```
# Crée un réseau de neurones avec Keras

from keras.models import Sequential
from keras.layers import Dense, Dropout
import numpy as np
import h5py
import sys
from os import remove, path
from glob import glob

# Paramètres
#Affichage du traitement des données
AFFICHAGE = True
# Nombre de coefficients cepstraux
nbCoef = 13
# Nombre de valeurs à prélever pour obtenir une "fenêtre de parole"
nbVal = 101
# Décalage entre chaque prélevement de fenêtre de parole
shift = 10
#Nombre de langages différents = de neurones en sortie
nbSorties = 4


def formatDataMemory(dataFileName, withoutLabel=False):
    '''
    Formatte les données d'un fichier hdf5 qui est organisé en
       plusieurs datasets
    qui contiennent chacun des fenêtres de parole en deux tableaux
       numpy (données et étiquettes)
    lisibles par Keras.
    :param dataFile: Le fichier hdf5 d'entrée.
    :param nbVal: Le nombre de valeurs à prélever pour obtenir une
       "fenêtre de parole"
    :param nbCoef: Les nombre de coefficient par vecteur acoustique
    :param shift: Le décalage entre chaque fenêtre de parole
    :param withoutLabel:Permet de spécifier que les données ne sont
       pas étiquettées
    '''
    hdf5In = h5py.File(dataFileName, "r")

    # Calcul du nombre total de fenêtres de parole de la base
       d'apprentissage
    # Une fenêtre est une matrice contenant des coefficients
       cepstraux :
    # - de nbVal lignes
    # - de nbCoef colonnes
    totalFrames = 0
    for dataset in hdf5In.values():                                #
       Pour chaque dataset du fichier hdf5
        frames = int ((dataset.shape[0] - nbVal) / shift) + 1   # Nb
           fenêtres du dataset
        totalFrames += frames

    X = np.zeros([totalFrames, nbVal * nbCoef], dtype=np.float32)
    Y = np.zeros([totalFrames, nbSorties], dtype=np.float32)
    # On va stocker les valeurs de la fenêtre de parole dans un
```

```
        tableau numpy temporaire
    frameValues = np.zeros([nbVal * nbCoef], dtype=np.float32)
    frameIndex = 0
    # Parcours des datasets du fichier d'entrée afin de remplir les
        datasets temporaires
    for dataset in hdf5In.values():
        if AFFICHAGE : print('Traitement de ' + dataset.name)
        # On prend chaque fenêtre de parole du dataset courant
        for frameStartIndex in range(0, len(dataset) - nbVal, shift):
            # Une fenêtre de nbVal vecteurs
            frameVectors = dataset[frameStartIndex : frameStartIndex
                + nbVal]
            # On prend le langage sur le premier vecteur de la
                fenêtre (dernière valeur du vecteur)
            if not withoutLabel:
                language = frameVectors[0][len(frameVectors[0])-1]
            vectorIndex = 0
            # Pour chaque vecteur (contenant nbCoef valeurs) de la
                fenêtre courante
            for vector in frameVectors :
                # On remplit le tableau de la fenêtre de parole avec
                    les coefficients du vecteur
                frameValues[vectorIndex : vectorIndex + nbCoef] =
                    vector[0 : nbCoef]
                vectorIndex += nbCoef  # au fur et à mesure

            # On remplit les tableaux numpy avec les données et le
                langage
            X[frameIndex] = frameValues
            if not withoutLabel:
                Y[frameIndex, int(language)] = 1
            frameIndex += 1

    hdf5In.close()

    if withoutLabel:
        return X
    else:
        return (X, Y)

def writeProbaToFile(outFileName, exampleFileName, exampleFileLang,
    probaArray, langStrList, numberPrecision=2, numberOfMinSpaces=2):
    np.round(probaArray, decimals=numberPrecision)  # On arrondit
        les probas

    exampleFileName = path.basename(exampleFileName)

    # Variables pour l'alignement (formattage)
    probaNbOfChar = 2 + numberPrecision                         # Nb de
        caractères pris par une proba
    langNbOfChar = max(len(lang) for lang in langStrList)   # Nb de
        caractères max pris par une langue
    nbOfCharPerColumn = max(probaNbOfChar, langNbOfChar)    # Nb de
        caractères dans chaque colonne

    # On forme le header avec le nom du fichier et les langages dans
```

```
        l'ordre
    headerStr = 'Fichier ' + exampleFileName + ' (langue : ' +
        exampleFileLang +')\n'
    for lang in langStrList:    # Ligne des langages
        offset = nbOfCharPerColumn - len(lang)
        headerStr += str(lang) + ' ' * (offset + numberOfMinSpaces)

    # Le séparateur entre tous les nombres à écrire (un certain nb
        d'espaces)
    numberDelimiter = ' ' * (nbOfCharPerColumn - probaNbOfChar +
        numberOfMinSpaces)

    # On forme le footer avec le nom du fichier pour rappel
    footerStr = 'Fichier ' + exampleFileName + ' (langue : ' +
        exampleFileLang +')\n'
    # Et la moyenne des probas pour chaque colonne (= langue)
    average = np.round(probaArray.mean(axis=0),
        decimals=numberPrecision)
    footerStr += 'Moyenne :\n' +
        numberDelimiter.join(np.char.mod('%.2f', average))
    footerStr +=
        '--------------------------------------------------\n'

    fmtValue = '%1.' + str(numberPrecision) + 'f'   # Format des
        nombres

    with open(outFileName, 'ab') as outFile:
        np.savetxt(outFile, probaArray, fmt=fmtValue,
            delimiter=numberDelimiter, newline='\n',
            header=headerStr, footer=footerStr, comments='')

#fonction temporaire : génère les fichiers txt pour voir les probas
    résultat
def generatePredict(model, predictFolder, language):
    langs = ['Arabe', 'Anglais', 'Francais', 'Allemand']
    for i in glob(predictFolder+'/*.hdf5'):
        Xtest = formatDataMemory(i)[0]
        proba = model.predict_proba(Xtest, batch_size=128, verbose=0)
        writeProbaToFile("probas.txt", path.splitext(i)[0],
            language, proba, langs)

if __name__ == '__main__':
    # On récupère les données
    dataFileName = sys.argv[1]
    devFileName = sys.argv[2]
    testFileName = sys.argv[3]

    # On formatte les données
    (X, Y) = formatDataMemory(dataFileName)
    (Xdev, Ydev) = formatDataMemory(devFileName)
    (Xtest, Ytest) = formatDataMemory(testFileName)

    # On crée le modèle séquentiel, avec 4 couches
    inputShape = nbVal * nbCoef
    model = Sequential()
    model.add(Dense(inputShape, input_shape=(inputShape,),
```

```
        kernel_initializer='glorot_normal', activation='relu'))
    model.add(Dropout(0.2, input_shape=(inputShape,)))
    model.add(Dense(256, kernel_initializer='glorot_normal',
        activation='relu'))
    model.add(Dropout(0.2, input_shape=(inputShape,)))
    model.add(Dense(256, kernel_initializer='glorot_normal',
        activation='relu'))
    model.add(Dense(nbSorties, kernel_initializer='glorot_normal',
        activation='softmax'))

    # On compile le modèle
    model.compile(loss='categorical_crossentropy',
        optimizer='RMSprop', metrics=['accuracy'])

    # On entraîne le modèle
    model.fit(X, Y, epochs=100, batch_size=128,
        validation_data=(Xdev, Ydev))

    generatePredict(model, 'hdf5Predict/Arabic', 'Arabic')
    generatePredict(model, 'hdf5Predict/English', 'English')
    generatePredict(model, 'hdf5Predict/French', 'French')
    generatePredict(model, 'hdf5Predict/German', 'German')


#Formatte les données en le mettant dans un tableau hdf5 temporaire
    (utile pour les fichiers dépassant la taille de la ram)
def formatDataHdf5(dataFileName, hdf5Tmp, withoutLabel=False):
    '''
    Formatte les données d'un fichier hdf5 qui est organisé en
        plusieurs datasets
    qui contiennent chacun des fenêtres de parole en deux tableaux
        numpy (données et étiquettes)
    lisibles par Keras.
    :param dataFile: Le fichier hdf5 d'entrée.
    :param hdf5Tmp: Le fichier hdf5 contenant les donnée structurées
    :param nbVal: Le nombre de valeurs à prélever pour obtenir une
        "fenêtre de parole"
    :param nbCoef: Les nombre de coefficient par vecteur acoustique
    :param shift: Le décalage entre chaque fenêtre de parole
    :param withoutLabel:Permet de spécifier que les données ne sont
        pas étiquettées
    '''
    hdf5In = h5py.File(dataFileName, "r")

    # Calcul du nombre total de fenêtres de parole de la base
        d'apprentissage
    # Une fenêtre est une matrice contenant des coefficients
        cepstraux :
    # - de nbVal lignes
    # - de nbCoef colonnes
    totalFrames = 0
    for dataset in hdf5In.values():                                 #
        Pour chaque dataset du fichier hdf5
        frames = int ((dataset.shape[0] - nbVal) / shift) + 1   # Nb
            fenêtres du dataset
```

```
    totalFrames += frames

# On places les datasets générés par cette fonction dans des
   groupes nommés de 1 à n
groupLastName = 0
for groupName in hdf5Tmp:
    groupLastName = groupName
newGroupName = str(int(groupLastName) + 1)
# Création des datasets contenant les données formatées
hdf5Tmp.create_dataset(newGroupName+"/examples", (totalFrames,
   nbVal * nbCoef)) # dataset données
X = hdf5Tmp.get(newGroupName+"/examples")
if not withoutLabel:
    hdf5Tmp.create_dataset(newGroupName+"/languages",
       (totalFrames, 4))           # dataset étiquettes
    Y = hdf5Tmp.get(newGroupName+"/languages")

# On va stocker les valeurs de la fenêtre de parole dans un
   tableau numpy temporaire
# cela permet d'accélerer le remplissage des datasets temporaires
frameValues = np.zeros([nbVal * nbCoef])
frameIndex = 0
# Parcours des datasets du fichier d'entrée afin de remplir les
   datasets temporaires
for dataset in hdf5In.values():
    print(dataset)
    # On prend chaque fenêtre de parole du dataset courant
    for frameStartIndex in range(0, len(dataset) - nbVal, shift):
        # Une fenêtre de nbVal vecteurs
        frameVectors = dataset[frameStartIndex : frameStartIndex
           + nbVal]
        # On prend le langage sur le premier vecteur de la
           fenêtre (dernière valeur du vecteur)
        if not withoutLabel:
            language = frameVectors[0][len(frameVectors[0])-1]
        vectorIndex = 0
        # Pour chaque vecteur (contenant nbCoef valeurs) de la
           fenêtre courante
        for vector in frameVectors :
            # On remplit le tableau de la fenêtre de parole avec
               les coefficients du vecteur
            frameValues[vectorIndex : vectorIndex + nbCoef] =
               vector[0 : nbCoef]
            vectorIndex += nbCoef  # au fur et à mesure

        # On remplit le dataset data avec les coefficients du
           vecteur
        X[frameIndex] = frameValues
        if not withoutLabel:
            tabTemp = np.zeros([4])
            tabTemp[int(language)] = 1
            Y[frameIndex] = tabTemp
        frameIndex += 1

hdf5In.close()
```

```
    if withoutLabel:
        return X
    else:
        return (X, Y)
```