Language translator

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

from keras.preprocessing.text import Tokenizer

import tensorflow as tf

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.utils import to\_categorical

from keras.models import Sequential, load\_model

from keras.layers import Embedding, Dense, LSTM, Dropout, TimeDistributed, RepeatVector

from collections import Counter

from wordcloud import WordCloud

from sklearn.model\_selection import train\_test\_split

from keras.callbacks import EarlyStopping, ReduceLROnPlateau

from keras.losses import SparseCategoricalCrossentropy

df = pd.read\_csv("e.csv", encoding="latin-1")

df.head()

eng = df['English words/sentences']

fra = df['French words/sentences']

eng\_word\_counter = Counter([word for sentence in eng for word in sentence.split()])

print("Total count of English words:",len([word for sentence in eng for word in sentence.split()]))

print("Count of distinct English words:",len(eng\_word\_counter))

print("10 most common English words:",list(zip(\*eng\_word\_counter.most\_common(10)))[0])

fra\_word\_counter = Counter([word for sentence in fra for word in sentence.split()])

print("Total count of French words:",len([word for sentence in fra for word in sentence.split()]))

print("Count of distinct French words:",len(fra\_word\_counter))

print("10 most common French words:",list(zip(\*fra\_word\_counter.most\_common(10)))[0])

plt.figure(figsize=(12,8))

wc = WordCloud(width=600,height=300).generate(' '.join(eng))

plt.imshow(wc)

plt.show();

plt.figure(figsize=(12,8))

wc = WordCloud(width=600,height=300).generate(' '.join(fra))

plt.imshow(wc)

plt.show();

def word\_count(line):

  return len(line.split())

df['English\_word\_count'] = df['English words/sentences'].apply(lambda x: word\_count(x))

df['French\_word\_count'] = df['French words/sentences'].apply(lambda x: word\_count(x))

fig, axes = plt.subplots(nrows=1,ncols=2)

sns.distplot(df['English\_word\_count'],ax=axes[0])

sns.distplot(df['French\_word\_count'],ax=axes[1])

sns.despine()

plt.show();

def create\_tokenizer(sentences):

  tokenizer = Tokenizer()

  tokenizer.fit\_on\_texts(sentences)

  return tokenizer

def max\_sentence\_length(lines):

  return max(len(sentence.split()) for sentence in lines)

def encode\_sequences(tokenizer,sentences,max\_sent\_len):

  text\_to\_seq = tokenizer.texts\_to\_sequences(sentences) # encode sequences with integers

  text\_pad\_seq = pad\_sequences(text\_to\_seq,maxlen=max\_sent\_len,padding='post') # pad sequences with 0

  return text\_pad\_seq

# Prepare English tokenizer

eng\_tokenizer = create\_tokenizer(eng)

eng\_vocab\_size = len(eng\_tokenizer.word\_index) + 1

max\_eng\_sent\_len = max\_sentence\_length(eng)

print("English vocabulary size:", eng\_vocab\_size)

print("Maximum length of English sentences:", max\_eng\_sent\_len)

# Prepare French tokenizer

fra\_tokenizer = create\_tokenizer(fra)

fra\_vocab\_size = len(fra\_tokenizer.word\_index) + 1

max\_fra\_sent\_len = max\_sentence\_length(fra)

print("French vocabulary size:", fra\_vocab\_size)

print("Maximum length of French sentences:", max\_fra\_sent\_len)

max\_eng\_sent\_len = 22

max\_fra\_sent\_len = 22

X = encode\_sequences(eng\_tokenizer,eng,max\_eng\_sent\_len)

y = encode\_sequences(fra\_tokenizer,fra,max\_fra\_sent\_len)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.15,shuffle=True,random\_state=101)

def create\_model(src\_vocab\_size,dest\_vocab\_size,src\_timesteps,dest\_timesteps,no\_units):

  model = Sequential()

  model.add(Embedding(src\_vocab\_size,no\_units,input\_length=src\_timesteps,mask\_zero=True))

  model.add(LSTM(no\_units))

  model.add(RepeatVector(dest\_timesteps))

  model.add(LSTM(no\_units,return\_sequences=True))

  model.add(TimeDistributed(Dense(1024,activation='relu')))

  model.add(Dropout(0.2))

  model.add(TimeDistributed(Dense(dest\_vocab\_size,activation='softmax')))

  return model

model = create\_model(eng\_vocab\_size,fra\_vocab\_size,max\_eng\_sent\_len,max\_fra\_sent\_len,512)

model.summary()

model.compile(loss=SparseCategoricalCrossentropy(),optimizer='rmsprop',metrics='accuracy')

es = EarlyStopping(monitor='val\_accuracy',patience=5,mode='max',verbose=1)

lr = ReduceLROnPlateau(monitor='val\_accuracy',patience=3,mode='max',verbose=1,factor=0.1,min\_lr=0.001)

r = model.fit(X\_train,

              y\_train.reshape(y\_train.shape[0],y\_train.shape[1],1),

              epochs=10,

              batch\_size=512,

              callbacks=[es,lr],

              validation\_data=(X\_test,y\_test.reshape(y\_test.shape[0],y\_test.shape[1],1)))

model.evaluate(X\_test,y\_test.reshape(y\_test.shape[0],y\_test.shape[1],1))

model.save('english\_to\_french\_translator.h5')

translator\_model = load\_model('english\_to\_french\_translator.h5')

translator\_model

ef translate\_sentence(sentence, eng\_tokenizer, fra\_tokenizer, max\_eng\_sent\_len, max\_fra\_sent\_len, translator\_model):

    # Tokenize and encode the input sentence

    sentence\_seq = eng\_tokenizer.texts\_to\_sequences([sentence])

    padded\_sentence = pad\_sequences(sentence\_seq, maxlen=max\_eng\_sent\_len, padding='post')

    # Get the model prediction for the input sentence

    predicted\_seq = translator\_model.predict(padded\_sentence)

    # Decode the predicted sequence into French

    translated\_sentence = ''

    for token in predicted\_seq[0]:

        sampled\_token\_index = np.argmax(token)

        if sampled\_token\_index == 0:

            break

        word = fra\_tokenizer.index\_word[sampled\_token\_index]

        translated\_sentence += ' ' + word

    return translated\_sentence.strip()

# Example usage:

english\_sentence = "How are you doing today?"

translated\_sentence = translate\_sentence(english\_sentence, eng\_tokenizer, fra\_tokenizer, max\_eng\_sent\_len, max\_fra\_sent\_len, translator\_model)

print(f"English: {english\_sentence}")

print(f"Translated French: {translated\_sentence}")