INSAT	Module : Deep Learning
Département Mathématiques et Informatique	Sections : GL4 & RT4
Année Universitaire : 2023 - 2024	Enseignante : Sana Hamdi sana.hamdi@fst.utm.tn

TP N°4 : Les Réseaux de Neurones Récurrents

Objectif :

Ce TP a pour objectif d'appliquer trois types différents de réseaux de neurones profonds sur un ensemble de données du monde réel. Nous utiliserons le réseau de neurones à connexion dense (réseau de neurones de base), le réseau de neurones convolutionnel (CNN) et le réseau de mémoire à long court terme (LSTM), qui est une variante des réseaux de neurones récurrents.

Nous allons utiliser Keras Embedding Layer et GloVe comme word embedding pour convertir le texte en forme numérique.

Dataset :

IMDB dataset contient 50 000 records (reviews de films) utilisés pour la classification des sentiments. Chaque record contient deux colonnes : review et sentiment. La colonne de review contient le texte du review et la colonne de sentiment contient le sentiment correspondant. La colonne de sentiment peut avoir deux valeurs, à savoir "positive" et "négative", ce qui fait de notre problème un problème de classification binaire.

Importation des bibliothèques :

```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords

from numpy import array
from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.core import Activation, Dropout, Dense
from keras.layers import Flatten
from keras.layers import Conv1D
from keras.layers import GlobalMaxPooling1D
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
```

Importation du dataset :

Importer IMDB dataset en utilisant la méthode read_csv de pandas. Quelle est la dimention du dataset ? Contient-il des valeurs NULL ? Affichez les 5 premières lignes. Utiliser la méthode countplot de la librairie seaborn pour voir si le dataset est équilibré ou non ?

Afficher le 4ème review, que remarquez-vous ?

Prétraitement de données :

```
def preprocess_text(sen):
    # Removing html tags
    sentence = remove_tags(sen)
    # Remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sentence)
    # Single character removal
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)
    # Removing multiple spaces
    sentence = re.sub(r'\s+', ' ', sentence)
    return sentence

TAG_RE = re.compile(r'<[^>]+>')
def remove_tags(text):
    return TAG_RE.sub('', text)
```

1. Exécutez le code ci-dessus et expliquez-le.
2. Prétraitez les reviews et stockez-les dans une nouvelle liste comme indiqué ci-dessous.
3. Réafficher le 4ème review. Que remarquez-vous ?

```
X = []
sentences = list(movie_reviews['review'])
for sen in sentences:
    X.append(preprocess_text(sen))
```

4. Nous devons convertir nos étiquettes en chiffres. Puisque nous n'avons que deux étiquettes dans la sortie, à savoir "positive" et "négative". Nous pouvons simplement les convertir en nombres entiers en remplaçant "positif" par le chiffre 1 et négatif par le chiffre 0 comme indiqué ci-dessous :

```
y = movie_reviews['sentiment']
y = np.array(list(map(lambda x: 1 if x=="positive" else 0, y)))
```

5. Diviser le dataset en training et test sets.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=42)
```

La couche de l’embedding

Dans un premier temps, nous allons utiliser la classe Tokenizer du module keras.preprocessing.text pour créer un dictionnaire word-to-index. Dans le dictionnaire word-to-index, chaque mot du corpus est utilisé comme clé, tandis qu'un

index unique correspondant est utilisé comme valeur pour la clé. Exécutez le script suivant :

```
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X_train)

X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)
```

X_train contient 40000 listes correspondant aux phrases où chacune contient des entiers. La taille de chaque liste est différente car les phrases possèdent des tailles différentes. Le script ci-dessous trouve la taille du vocabulaire, puis effectue un remplissage sur l'ensemble d'entraînement et de test.

```
# Adding 1 because of reserved 0 index
vocab_size = len(tokenizer.word_index) + 1
maxlen = 100
X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
```

Les listes ont la même longueur, c'est-à-dire 100. De plus, la variable vocabulary_size contient maintenant une valeur 92547, ce qui signifie que le corpus contient 92547 mots uniques.

Nous utiliserons GloVe pour créer notre matrice de features. Dans le script suivant, nous chargeons GloVe et créons un dictionnaire qui contiendra des mots en tant que clés et leur liste d'embeddings correspondante en tant que valeurs.

```
from numpy import array
from numpy import asarray
from numpy import zeros

embeddings_dictionary = dict()
glove_file = open("path…/glove.6B.100d.txt", encoding="utf8")

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:],dtype='float32')
    embeddings_dictionary [word] = vector_dimensions
glove_file.close()
```

Enfin, nous allons créer une matrice d'embeddings où chaque numéro de ligne correspondra à l'index du mot dans le corpus. La matrice aura 100 colonnes où chaque colonne contiendra les embeddings GloVe pour les mots de notre corpus.

```
embedding_matrix = zeros((vocab_size, 100))
for word, index in tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector
```

A. Classification avec un simple réseau de neurones

1) Créer le modèle qui respecte les paramètres suivants :

1. *Select keras model type"*: `sequential`
2. pour *"LAYER"*:
    a. *"1: LAYER"*:
        i. *"Choose the type of layer"*: `Embedding`

```
Embedding(vocab_size, 100, weights=[embedding_matrix],
input_length=maxlen , trainable=False)
```

    b. *"2: LAYER"*:
        i. *"Choose the type of layer"*: `Flatten`
    c. *"3: LAYER"*:
        i. *"Choose the type of layer"*: `Dense`
            1. *"units"*: `1`
            2. *"Activation function"*: `sigmoid`
3. pour *"Compile Parameters"*:
    a. *"Select a loss function"*: `binary_crossentropy`
    b. *"Select an optimizer"*: `adam`
    c. *"Select metrics"*: `acc`
4. pour *"Fit Parameters"*:
    a. *"epochs"*: `6`
    b. *"batch_size"*: `128`
    c. *"Verbose"*: `1`
    d. *"validation split"*: `0.2`

Le résultat doit ressembler à :

```
Epoch 1/6
32000/32000 [==============================] - 2s 52us/step - loss: 0.6076 - acc: 0.6713 - val
_loss: 0.5331 - val_acc: 0.7356
Epoch 2/6
32000/32000 [==============================] - 1s 38us/step - loss: 0.4979 - acc: 0.7627 - val
 loss: 0.5180 - val acc: 0.7425
Epoch 3/6
32000/32000 [==============================] - 1s 40us/step - loss: 0.4631 - acc: 0.7845 - val
_loss: 0.5280 - val_acc: 0.7393
Epoch 4/6
32000/32000 [==============================] - 1s 37us/step - loss: 0.4378 - acc: 0.7979 - val
_loss: 0.5548 - val_acc: 0.7280
Epoch 5/6
32000/32000 [==============================] - 1s 37us/step - loss: 0.4246 - acc: 0.8046 - val
 loss: 0.5218 - val acc: 0.7451
Epoch 6/6
32000/32000 [==============================] - 1s 36us/step - loss: 0.4095 - acc: 0.8142 - val
_loss: 0.5380 - val_acc: 0.7421
```

2) Évaluer le modèle en utilisant la méthode `evaluate` qui renvoie les scores de « loss » et « accuracy »

```
score = model.evaluate(X_test, y_test, verbose=1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

3) Essayons de tracer les fonctions de loss et de accuracy pour les training et test sets. Que remarquez-vous ?

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```

B. Classification avec un réseau de neurones convolutionnel

1) Créer le modèle qui respecte les paramètres suivants :

1. *Select keras model type"*: `sequential`

2. pour *"LAYER"*:
   "1: LAYER":
   "Choose the type of layer": `Embedding`

```
Embedding(vocab_size, 100, weights=[embedding_matrix],
input_length
```

   "2: LAYER":
   "Choose the type of layer": `Convolutional - Conv1D`
   a. *"filters"*: `128`
   b. *"kernel_size"*: `5`
   c. *"Activation function"*: `relu`
   "3: LAYER":
   "Choose the type of layer": `Pooling - MaxPooling1D`
   "4: LAYER":
   "Choose the type of layer": `Dense`
   a. "units": `1`
   b. "Activation function": `sigmoid`

3. pour *"Compile Parameters"*:
   a. *"Select a loss function"*: `binary_crossentropy`
   b. *"Select an optimizer"*: `adam`
   c. *"Select metrics"*: `acc`
4. pour *"Fit Parameters"*:
   a. *"epochs"*: `6`
   b. *"batch_size"*: `128`
   c. *"Verbose"*: `1`
   d. *"validation split"*: `0.2`

2) Evaluer le modèle.
3) Tracer les fonctions « loss » et « accuracy » pour les training et test sets.

C. Classification avec un réseau de neurones récurrent (LSTM)

1) Créer le modèle qui respecte les paramètres suivants :

1. *Select keras model type"*: `sequential`

5. pour *"LAYER"*:
   "1: LAYER":
   "Choose the type of layer": `Embedding`
   `Embedding(vocab_size, 100, weights=[embedding_matrix], input_length`
   "2: LAYER":
   "Choose the type of layer": `Recurrent -LSTM`
   a. *"units"*: `128`
   "3: LAYER":
   "Choose the type of layer": `Dense`
   a. "units": `1`
   b. "Activation function": `sigmoid`

6. pour *"Compile Parameters"*:
   a. *"Select a loss function"*: `binary_crossentropy`
   b. *"Select an optimizer"*: `adam`
   c. *"Select metrics"*: `acc`
7. pour *"Fit Parameters"*:
   a. *"epochs"*: `6`
   b. *"batch_size"*: `128`
   c. *"Verbose"*: `1`
   d. *"validation split"*: `0.2`

2) Evaluer le modèle.
3) Tracer les fonctions « loss » et « accuracy » pour les training et test sets.
4) Comparer les résultats des 3 classifieurs
5) Choisir une instance aléatoirement et donner sa polarité.
6) Que remarquez-vous ?

D. Compte rendu :

1. Remplir le tableau suivant par les valeurs de accuracy correspondantes au test set :

Word embedding / Classifieur	Glove	Wor2vec	Fasttext	Tf-idf
LSTM				
GRU				
RNN + CNN				

2. Décrivez vos choix et interpréter les résultats.