In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.moun
t("/content/drive", force_remount=True).
```

Neural Machine Translation

Machine Translation is the task of converting a source text given in one language to text in another language.

Machine Translation is considered as a difficult task because of ambiguity and flexibility of human language.

Classical machine translation methods often involve rules for converting text in the source language to the target language. One such method is Neural Machine Translation.

Here, English -> Hindi

Here, we will be using Neural Machine Translation method for the conversion of source language as English to target langauage Hindi.

Neural Machine translation systems are also known as End to End systems because only one model is required for performing translation.

About the corpus: Corpus is taken from https://www.clarin.eu/resource-families/parallel-corpora. The corpus contains english and hindi sentences chosen from different TED talks, news and wikipedia articles. It has 3 columns and 127607 rows. First we will perform some data preprocessing on the dataset before applying the model as the dataset contains null and duplicate values.

Preprocessing steps:

1. Checking for null values
2. Checking for duplicate values
3. Converting to lower case
4. Removing special characters
5. Removing digits
6. Removing quotes
7. Adding start and end token
8. Adding columns to store number of tokens

In [2]:
```
import pandas as pd
import numpy as np
```

In [3]:
```
df = pd.read_csv('/content/drive/MyDrive/Hindi_English_Truncated_Corpus.csv')
df.head()
```

Out[3]:

	source	english_sentence	hindi_sentence
0	ted	politicians do not have permission to do what ...	राजनीतिज्ञों के पास जो कार्य करना चाहिए, वह कर...

	source	english_sentence	hindi_sentence
1	ted	I'd like to tell you about one such child,	मई आपको ऐसे ही एक बच्चे के बारे में बताना चाहू...
2	indic2012	This percentage is even greater than the perce...	यह प्रतिशत भारत में हिन्दुओं प्रतिशत से अधिक है।
3	ted	what we really mean is that they're bad at not...	हम ये नहीं कहना चाहते कि वो ध्यान नहीं दे पाते
4	indic2012	.The ending portion of these Vedas is called U...	इन्हीं वेदों का अंतिम भाग उपनिषद कहलाता है।

In [4]:
```
df = df.drop(['source'], axis=1)
```

In [5]:
```
df.head()
```

Out[5]:

	english_sentence	hindi_sentence
0	politicians do not have permission to do what ...	राजनीतिज्ञों के पास जो कार्य करना चाहिए, वह कर...
1	I'd like to tell you about one such child,	मई आपको ऐसे ही एक बच्चे के बारे में बताना चाहू...
2	This percentage is even greater than the perce...	यह प्रतिशत भारत में हिन्दुओं प्रतिशत से अधिक है।
3	what we really mean is that they're bad at not...	हम ये नहीं कहना चाहते कि वो ध्यान नहीं दे पाते
4	.The ending portion of these Vedas is called U...	इन्हीं वेदों का अंतिम भाग उपनिषद कहलाता है।

In [6]:
```
df.shape
```

Out[6]:
```
(127607, 2)
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
english_sentence    2
hindi_sentence      0
dtype: int64
```

In [8]:
```
df = df.dropna()
```

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
english_sentence    0
hindi_sentence      0
dtype: int64
```

In [10]:
```
df.duplicated(keep='first').sum()
```

Out[10]:
```
2780
```

In [11]:
```
df = df.drop_duplicates()
```

In [12]:
```
df.duplicated(keep='first').sum()
```

Out[12]:
```
0
```

In [13]:
```
df.shape
```

Out[13]:
```
(124825, 2)
```

In [14]:
```
# converting to lower case
```

In [15]:
```
df['english_sentence'] = df['english_sentence'].apply(lambda s: s.lower())

df['hindi_sentence'] = df['hindi_sentence'].apply(lambda s: s.lower())
```

In [16]:
```
# removing quotes
```

In [17]:
```
import re

df['english_sentence'] = df['english_sentence'].apply(lambda s: re.sub("'", '', s))
df['hindi_sentence'] = df['hindi_sentence'].apply(lambda s: re.sub("'", '',  s))
```

In [18]:
```
import string

special_chars = set(string.punctuation)

# removing special characters

df['english_sentence'] = df['english_sentence'].apply(lambda x: ''.join(ch for ch in x
df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x: ''.join(ch for ch in x if c
```

In [19]:
```
import string
digits = str.maketrans('', '', string.digits)

df['english_sentence'] = df['english_sentence'].apply(lambda x: x.translate(digits))
df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x: x.translate(digits))
df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x: re.sub("[२३०८१५७९४६]", "",
```

In [20]:
```
# ADDING START AND END TOKENS
```

In [21]:
```
df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x: '<START>' + x + ' <END>')
```

In [22]:
```
df.head()
```

Out[22]:

	english_sentence	hindi_sentence
0	politicians do not have permission to do what ...	<START>राजनीतिज्ञों के पास जो कार्य करना चाहिए...
1	id like to tell you about one such child	<START>मई आपको ऐसे ही एक बच्चे के बारे में बता...
2	this percentage is even greater than the perce...	<START>यह प्रतिशत भारत में हिन्दुओं प्रतिशत से...
3	what we really mean is that theyre bad at not ...	<START>हम ये नहीं कहना चाहते कि वो ध्यान नहीं ...
4	the ending portion of these vedas is called up...	<START>इन्हीं वेदों का अंतिम भाग उपनिषद कहलाता...

In [23]:
```
all_eng_words = set()
for eng in df['english_sentence']:
    for word in eng.split():
        if word not in all_eng_words:
            all_eng_words.add(word)

all_hindi_words = set()
for hin in df['hindi_sentence']:
    for word in hin.split():
```

```
            if word not in all_hindi_words:
                all_hindi_words.add(word)
```

In [24]:
```
len(all_eng_words)
```

Out[24]: 72686

In [25]:
```
len(all_hindi_words)
```

Out[25]: 92069

In [26]:
```
df['length_eng_sentence'] = df['english_sentence'].apply(lambda x:len(x.split(" ")))
df['length_hin_sentence'] = df['hindi_sentence'].apply(lambda x:len(x.split(" ")))
```

In [27]:
```
df.head()
```

Out[27]:

	english_sentence	hindi_sentence	length_eng_sentence	length_hin_sentence
0	politicians do not have permission to do what ...	<START>राजनीतिज्ञों के पास जो कार्य करना चाहिए...	12	15
1	id like to tell you about one such child	<START>मई आपको ऐसे ही एक बच्चे के बारे में बता...	9	12
2	this percentage is even greater than the perce...	<START>यह प्रतिशत भारत में हिन्दुओं प्रतिशत से...	10	10
3	what we really mean is that theyre bad at not ...	<START>हम ये नहीं कहना चाहते कि वो ध्यान नहीं ...	12	12
4	the ending portion of these vedas is called up...	<START>इन्हीं वेदों का अंतिम भाग उपनिषद कहलाता...	9	9

In [28]:
```
df[df['length_eng_sentence']>30].shape
df=df[df['length_eng_sentence']<=20]
df=df[df['length_hin_sentence']<=20]
```

In [29]:
```
df.shape
```

Out[29]: (81111, 4)

In [30]:
```
print("maximum length of Hindi Sentence ", max(df['length_hin_sentence']))
print("maximum length of English Sentence ", max(df['length_eng_sentence']))
```

```
maximum length of Hindi Sentence  20
maximum length of English Sentence  20
```

In [31]:
```
max_length_src = max(df['length_hin_sentence'])
max_length_tar = max(df['length_eng_sentence'])
```

In [32]:
```
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense
from tensorflow.keras.models import Model
```

In [33]:
```
input_words = sorted(list(all_eng_words))
target_words = sorted(list(all_hindi_words))
```

```
num_encoder_tokens = len(all_eng_words)+1
num_decoder_tokens = len(all_hindi_words)+1
num_encoder_tokens, num_decoder_tokens
```

Out[33]: (72687, 92070)

In [34]:
```
df[df['length_eng_sentence']>30].shape
```

Out[34]: (0, 4)

In [35]:
```
df = df[df['length_eng_sentence']<=20]
df = df[df['length_hin_sentence']<=20]
```

In [36]:
```
df.shape
```

Out[36]: (81111, 4)

In [37]:
```
num_decoder_tokens += 1 #for zero padding
```

In [38]:
```
input_token_index = dict([(word, i+1) for i, word in enumerate(input_words)])
target_token_index = dict([(word, i+1) for i, word in enumerate(target_words)])
```

In [39]:
```
reverse_input_char_index = dict((i, word) for word, i in input_token_index.items())
reverse_target_char_index = dict((i, word) for word, i in target_token_index.items())
```

In [40]:
```
df = shuffle(df)
df.head(10)
```

Out[40]:

	english_sentence	hindi_sentence	length_eng_sentence	length_hin_sentence
83543	if there is desire and will sanskrit can be ma...	<START>यदि इच्छाशक्ति हो तो संस्कृत को हिब्रू...	16	18
112903	and youll still go to google	<START>अब भी गूगल तक पहुँच पायेंगे। <END>	6	7
22951	walmiquiy ramayana publisher rustic book store...	<START>वाल्मीकीय रामायण प्रकाशक देहाती पुस्तक ...	7	8
127248	do things together	<START>मिलकर काम करें <END>	3	4
35404	it has royal bedroom prayer room one balcony ...	<START>इनमें राजसी शयनकक्ष प्रार्थनाकक्ष एक बर...	14	12
22004	the very idea of nationhood had completely dis...	<START>राष्ट्रीयता का विचार ही पूरी तरह गायब ह...	9	12
57378	the rest of its from the cloud	<START>इसके अलावा बाकी बादल से है <END>	7	7
78886	he immediately informed the police	<START>उन्होंने इसकी सूचना तुरंत पुलिस को दी ...	6	9

	english_sentence	hindi_sentence	length_eng_sentence	length_hin_sentence
17524	searcheses on research articles in indian univ...	<START>भारतीय विश्वविद्यालयों में संस्कृत पर आ...	10	14
99263	scale inwarrence in international terrorism	<START>वैश्विक आतंकवाद में स्केल इन्वारिंस <END>	5	6

In [41]:
```
X, y = df['english_sentence'], df['hindi_sentence']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,random_state=
X_train.shape, X_test.shape
```

Out[41]: ((64888,), (16223,))

In [42]:
```
X_train.to_pickle('X_train.pkl')
X_test.to_pickle('X_test.pkl')
```

In [43]:
```
def generate_batch(X = X_train, y = y_train, batch_size = 128):
    ''' Generate a batch of data '''
    while True:
        for j in range(0, len(X), batch_size):
            encoder_input_data = np.zeros((batch_size, max_length_src),dtype='float32')
            decoder_input_data = np.zeros((batch_size, max_length_tar),dtype='float32')
            decoder_target_data = np.zeros((batch_size, max_length_tar, num_decoder_tok
            for i, (input_text, target_text) in enumerate(zip(X[j:j+batch_size], y[j:j+
                for t, word in enumerate(input_text.split()):
                    encoder_input_data[i, t] = input_token_index[word]
                for t, word in enumerate(target_text.split()):
                    if t<len(target_text.split())-1:
                        decoder_input_data[i, t] = target_token_index[word]
                    if t>0:
                        decoder_target_data[i, t - 1, target_token_index[word]] = 1.
            yield([encoder_input_data, decoder_input_data], decoder_target_data)
```

In [44]:
```
latent_dim = 64
```

In [45]:
```
# Encoder
encoder_inputs = Input(shape=(None,))
enc_emb =  Embedding(num_encoder_tokens, latent_dim, mask_zero = True)(encoder_inputs)
encoder_lstm = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(enc_emb)
encoder_states = [state_h, state_c]
```

In [46]:
```
decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(num_decoder_tokens, latent_dim, mask_zero = True)
dec_emb = dec_emb_layer(decoder_inputs)
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(dec_emb, initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

In [47]:
```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
```

In [48]:
```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
input_1 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________
input_2 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________
embedding (Embedding)           (None, None, 64)     4651968     input_1[0][0]
__________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 64)     5892544     input_2[0][0]
__________________________________________________________________________________________
lstm (LSTM)                     [(None, 64), (None,  33024       embedding[0][0]
__________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, None, 64), ( 33024       embedding_1[0][0]
                                                                 lstm[0][1]
                                                                 lstm[0][2]
__________________________________________________________________________________________
dense (Dense)                   (None, None, 92071)  5984615     lstm_1[0][0]
==========================================================================================
Total params: 16,595,175
Trainable params: 16,595,175
Non-trainable params: 0
__________________________________________________________________________________________
```

In [49]:
```
train_samples = len(X_train)
val_samples = len(X_test)
batch_size = 128
epochs = 25
```

In [50]:
```
model.fit_generator(generator = generate_batch(X_train, y_train, batch_size = batch_siz
                    steps_per_epoch = train_samples//batch_size,
                    epochs=epochs,
                    validation_data = generate_batch(X_test, y_test, batch_size = batch
                    validation_steps = val_samples//batch_size)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1844:
UserWarning: `Model.fit_generator` is deprecated and will be removed in a future versio
n. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/25
506/506 [==============================] - 425s 766ms/step - loss: 3.6837 - val_loss: 3.
1418
Epoch 2/25
506/506 [==============================] - 394s 780ms/step - loss: 3.0984 - val_loss: 3.
0176
Epoch 3/25
506/506 [==============================] - 401s 793ms/step - loss: 2.9672 - val_loss: 2.
9180
Epoch 4/25
506/506 [==============================] - 390s 772ms/step - loss: 2.8579 - val_loss: 2.
8476
```

```
Epoch 5/25
506/506 [==============================] - 393s 778ms/step - loss: 2.7804 - val_loss: 2.
7982
Epoch 6/25
506/506 [==============================] - 392s 775ms/step - loss: 2.7150 - val_loss: 2.
7529
Epoch 7/25
506/506 [==============================] - 388s 767ms/step - loss: 2.6573 - val_loss: 2.
7187
Epoch 8/25
506/506 [==============================] - 385s 761ms/step - loss: 2.6101 - val_loss: 2.
6913
Epoch 9/25
506/506 [==============================] - 384s 760ms/step - loss: 2.5659 - val_loss: 2.
6707
Epoch 10/25
506/506 [==============================] - 378s 748ms/step - loss: 2.5262 - val_loss: 2.
6516
Epoch 11/25
506/506 [==============================] - 383s 757ms/step - loss: 2.4886 - val_loss: 2.
6345
Epoch 12/25
506/506 [==============================] - 376s 743ms/step - loss: 2.4577 - val_loss: 2.
6264
Epoch 13/25
506/506 [==============================] - 372s 736ms/step - loss: 2.4225 - val_loss: 2.
6068
Epoch 14/25
506/506 [==============================] - 367s 725ms/step - loss: 2.3937 - val_loss: 2.
6093
Epoch 15/25
506/506 [==============================] - 377s 746ms/step - loss: 2.3647 - val_loss: 2.
5895
Epoch 16/25
506/506 [==============================] - 370s 732ms/step - loss: 2.3387 - val_loss: 2.
5833
Epoch 17/25
506/506 [==============================] - 368s 728ms/step - loss: 2.3095 - val_loss: 2.
5757
Epoch 18/25
506/506 [==============================] - 367s 726ms/step - loss: 2.2857 - val_loss: 2.
5736
Epoch 19/25
506/506 [==============================] - 364s 720ms/step - loss: 2.2602 - val_loss: 2.
5666
Epoch 20/25
506/506 [==============================] - 369s 730ms/step - loss: 2.2387 - val_loss: 2.
5624
Epoch 21/25
506/506 [==============================] - 365s 722ms/step - loss: 2.2153 - val_loss: 2.
5672
Epoch 22/25
506/506 [==============================] - 365s 722ms/step - loss: 2.1936 - val_loss: 2.
5567
Epoch 23/25
506/506 [==============================] - 369s 730ms/step - loss: 2.1705 - val_loss: 2.
5544
Epoch 24/25
506/506 [==============================] - 369s 730ms/step - loss: 2.1475 - val_loss: 2.
5532
Epoch 25/25
506/506 [==============================] - 367s 726ms/step - loss: 2.1283 - val_loss: 2.
5520
```

Out[50]: <tensorflow.python.keras.callbacks.History at 0x7fdb0db1e9d0>

In [51]:
```
model.save_weights('nmt_weights.h5')
```

In [53]:
```
encoder_model = Model(encoder_inputs, encoder_states)

decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]

dec_emb2= dec_emb_layer(decoder_inputs)


decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=decoder_sta
decoder_states2 = [state_h2, state_c2]
decoder_outputs2 = decoder_dense(decoder_outputs2)

decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs2] + decoder_states2)
```

In [64]:
```
def decode_sequence(input_seq):
    states_value = encoder_model.predict(input_seq)
    target_seq = np.zeros((1,1))
    target_seq[0, 0] = target_token_index['<START>']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]
        decoded_sentence += ' '+sampled_char

        if (sampled_char == '<END>' or
           len(decoded_sentence) > 50):
            stop_condition = True

        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        states_value = [h, c]

    return decoded_sentence
```

In [77]:
```
train_gen = generate_batch(X_train, y_train, batch_size = 1)
k=-1
```

In [78]:
```
k+=1
(input_seq, actual_output), _ = next(train_gen)
decoded_sentence = decode_sequence(input_seq)
print('Input english sentence:', X_train[k:k+1].values[0])
print('Actual hindi translation:', y_train[k:k+1].values[0][7:-5])
print('Predicted hindi translation:', decoded_sentence[:-5])
```

```
Input english sentence: and as soon as i put those first youtube videos up
Actual hindi translation: और जैसे ही मैने वो पहले यूट्यूब विडियो लगाये
Predicted hindi translation:  मुझे लगता है कि मैं अपने पास एक छोटा सा वीडियो
```

In [82]:
```
from nltk.translate.bleu_score import sentence_bleu
```

```
score = sentence_bleu(X_train[k:k+1].values[0], decoded_sentence[:-5])
print("Bleu score is", score)
```

```
Bleu score is 0.37991784282579627
/usr/local/lib/python3.7/dist-packages/nltk/translate/bleu_score.py:490: UserWarning:
Corpus/Sentence contains 0 counts of 2-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)
```

In [93]:
```
nactual = []
npredicted = []
k = 0
while k != 1000:
  (input_seq, actual_output), _ = next(train_gen)
  decoded_sentence = decode_sequence(input_seq)
  nactual.append(X_train[k:k+1].values[0])
  npredicted.append(decoded_sentence[:-5])
  k = k+1
```

In [95]:
```
from nltk.translate.bleu_score import corpus_bleu

score = corpus_bleu(nactual, npredicted)
print(score)
```

```
0.41657275568301416
/usr/local/lib/python3.7/dist-packages/nltk/translate/bleu_score.py:490: UserWarning:
Corpus/Sentence contains 0 counts of 2-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)
```

This model requires atleast 80-100 epochs, but due to lack of computational power, this model ran only 25 epoch, that too took around 8 hours of training time. Therefore the accuracy is not up to the mark. But, if trained for 100 epochs, it would be a lot better when trained for 100 epochs. This bleu score is for 1000 training examples.

https://colab.research.google.com/drive/16U6wBfn7145gSvIO0XAatIN76nBazymp?usp=sharing

Refer the above link for the original code