import pandas as pd

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.layers import Input, Embedding, GlobalAveragePooling1D, Dropout, Dense

from tensorflow.keras.preprocessing import sequence

from transformer import Encoder # Assurez-vous d'importer ou de définir la classe Encoder

# ... (votre code précédent)

if \_\_name\_\_ == "\_\_main\_\_":

# Chargement des données

X\_train, Y\_train = load\_data()

# Instanciation de l'objet de configuration

config = GeneralConfig()

# Mise en forme des données

X\_train = X\_train.values.reshape(len(X\_train), config.num\_features, 1)

# Construction du modèle Transformer avec 3 blocs

inputs\_A = Input(shape=(config.sequence\_length,), dtype='int32')

embeddings = Embedding(config.max\_features, config.model.embedding\_size)(inputs\_A)

mask\_inputs = padding\_mask(inputs\_A) # Assurez-vous d'avoir la fonction padding\_mask définie

# Ajout de 3 blocs d'encodeur

num\_blocks = 3

out\_seq = embeddings # Initialisation avec les embeddings

for \_ in range(num\_blocks):

out\_seq = Encoder(

n\_layers=config.model.num\_blocks,

d\_model=config.model.embedding\_size,

num\_heads=config.model.num\_heads,

middle\_units=config.model.filters,

max\_seq\_len=config.sequence\_length)([out\_seq, mask\_inputs])

out\_seq = GlobalAveragePooling1D()(out\_seq)

out\_seq = Dropout(0.3)(out\_seq)

outputs\_A = Dense(64, activation='softmax')(out\_seq)

outputs = Dense(2, activation='sigmoid')(outputs\_A)

model = Model(inputs=inputs\_A, outputs=outputs)

print(model.summary())

# Compilation du modèle

opt = Adam(lr=0.0001, decay=0.00001)

loss = 'sparse\_categorical\_crossentropy'

model.compile(loss=loss,

optimizer=opt,

metrics=['accuracy'])

print('Entraînement...')

history = model.fit(X\_train, Y\_train,

batch\_size=config.batch\_size,

epochs=100,

validation\_split=0.2)

L'ajout de plusieurs blocs d'encodeur est une caractéristique du modèle de transformer pour améliorer la représentation apprise des séquences. Chaque bloc d'encodeur comprend plusieurs couches d'attention multi-têtes suivies de couches d'alimentation avant (feedforward). Cela permet au modèle de capturer des motifs complexes dans les séquences.

Explication du code :

1. `num\_blocks = 3`: Cela définit le nombre de blocs d'encodeur que vous souhaitez ajouter à votre modèle. Dans cet exemple, nous ajoutons trois blocs.

2. `out\_seq = embeddings`: Au début, `out\_seq` est initialisé avec les embeddings de la séquence en entrée. Cela représente la séquence après la couche d'embedding.

3. `for \_ in range(num\_blocks)`: C'est une boucle qui ajoute plusieurs blocs d'encodeur. Le nombre de blocs est déterminé par la valeur de `num\_blocks`.

4. `Encoder(...)`: À chaque itération de la boucle, un nouveau bloc d'encodeur est ajouté à `out\_seq`. Chaque bloc d'encodeur est défini par la classe `Encoder` avec les paramètres spécifiés.

- `n\_layers=config.model.num\_blocks`: Cela spécifie le nombre de couches dans chaque bloc d'encodeur. Vous pouvez ajuster cela en fonction de vos besoins.

- `d\_model=config.model.embedding\_size`: Cela spécifie la dimension de l'espace de représentation pour chaque mot. Cela devrait correspondre à la dimension de vos embeddings.

- `num\_heads=config.model.num\_heads`: Cela spécifie le nombre de têtes dans la couche d'attention multi-têtes.

- `middle\_units=config.model.filters`: Cela spécifie le nombre d'unités dans la couche d'alimentation avant.

- `max\_seq\_len=config.sequence\_length`: Cela spécifie la longueur maximale de la séquence. Vous pouvez ajuster cela en fonction de la longueur maximale de vos séquences.

En ajoutant plusieurs blocs d'encodeur, le modèle peut apprendre des représentations hiérarchiques et complexes des séquences, ce qui peut améliorer ses performances pour certaines tâches.