnist()
names(mnist)

##### Initialisation de la session h2o
```

```
h2o.no_progress()
h2o.init(max_mem_size = "5g")

###télégargement des données
mnist <- dslabs::read_mnist()
```

2. Convertir les données MNIST en un ensemble de données d'entrée sous h2o.

```
features <- as.h2o(mnist$train$images)
testing <- as.h2o(mnist$test$images)
```

3. Entrainer un AE d'une couche intermédiaire de dimension 2 et d'une fonction d'activation 'Tanh' sur les données d'apprentissage de MNIST. En cas de message d'avertissement (Warning), expliquer la cause de ce message.

```
ae1 <- h2o.deeplearning(
  x = seq_along(features),
  training_frame = features,
  autoencoder = TRUE,
  hidden = c(2),
  activation = 'Tanh',
  sparse = TRUE )


# Extract the deep features
ae1_codings <- h2o.deepfeatures(ae1, features, layer = 1)
ae1_codings
```

4. Déterminer le meilleur AE parmi les architectures suivantes pour les couches intermédiaires : c(50) ; c(100) ; c(300, 100, 300) ; c(100, 50, 100) ; c(250, 100, 50, 100, 250).
Ordonner la performance des différents AE en terme de 'mse' et interpréter le résultat.

```
# Hyperparameter search grid
hyper_grid <- list(hidden = list(
  c(50),
  c(100),
  c(300, 100, 300),
  c(100, 50, 100),
  c(250, 100, 50, 100, 250)
))


# Execute grid search
ae_grid <- h2o.grid(
  algorithm = 'deeplearning',
```

```
  x = seq_along(features),
  training_frame = features,
  grid_id = 'autoencoder_grid',
  autoencoder = TRUE,
  activation = 'Tanh',
  hyper_params = hyper_grid,
  sparse = TRUE,
  ignore_const_cols = FALSE,
  seed = 123 )


# Print grid details
h2o.getGrid('autoencoder_grid',
        sort_by = 'mse',
        decreasing = FALSE)
```

5. Choisissez un échantillon aléatoire de 4 images de l'ensemble test des données MNIST, puis renommer les colonnes de cet échantillon par : V1; V2; V3; ....;V784.

```
# Get sampled test images
index <- sample(1:nrow(mnist$test$images), 4)
sampled_digits <- mnist$test$images[index, ]
colnames(sampled_digits) <- paste0("V", seq_len(ncol(sampled_digits)))
```

6. En utilisant le meilleur modèle de la partie (4) reconstruisez les 4 images de la partie (5). Puis visualiser les images originales vs reconstruites.

```
# Predict reconstructed pixel values
best_model_id <- ae_grid@model_ids[[1]]
best_model <- h2o.getModel(best_model_id)
reconstructed_digits <- predict(best_model, as.h2o(sampled_digits))
names(reconstructed_digits) <- paste0("V",
        seq_len(ncol(reconstructed_digits)))

combine <- rbind(sampled_digits, as.matrix(reconstructed_digits))


# Plot original versus reconstructed
par(mfrow = c(1, 3), mar=c(1, 1, 1, 1))
layout(matrix(seq_len(nrow(combine)), 4, 2, byrow = FALSE))
for(i in seq_len(nrow(combine))) {
  image(matrix(combine[i, ], 28, 28)[, 28:1],
  xaxt="n", yaxt="n")
}
```

Nous pouvons également utiliser des AE pour la détection des anomalies (Sakurada et Yairi 2014). Puisque la fonction de perte

d'un AE mesure l'erreur de reconstruction, nous pouvons extraire ces informations pour identifier les observations qui ont des taux d'erreur plus élevés. Ces observations ont des attributs de caractéristiques qui diffèrent considérablement des autres caractéristiques. Nous pourrons considérer ces caractéristiques comme anormales ou aberrantes.

7. Calculer les erreurs de reconstruction de toutes les images de l'ensemble test en utilisant le meilleur AE de la partie (4). Puis représenter graphiquement la distribution de ces erreurs et interpréter le résultat.

```
# Extract reconstruction errors
(reconstruction_errors <- h2o.anomaly(best_model, features))
# Plot distribution
reconstruction_errors <- as.data.frame(reconstruction_errors)
ggplot(reconstruction_errors, aes(Reconstruction.MSE)) +
geom_histogram()
```

8. Représenter les images originales vs reconstruites des 4 images ayant les erreurs de reconstruction les plus élevées. Interpréter le résultat.

```
###plot the worst five
index=which(reconstruction_errors$Reconstruction.MSE %in%
sort(reconstruction_errors$Reconstruction.MSE,
decreasing = TRUE)[1:4])
worst_digits <- mnist$train$images[index, ]
reconstructed_worst_digits <- predict(best_model,
as.h2o(worst_digits))
combine_worst <- rbind(worst_digits,
as.matrix(reconstructed_worst_digits))
# Plot original versus reconstructed
par(mfrow = c(1, 3), mar=c(1, 1, 1, 1))
layout(matrix(seq_len(nrow(combine_worst)), 5, 2, byrow = FALSE))
for(i in 1:nrow(combine_worst)) {
  image(matrix(combine_worst[i, ], 28, 28)[, 28:1])
}
```

Problème IV : Reconstruction d'images de mode (utilisation Keras avec Python 3) :

1. On suppose que Anaconda est installé. Nous commençons par importer les librairies utiles. De plus nous allons téléchargé et traité la base de données 'fashion_mnist'.

```
import os
import numpy as np
import pandas as pd
import keras
from keras import layers
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import regularizers
from keras.layers import Input, Dense
from keras.models import Model
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, losses
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.datasets import fashion_mnist

#import data
mnist = fashion_mnist.load_data()

#training and testing sets
(x_tr, _), (x_tst, _) = mnist
print(x_tr.shape)
print(x_tst.shape)

#Visualiser une image dans la base mnist
plt.imshow(x_tst[0])
plt.gray()

#Normalisation et redimensionnement des données
x_tr = x_tr.astype('float32') / 255.
x_tst = x_tst.astype('float32') / 255.
x_tr = x_tr.reshape(len(x_tr),
        np.prod(x_tr.shape[1:]))
x_tst = x_tst.reshape(len(x_tst),
        np.prod(x_tst.shape[1:]))
```

2. Construction d'un AE avec une couche transitoire de dimension 64 et les fonctions d'activation 'relu' et 'sigmoid' respectivement

```
encoding_dim = 64
#the input layer with 784 features
input_layer = Input(shape=(784,))
#the hidden or encoded layer with 64 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()
```

3. Compiler l'AE de la partie (2) en considérant 'adam' comme algorithme d'optimisation (optimizer) et 'MeanSquaredError' comme fonction de perte. Puis faire l'apprentissage et le test du modèle en utilisant les données mnist (Fixer le paramètre epochs = 10).

```
autoencoder.compile(optimizer='adam', loss='MeanSquaredError')
autoencoder.fit(x_tr,x_tr,
                epochs=10,
        validation_data=(x_tst,x_tst))
```

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

```
reconstructed_imgs = autoencoder.predict(x_tst)

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(x_tst[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(reconstructed_imgs[i].reshape(28,28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

5. Refaire les parties (2, 3 et 4) en considérant un AE profond (3 couches intermédiaires symétriques de tailles 128 ; 64 et 128 respectivement). Comparer la performance (au niveau des données test) des deux AE et interpréter le résultat.
6. CHALLENGE!! : refaire les parties (2, 3 et 4) en considérant un AE convolutif d'architecture symétrique de votre choix. Comparer la performance (au niveau des données test) des trois AE et interpréter le résultat.