RAPPORT DE PROJET : GOODREADS BOOKS REVIEWS

4IABD1

ANNA DIAW

NADEJDA DOROSENCO

ADEM AIT IDIR

IMPORT DES DONNÉES :

Nous avons commencé par charger les fichiers de traitement, test et soumission dans Google.drive pour aller plus vite puis on les importe au début du modèle.

```
from google.colab import drive
drive.mount('/content/drive/')
```

Nous les lisons ensuite avec la bibliothèque pandas et chargeas les fichiers de données d'entraînement et de test à partir de Google Drive. Les données d'entraînement sont utilisées pour entraîner le modèle, tandis que les données de test sont utilisées pour évaluer les performances du modèle

```
train_df = pd.read_csv('/content/drive//MyDrive/goodreads_train.csv')
test_df = pd.read_csv('/content/drive//MyDrive/goodreads_test.csv')
```

MODELES :

Les modèle MLP, CNN, RESNET et RNN ont les mêmes processus de prétraitement qui est le suivant :

- Nous utilisons la bibliothèque NLTK pour supprimer les “stopwords" anglais avec la fonction remove_stopwords qui prend en entrée un texte et supprime tous les mots qui se trouvent dans la liste de stopwords.

- maxlen et max-words définissent les variables maxler et maxwords, qui représentent respectivement la longueur maximale des commentaires et le nombre maximum de mots uniques dans le dictionnaire de mots.

- Nous créons un obiet tokenizer en utilisant la classe Tokenizer de Keras. num words est utilisé pour définir le nombre maximum de mots uniques à prendre en compte dans le dictionnaire de mots.

- Ensuite nous utilisons tokenizer.fit_on_texts() qui crée le dictionnaire de mots avec la méthode fit_on_texts() de tokenizer. Elle crée un dictionnaire de mots unique à partir de ces textes.

- Puis nous faisons appel à la méthode tokenizer.texts_to_sequences qui transforme chaque commentaires en une séquence de nombres entiers.

- Ensuite, nous utilisons tokenizer.word_index pour récupèrer le dictionnaire de mots créé précédemment à partir de la méthode que nous stockons dans la variable word_index.

- Enfin, pad_sequences(trainsequences,padding-'post',maxlen-500) crée une séquence d'entiers de même longueur en utilisant la méthode

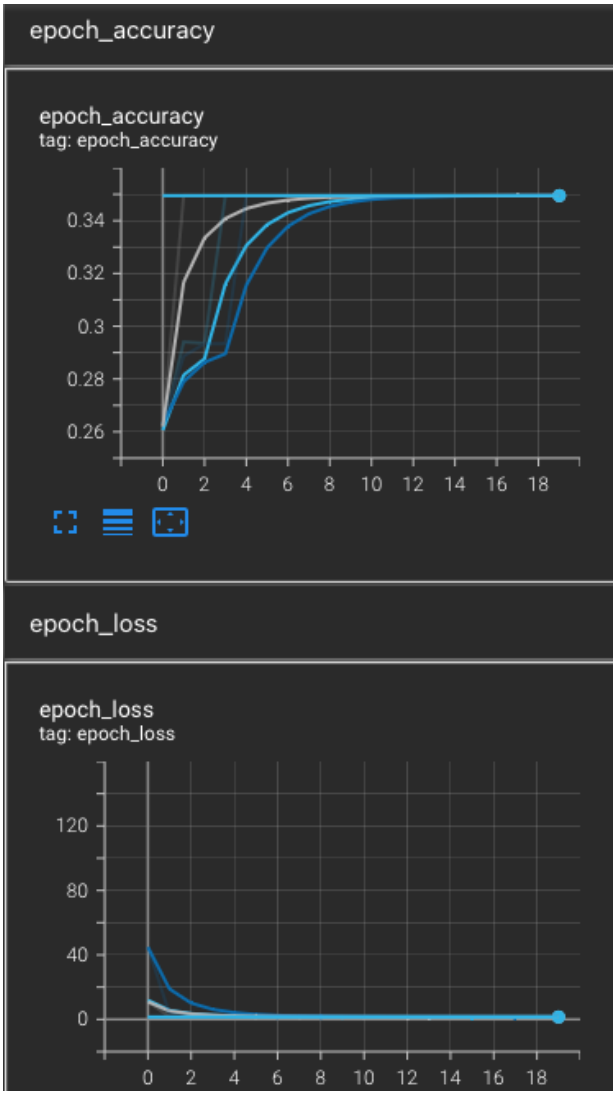
```
[ ]  ## Supprimer les "Stopwords"

     from nltk.corpus import stopwords
     nltk.download('stopwords')

     stop = set(stopwords.words("english"))
     def remove_stopwords(text):
       text = [word.lower() for word in text.split() if word.lower() not in stop]
       return " ".join(text)

     text["review_text"] = text["review_text"].map(remove_stopwords)
```

```
     %%time
     ## Commencer le traitement de la donnée

     maxlen = 500
     max_words = 10000
     tokenizer = Tokenizer(num_words = max_words)
     tokenizer.fit_on_texts(text.review_text)
     trainsequences = tokenizer.texts_to_sequences(dfre.review_text)
     testsequences = tokenizer.texts_to_sequences(dfret.review_text)
```

```
[ ]  %%time
     word_index = tokenizer.word_index
     train = pad_sequences(trainsequences,padding = 'post', maxlen=500 )
     test = pad_sequences(testsequences,padding = 'post', maxlen=500)
     labels = ohe.fit_transform(dftr[["rating"]]).toarray()
     print('Found %s unique tokens.' % len(word_index))
     print('Shape of train data:', train.shape)
     print("Shape of test data:",test.shape)
     print('Shape of label tensor:', labels.shape)
```

1_ MLP :

```
from keras.models import Sequential
from keras.layers import Dense,Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers import SGD
from keras.regularizers import l2

model = Sequential()
model.add(Dense(64, input_dim=maxlen, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(32, input_dim=maxlen, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(6, activation='softmax'))
model.compile(loss="categorical_crossentropy", optimizer=Adam(0.001), metrics=['accuracy'])
model.summary()
```

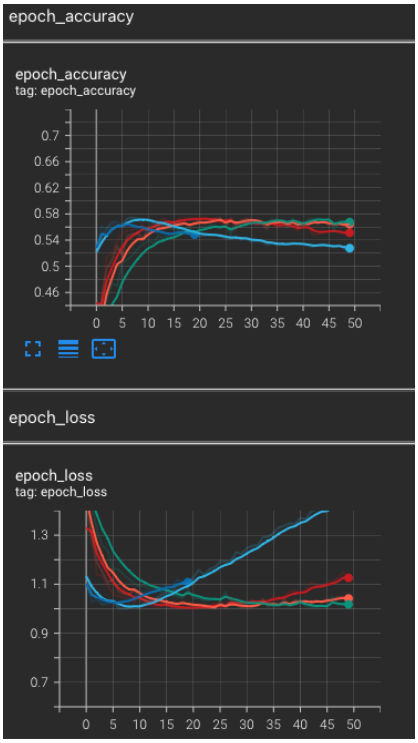


Superposition de validation accuracy

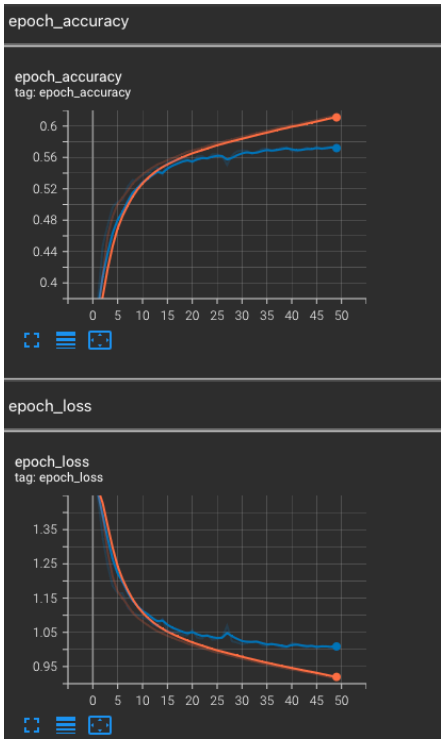
MEILLEURE VALIDATION ACCURACY : 0.3495

2_ CNN :

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, Dense, Dropout, Flatten
from tensorflow.keras.optimizers import SGD

model = Sequential()
model.add(Embedding(input_dim=max_words,output_dim=128,input_length=maxlen))
model.add(Conv1D(filters=64,kernel_size=3,padding='same',activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(50,activation='relu'))
model.add(Dense(6,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer=SGD(learning_rate=0.03, momentum=0.9),metrics=['accuracy'])
model.summary()
```



Superposition de validation accuracy

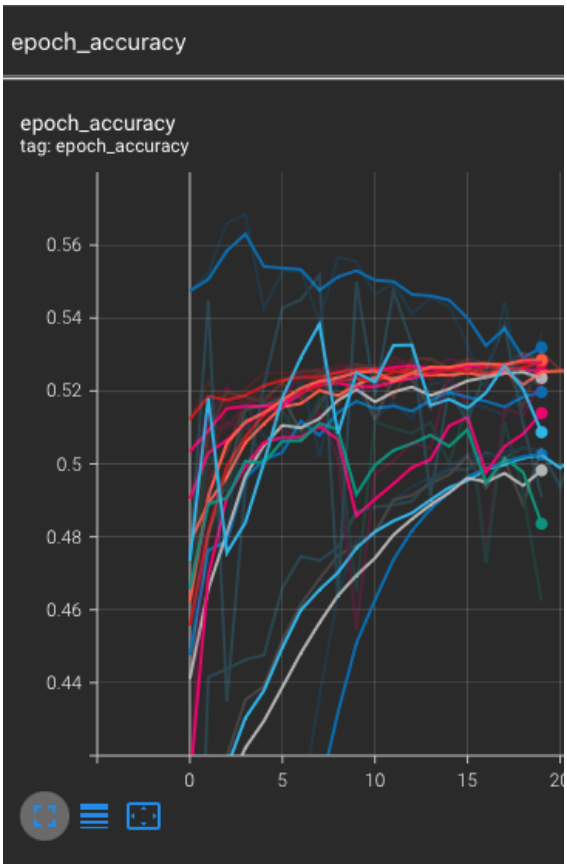


Meilleur train et validation accuracy

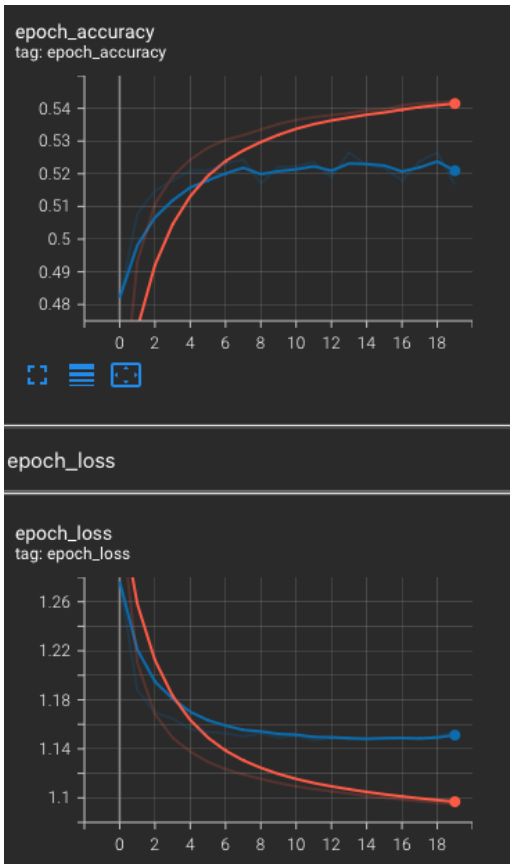
MEILLEURE VALIDATION ACCURACY : 0.5705

3_ RESNET :

```
from tensorflow.keras.layers import Input, Embedding, Conv1D, BatchNormalization, Activation, Add, MaxPooling1D, GlobalAveragePooling1D, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam


def residual_block(inputs, filters, kernel_size):
    res = Conv1D(filters=filters, kernel_size=kernel_size, padding='valid')(inputs)
    res = BatchNormalization()(res)
    shortcut = inputs
    res = Add()([res, shortcut])
    res = Activation('relu')(res)
    return res

input_layer = Input(shape=(maxlen,))

embedding_layer = Embedding(input_dim=max_words, output_dim=128, input_length=maxlen)(input_layer)

filters = 8
kernel_size = 1

res = Conv1D(filters=filters, kernel_size=kernel_size, padding='valid')(embedding_layer)
res = BatchNormalization()(res)
res = Activation('relu')(res)
res = residual_block(res, filters, kernel_size)
res = MaxPooling1D(pool_size=2)(res)
res = GlobalAveragePooling1D()(res)

output_layer = Dense(6, activation='softmax')(res)

model = Model(inputs=input_layer, outputs=output_layer)

model.compile(loss='categorical_crossentropy', optimizer=Adam(0.001), metrics=['accuracy'])

model.summary()
```



Superposition de validation accuracy



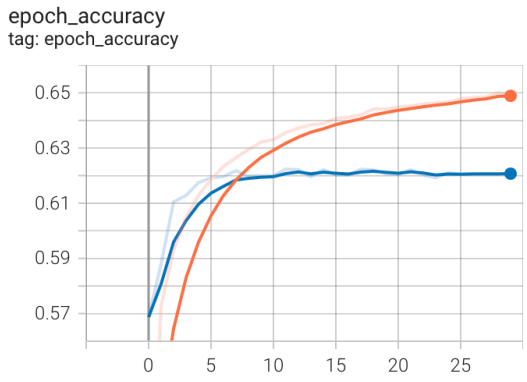
Meilleur train et validation accuracy

MEILLEURE VALIDATION ACCURACY : 0.5209

4_ RNN (LSTM) :

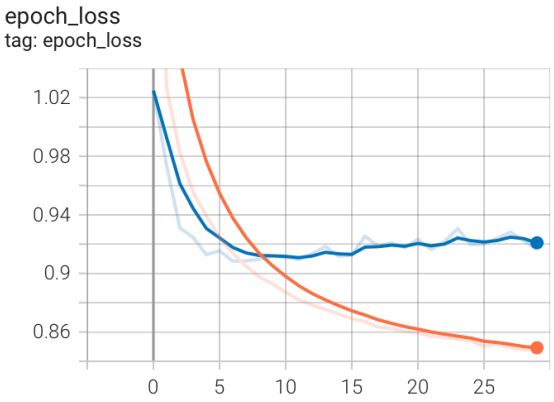
```
## Le Modele

from keras.models import Sequential
from keras.layers import Dense,Dropout,Embedding,Bidirectional,LSTM
from tensorflow.keras.optimizers import Adam
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint
from keras.regularizers import l2


model = Sequential()
model.add(Embedding(10000,64))
model.add(Bidirectional(LSTM(128,dropout = 0.3,return_sequences=True)))
model.add(Bidirectional(LSTM(64,dropout = 0.3)))
model.add(Dense(64,activation='relu',kernel_regularizer=l2(0.01)))
model.add(Dropout(0.5))
model.add(Dense(6,activation="softmax"))
model.compile(loss="categorical_crossentropy", optimizer=Adam(0.005), metrics=['accuracy'])
model.summary()
```

epoch_accuracy



Superposition de validation accuracy

epoch_loss



Superposition de validation accuracy

MEILLEURE VALIDATION ACCURACY : 0.6

5_ TRANSFORMERS :

Ensuite, nous instancions un objet "tokenizer" à partir du modèle BERT pré-entraîné "bert-base-uncased". Le tokenizer est utilisé pour convertir les textes en entrée en représentations numériques

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

Nous créons ensuite une fonction encode_text qui est utilisée pour convertir les textes d'entrée en représentations numériques prêtes à être utilisées par le modèle BERT pour l'entrainement. Elle utilise le tokenizer créé précédemment pour convertir chaque texte en une séquence d'identifiants de token ("input_ids") et une séquence binaire d'indicateurs d'attention ("attention_masks").

Ensuite, le modèle BERT est défini à l'aide de la fonction build_model(). Cette fonction utilise la classe TFBertModel de la bibliothèque Transformers pour charger le modèle BERT pré-entraîné bert-base-uncased. Le modèle BERT prend en entrée deux tenseurs : input_ids et attention_mask. Les tenseurs sont définis à l'aide de la classe Input de Keras.

Le tenseur input_ids contient les identifiants des tokens générés à partir des textes d'entrée. Le tenseur attention_mask est utilisé pour indiquer à BERT quels tokens du texte d'entrée doivent être ignorés lors du traitement. Les tokens ignorés sont remplacés par des zéros. Après l'entrée, le tenseur input_ids et le tenseur attention_mask sont passés au modèle BERT, qui renvoie une sortie sous la forme d'un tuple contenant deux tenseurs. Le premier tenseur contient la représentation de l'ensemble du texte en sortie de BERT, et le deuxième tenseur contient la représentation de la première position (ou token) du texte en sortie de BERT.

La sortie du modèle BERT est ensuite passée à une couche de sortie dense (Dense) qui prédit la note donnée au livre. La fonction d'activation de cette couche est linear car il s'agit d'une tâche de régression et non de classification. Le modèle est compilé à l'aide de la fonction compile(). Le paramètre loss est défini sur mse pour indiquer qu'il s'agit d'une tâche de régression et que la fonction de perte à utiliser est la MSE (Mean Squared Error). L'optimiseur utilisé est Adam avec un taux d'apprentissage de 1e-5. Ensuite, les données d'entrée et de sortie sont remodelées (reshape()) pour qu'elles aient la forme attendue par le modèle. Le modèle est entraîné à l'aide de la fonction fit(). Les données d'entrée sont les tenseurs train_input_ids et train_attention_masks, et les données de sortie sont les notes des livres dans train_df. Le paramètre validation_split est défini sur 0.2 pour indiquer que 20% des données d'entraînement doivent être utilisées pour la validation. Le modèle est entraîné pendant 10 époques avec une taille de lot (batch_size) de 32.

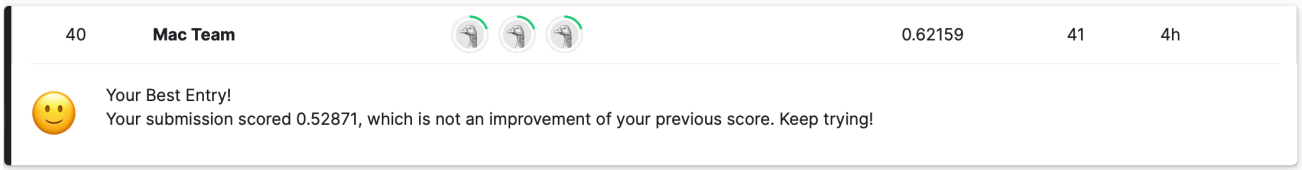
Après l'entraînement, le modèle est utilisé pour faire des prédictions sur les données de test (test_input_ids et test_attention_masks) à l'aide de la fonction predict(). Les prédictions sont enregistrées dans un tableau predictions.

```
[ ]  # Définition du modèle BERT
     def build_model():
         bert_model = TFBertModel.from_pretrained('bert-base-uncased')
         input_ids = Input(shape=(512,), dtype=tf.int32, name='input_ids')
         attention_mask = Input(shape=(512,), dtype=tf.int32, name='attention_mask')
         bert_output = bert_model({'input_ids': input_ids, 'attention_mask': attention_mask})
         output = Dense(6, activation='linear')(bert_output[1])
         model = Model(inputs=[input_ids, attention_mask], outputs=output)
         model.compile(loss='mse', optimizer=Adam(lr=1e-5))
         return model

[ ]  train_input_ids = train_input_ids.reshape(train_input_ids.shape[0], -1)
     train_attention_masks = train_attention_masks.reshape(train_attention_masks.shape[0], -1)
     test_input_ids = test_input_ids.reshape(test_input_ids.shape[0], -1)
     test_attention_masks = test_attention_masks.reshape(test_attention_masks.shape[0], -1)

[ ]  # Entraînement du modèle
     model = build_model()
     model.fit(
         [train_input_ids, train_attention_masks],
         train_df['rating'].values,
         validation_split=0.2,
         epochs=10,
         batch_size=32
     )

[ ]  # Prédiction sur les données de test
     predictions = model.predict([test_input_ids, test_attention_masks]).flatten()
```

PREDICTIONS :

Enfin, les prédictions sont enregistrées dans un fichier CSV à l'aide de la fonction to_csv() de Pandas. Le fichier CSV est enregistré sous le nom MLP_submission.csv dans le répertoire Google Drive.

CLASSEMENT KAGGLE :



Nous nous retrouvons à la 40e place sur 226 équipes