Réunion 06/04

Nous avons converti les mfcc en un fichier hdf5 (environ 330Mo) qui contient tous les vecteurs acoustiques (avec la langue en 14è valeur) de tous les fichiers, séparés en datasets (un dataset par fichier).

Du coup nous traitons les données en en faisant un autre fichier hdf5, avec un dataset de taille `nombre_d'exemples` * `403` pour les valeurs audio et un autre de taille `nombre_d'exemples` pour les étiquettes.

Code du fichier lançant Keras :

```
# Crée un réseau de neurones avec Keras

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.utils.np_utils import to_categorical
import numpy as np
import h5py
import sys
from os import remove

def formatDataForKeras(dataFileName, hdf5Tmp, nbVal, nbCoef, shift,
   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,))           # 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.empty([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:
                Y[frameIndex] = language
            frameIndex += 1

    hdf5In.close()

    if withoutLabel:
        return X
    else:
        Y = to_categorical(Y)   # On met les étiquettes sous forme
           de vecteurs binaires pour keras
        return (X, Y)

if __name__ == '__main__':
```

```
# Paramètres
# 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

# On récupère les données
dataFileName = sys.argv[1]
devFileName = sys.argv[2]
testFileName = sys.argv[3]
# On crée le fichier temporaire hdf5
tmpFile = "tmp.hdf5"
hdf5Tmp = h5py.File(tmpFile, "w")

# On formatte les données
(X, Y) = formatDataForKeras(dataFileName, hdf5Tmp, nbVal,
   nbCoef, shift)
(Xdev, Ydev) = formatDataForKeras(devFileName, hdf5Tmp, nbVal,
   nbCoef, shift)
(Xtest, Ytest) = formatDataForKeras(testFileName, hdf5Tmp,
   nbVal, nbCoef, shift)

# 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(4, 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=250, batch_size=128,
   validation_data=(Xdev, Ydev), shuffle='batch')

hdf5Tmp.close() # On peut fermer le fichier temporaire
remove(tmpFile) # et le supprimer

# On évalue le modèle
#score = model.evaluate(Xtest, Ytest)[1]
#print("Accuracy : " + score)
```

Résultat de l'exécution (10 epochs) :

```
Epoch 1/10
116960/116960 [==============================] - 15s - loss: 1.1964 - acc: 0.4636
    - val_loss: 1.1250 - val_acc: 0.5120
Epoch 2/10
116960/116960 [==============================] - 15s - loss: 0.9184 - acc: 0.6120
    - val_loss: 1.1396 - val_acc: 0.5623
Epoch 3/10
116960/116960 [==============================] - 15s - loss: 0.7703 - acc: 0.6934
    - val_loss: 1.3399 - val_acc: 0.5009
Epoch 4/10
116960/116960 [==============================] - 15s - loss: 0.6882 - acc: 0.7322
    - val_loss: 1.3696 - val_acc: 0.5423
Epoch 5/10
116960/116960 [==============================] - 15s - loss: 0.6344 - acc: 0.7574
    - val_loss: 1.1985 - val_acc: 0.5890
Epoch 6/10
116960/116960 [==============================] - 15s - loss: 0.5759 - acc: 0.7830
    - val_loss: 1.5915 - val_acc: 0.5550
Epoch 7/10
116960/116960 [==============================] - 15s - loss: 0.5509 - acc: 0.7937
    - val_loss: 2.4062 - val_acc: 0.5499
Epoch 8/10
116960/116960 [==============================] - 15s - loss: 0.5321 - acc: 0.8023
    - val_loss: 2.1470 - val_acc: 0.5685
Epoch 9/10
116960/116960 [==============================] - 15s - loss: 0.5104 - acc: 0.8105
    - val_loss: 2.2892 - val_acc: 0.5828
Epoch 10/10
116960/116960 [==============================] - 15s - loss: 0.4899 - acc: 0.8208
    - val_loss: 3.1017 - val_acc: 0.4728
```