Sequence to sequence implementation

There will be some functions that start with the word "grader" ex: grader_check_encoder(), grader_check_attention(), grader_onestepdecoder() etc, you should not change those function definition.Every Grader function has to return True.

Note 1: There are many blogs on the attention mechanisum which might be misleading you, so do read the references completly and after that only please check the internet. The best things is to read the research papers and try to implement it on your own.

Note 2: To complete this assignment, the reference that are mentioned will be enough.

Note 3: If you are starting this assignment, you might have completed minimum of 20 assignment. If you are still not able to implement this algorithm you might have rushed in the previous assignments with out learning much and didn't spend your time productively.

Task -1: Simple Encoder and Decoder

Implement simple Encoder-Decoder model

1. Download the **Italian** to **English** translation dataset from here
2. You will find **ita.txt** file in that ZIP, you can read that data using python and preprocess that data this way only:
3. You have to implement a simple Encoder and Decoder architecture
4. Use BLEU score as metric to evaluate your model. You can use any loss function you need.
5. You have to use Tensorboard to plot the Graph, Scores and histograms of gradients.
6. a. Check the reference notebook

   b. Resource 2

Load the data

```
!wget http://www.manythings.org/anki/ita-eng.zip
!unzip ita-eng.zip
```

```
--2021-10-15 06:51:43--  http://www.manythings.org/anki/ita-eng.zip
Resolving www.manythings.org (www.manythings.org)... 104.21.92.44,
172.67.186.54, 2606:4700:3030::6815:5c2c, ...
Connecting to www.manythings.org (www.manythings.org)|
104.21.92.44|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7730753 (7.4M) [application/zip]
```

```
Saving to: ‘ita-eng.zip’

 ita-eng.zip           0%[                    ]       0  --.-KB/s               
ita-eng.zip         100%[===================>]   7.37M  --.-KB/s    in 0.06s

2021-10-15 06:51:43 (116 MB/s) - ‘ita-eng.zip’ saved [7730753/7730753]

Archive:  ita-eng.zip
  inflating: ita.txt                 
  inflating: _about.txt              
```

lets download the glove vectors ("vectors for english words"), note that this file will have vectors with 50d, 100d and 300d, you can choose any one of them based on your computing power

__ In our assignment we will be passing english text to the decoder, so we will be using these vectors in decoder embedding layer __

```
!wget https://www.dropbox.com/s/ddkmtqz01jc024u/glove.6B.100d.txt

--2021-10-15 06:51:45--  https://www.dropbox.com/s/ddkmtqz01jc024u/glove.6B.100d.txt
Resolving www.dropbox.com (www.dropbox.com)... 162.125.67.18, 2620:100:6020:18::a27d:4012
Connecting to www.dropbox.com (www.dropbox.com)|162.125.67.18|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/ddkmtqz01jc024u/glove.6B.100d.txt [following]
--2021-10-15 06:51:46--  https://www.dropbox.com/s/raw/ddkmtqz01jc024u/glove.6B.100d.txt
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc8c7c5f96ee08ab2fc0639f80d9.dl.dropboxusercontent.com/cd/0/inline/BYHLQ7FZtI7r7NAQsoXdxOsXY36JpHcuX-E61UA2MSfHwNEedSjBNcTY20fDh8WHvd6XqNi1n5dXMocjLdk5C_nhsV-8b619u6GYdJKAvvLNpKLf_y_mImBGPv8c4niqMAkuWgrK6js0t5758MjIA5gP/file# [following]
--2021-10-15 06:51:46--  https://uc8c7c5f96ee08ab2fc0639f80d9.dl.dropboxusercontent.com/cd/0/inline/BYHLQ7FZtI7r7NAQsoXdxOsXY36JpHcuX-E61UA2MSfHwNEedSjBNcTY20fDh8WHvd6XqNi1n5dXMocjLdk5C_nhsV-8b619u6GYdJKAvvLNpKLf_y_mImBGPv8c4niqMAkuWgrK6js0t5758MjIA5gP/file
Resolving uc8c7c5f96ee08ab2fc0639f80d9.dl.dropboxusercontent.com (uc8c7c5f96ee08ab2fc0639f80d9.dl.dropboxusercontent.com)... 162.125.67.15, 2620:100:6020:15::a27d:400f
Connecting to uc8c7c5f96ee08ab2fc0639f80d9.dl.dropboxusercontent.com (uc8c7c5f96ee08ab2fc0639f80d9.dl.dropboxusercontent.com)|162.125.67.15|:443... connected.
```

```
HTTP request sent, awaiting response... 200 OK
Length: 347116733 (331M) [text/plain]
Saving to: ‘glove.6B.100d.txt’

glove.6B.100d.txt   100%[===================>] 331.04M  23.9MB/s    in 
15s

2021-10-15 06:52:01 (22.4 MB/s) - ‘glove.6B.100d.txt’ saved 
[347116733/347116733]
```

Loading data

if you observe the data file, each feild was seprated by a tab '\t'

```
import matplotlib.pyplot as plt
%matplotlib inline
# import seaborn as sns
import pandas as pd
import re
import tensorflow as tf
from tensorflow.keras.layers import Embedding, LSTM, Dense,Dot
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np

with open('ita.txt', 'r', encoding="utf8") as f:
    eng=[]
    ita=[]
    for i in f.readlines():
        eng.append(i.split("\t")[0])
        ita.append(i.split("\t")[1])
data = pd.DataFrame(data=list(zip(eng, ita)), 
columns=['english','italian'])
print(data.shape)
data.head()

(352040, 2)

  english   italian
0     Hi.      Ciao!
1     Hi.      Ciao.
2    Run!    Corri!
3    Run!    Corra!
4    Run!  Correte!

def decontractions(phrase):
    """decontracted takes text and convert contractions into natural 
form.
     ref: https://stackoverflow.com/questions/19790188/expanding-
```

```
english-language-contractions-in-python/47091490#47091490"""
    # specific
    phrase = re.sub(r"won\'t", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    phrase = re.sub(r"won\’t", "will not", phrase)
    phrase = re.sub(r"can\’t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)

    phrase = re.sub(r"n\’t", " not", phrase)
    phrase = re.sub(r"\’re", " are", phrase)
    phrase = re.sub(r"\’s", " is", phrase)
    phrase = re.sub(r"\’d", " would", phrase)
    phrase = re.sub(r"\’ll", " will", phrase)
    phrase = re.sub(r"\’t", " not", phrase)
    phrase = re.sub(r"\’ve", " have", phrase)
    phrase = re.sub(r"\’m", " am", phrase)

    return phrase

def preprocess(text):
    # convert all the text into lower letters
    # use this function to remove the contractions:
https://gist.github.com/anandborad/d410a49a493b56dace4f814ab5325bbd
    # remove all the spacial characters: except space ' '
    text = text.lower()
    text = decontractions(text)
    text = re.sub('[^A-Za-z0-9 ]+', '', text)
    return text

def preprocess_ita(text):
    # convert all the text into lower letters
    # remove the words betweent brakets ()
    # remove these characters: {'$', ')', '?', '"', '’', '.',  '°',
'!', ';', '/', "'", '€', '%', ':', ',', '('}
    # replace these spl characters with space: '\u200b', '\xa0', '-',
'/'
    # we have found these characters after observing the data points,
feel free to explore more and see if you can do find more
    # you are free to do more proprocessing
    # note that the model will learn better with better preprocessed
data
```

```
    text = text.lower()
    text = decontractions(text)
    text = re.sub('[$)\?"’.°!;\'€%:,(/]', '', text)
    text = re.sub('\u200b', ' ', text)
    text = re.sub('\xa0', ' ', text)
    text = re.sub('-', ' ', text)
    return text


data['english'] = data['english'].apply(preprocess)
data['italian'] = data['italian'].apply(preprocess_ita)
data.head()
```

	english	italian
0	hi	ciao
1	hi	ciao
2	run	corri
3	run	corra
4	run	correte

```
ita_lengths = data['italian'].str.split().apply(len)
eng_lengths = data['english'].str.split().apply(len)
```

```
for i in range(0,101,10):
    print(i,np.percentile(ita_lengths, i))
for i in range(90,101):
    print(i,np.percentile(ita_lengths, i))
for i in [99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100]:
    print(i,np.percentile(ita_lengths, i))
```

```
0 1.0
10 3.0
20 4.0
30 4.0
40 5.0
50 5.0
60 6.0
70 6.0
80 7.0
90 8.0
100 92.0
90 8.0
91 8.0
92 8.0
93 9.0
94 9.0
95 9.0
96 10.0
97 10.0
```

```
98 11.0
99 12.0
100 92.0
99.1 12.0
99.2 12.0
99.3 13.0
99.4 13.0
99.5 13.0
99.6 14.0
99.7 15.0
99.8 16.0
99.9 22.0
100 92.0
```

```python
for i in range(0,101,10):
    print(i,np.percentile(eng_lengths, i))
for i in range(90,101):
    print(i,np.percentile(eng_lengths, i))
for i in [99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100]:
    print(i,np.percentile(eng_lengths, i))
```

```
0 1.0
10 4.0
20 4.0
30 5.0
40 5.0
50 6.0
60 6.0
70 7.0
80 7.0
90 8.0
100 101.0
90 8.0
91 9.0
92 9.0
93 9.0
94 9.0
95 9.0
96 10.0
97 10.0
98 11.0
99 12.0
100 101.0
99.1 12.0
99.2 13.0
99.3 13.0
99.4 13.0
99.5 14.0
99.6 14.0
99.7 15.0
```

```
99.8 16.0
99.9 25.0
100 101.0
```

If you observe the values, 99.9% of the data points are having length < 20, so select the sentences that have words < 20 Inorder to do the teacher forcing while training of seq-seq models, lets create two new columns, one with <start> token at begining of the sentence and other column with <end> token at the end of the sequence

```
data['italian_len'] = data['italian'].str.split().apply(len)
data = data[data['italian_len'] < 20]

data['english_len'] = data['english'].str.split().apply(len)
data = data[data['english_len'] < 20]

data['english_inp'] = '<start> ' + data['english'].astype(str)
data['english_out'] = data['english'].astype(str) + ' <end>'

data = data.drop(['english','italian_len','english_len'], axis=1)
# only for the first sentance add a toke <end> so that we will have
<end> in tokenizer
data.head()

   italian  english_inp english_out
0     ciao   <start> hi    hi <end>
1     ciao   <start> hi    hi <end>
2    corri  <start> run   run <end>
3    corra  <start> run   run <end>
4  correte  <start> run   run <end>

data.sample(10)

                                                  italian  ...
english_out
196603                        voi avete fatto i vostri compiti  ...
have you done your homework <end>
186528                             lei chiamò mentre ero fuori  ...
she called while i was out <end>
271391                  tom non ha ancora firmato un contratto  ...
tom has not signed a contract yet <end>
29380                                               mostratevi  ...
show yourselves <end>
345238  tom sta facendo una lista di cose che devono e...  ...  tom is
making a list of things that need to be...
88202                                   io posso capire tom  ...
i can understand tom <end>
235961                        tom è entrato nellappartamento  ...
tom went inside the apartment <end>
255313                            lo devo fare prima di lunedì  ...
i have to do that before monday <end>
```

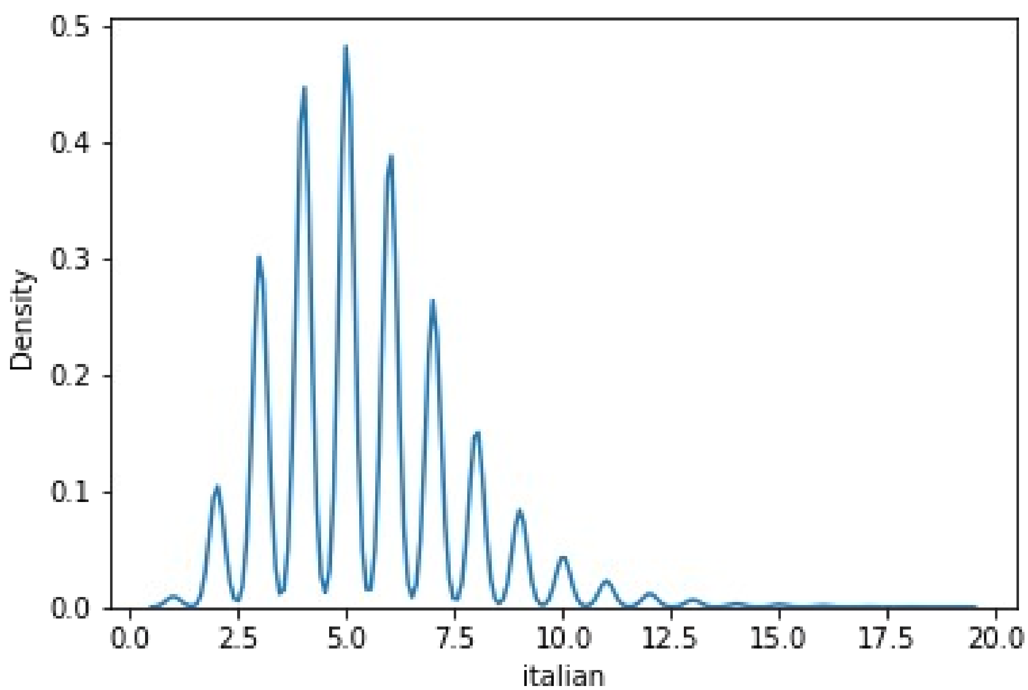
```
86809                            si diedero tutti le mani  ...
everyone shook hands <end>
80370                                 tom non ha fratelli  ...
tom has no brothers <end>

[10 rows x 3 columns]
```

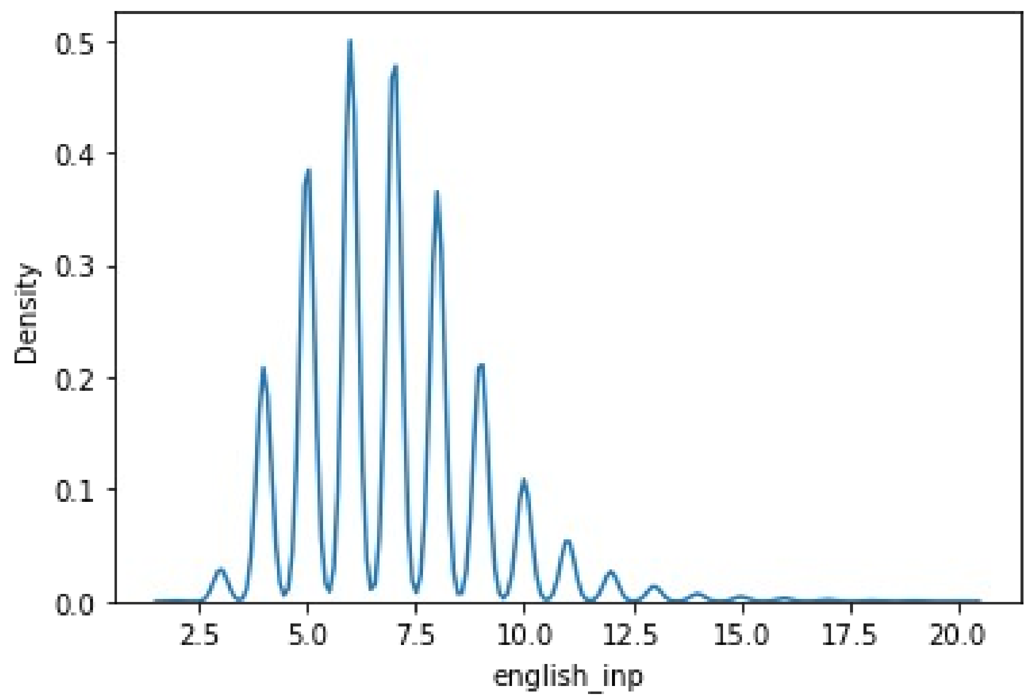
Teacher Forcing

Getting train and test

```
from sklearn.model_selection import train_test_split
train, validation = train_test_split(data, test_size=0.2)

print(train.shape, validation.shape)
# for one sentence we will be adding <end> token so that the tokanizer 
learns the word <end>
# with this we can use only one tokenizer for both encoder output and 
decoder output
train.iloc[0]['english_inp']= str(train.iloc[0]['english_inp'])+' 
<end>'
train.iloc[0]['english_out']= str(train.iloc[0]['english_out'])+' 
<end>'

(281244, 3) (70311, 3)

train.head()

                                         italian  ...
english_out
137299                sono un cittadino americano  ...      i am an 
american citizen <end> <end>
35656                         sono ancora da sola  ...
i am still alone <end>
152319            ho cercato di essere aggressivo  ...            i 
tried to be aggressive <end>
285291  il suo figlio più giovane ha cinque anni  ...  his youngest 
son is five years old <end>
166795          mi sono sbronzato la scorsa notte  ...           i got 
hammered last night <end>

[5 rows x 3 columns]

validation.head()

                                                  italian  ...
english_out
333047  la partita è stata annullata a causa della pes...  ...  the 
game was canceled because of heavy rain <end>
210120             siete passati dalla padella alla brace  ...
you have traded bad for worse <end>
```

```
328226  dobbiamo indossare delle uniformi scolastiche ...  ...    we
have to wear school uniforms at school <end>
20783                                    lo dimentichi e basta  ...
just forget it <end>
309449                  è sempre stata unattrice popolare  ...
she has always been a popular actress <end>

[5 rows x 3 columns]

ita_lengths = train['italian'].str.split().apply(len)
eng_lengths = train['english_inp'].str.split().apply(len)
import seaborn as sns
sns.kdeplot(ita_lengths)
plt.show()
sns.kdeplot(eng_lengths)
plt.show()
```





Creating Tokenizer on the train data and learning vocabulary

Note that we are fitting the tokenizer only on train data and check the filters for english, we need to remove symbols < and >

```
tknizer_ita = Tokenizer()
tknizer_ita.fit_on_texts(train['italian'].values)
tknizer_eng = Tokenizer(filters='!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n')
tknizer_eng.fit_on_texts(train['english_inp'].values)

vocab_size_eng=len(tknizer_eng.word_index.keys())
print(vocab_size_eng)
vocab_size_ita=len(tknizer_ita.word_index.keys())
print(vocab_size_ita)

13060
26563

#tknizer_eng.word_index['<start>'], tknizer_eng.word_index['<end>']

# def grader_1(data):
#     shape_value = data.shape ==(340044, 3)
#     tknizer = Tokenizer(char_level=True)
#     tknizer.fit_on_texts(data['italian'].values)
#     ita_chars = tknizer.word_index.keys()
#     diff_chars_ita = set(ita_chars)-set([' ', 't', 'a', 'o', 'r',
'e', 's', 'i', 'n', 'l', 'c', 'm', 'u', 'd', 'p', 'v', 'h', 'g', 'b',
'f', 'è', 'q', 'z', 'ò', 'à', 'y', 'é', 'ì', 'ù', 'k', 'w', '0', 'j',
'1', '3', '2', 'x', '9', '5', '8', '4', '6', '7', 'á', 'ñ', 'ê', 'ü',
```

```
'ō', 'î', 'ö', 'ú', 'º'])
#     tknizer = Tokenizer(char_level=True)
#     tknizer.fit_on_texts(data['english_inp'].values)
#     eng_chars = tknizer.word_index.keys()
#     diff_chars_eng = set(eng_chars)-set(['<','>',' ', 'e', 'o', 't',
'i', 'a', 'n', 's', 'h', 'r', 'l', 'd', 'm', 'y', 'u', 'w', 'g', 'c',
'p', 'f', 'b', 'k', 'v', 'j', 'x', 'z', 'q', '0', '1', '3', '2', '9',
'5', '8', '6', '4', '7'])
#     unique_char_value = (len(diff_chars_eng)==0) and
(len(diff_chars_ita)==0)
#     return unique_char_value and shape_value

# grader_1(data)
```

Creating embeddings for english sentences

```
embeddings_index = dict()
f = open('glove.6B.100d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

embedding_matrix = np.zeros((vocab_size_eng+1, 100))
for word, i in tknizer_eng.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

embedding_matrix.shape
```

```
(13061, 100)
```

Implement custom encoder decoder

Encoder

```
class Encoder(tf.keras.Model):
    '''
    Encoder model -- That takes a input sequence and returns encoder-
outputs,encoder_final_state_h,encoder_final_state_c
    '''

    def
__init__(self,inp_vocab_size,embedding_size,lstm_size,input_length):

        #Initialize Embedding layer
```

```
        #Intialize Encoder LSTM layer
        super().__init__()

        self.inp_vocab_size = inp_vocab_size
        self.embedding_size = embedding_size
        self.lstm_size = lstm_size
        self.input_length = input_length
        self.lstm_out = 0
        self.lstm_encoder_final_state_h = 0
        self.lstm_encoder_final_state_c = 0
        self.embedding =
Embedding(input_dim=self.inp_vocab_size,output_dim=self.embedding_size
,input_length=self.input_length,mask_zero=True,
name="embedding_layer_encoder")

        self.lstm =
LSTM(self.lstm_size,return_state=True,return_sequences=True,name="Enco
der_LSTM")

        #self.num_outputs = num_outputs


    def call(self,input_sequence,states):

        '''
          This function takes a sequence input and the initial states
of the encoder.
          Pass the input_sequence input to the Embedding layer, Pass
the embedding layer ouput to encoder_lstm
          returns -- encoder_output, last time step's hidden and cell
state
        '''
        input_embedd=self.embedding(input_sequence)

self.lstm_out,self.lstm_encoder_final_state_h,self.lstm_encoder_final_
state_c=self.lstm(input_embedd)
        return
self.lstm_out,self.lstm_encoder_final_state_h,self.lstm_encoder_final_
state_c




    def initialize_states(self,batch_size):
      '''
      Given a batch size it will return intial hidden state and intial
cell state.
      If batch size is 32- Hidden state is zeros of size
[32,lstm_units], cell state zeros is of size [32,lstm_units]
      '''
```

```
      return
tf.zeros((batch_size,self.lstm_size)),tf.zeros((batch_size,self.lstm_s
ize))
```

Grader function - 1

```
def grader_check_encoder():
    '''
        vocab-size: Unique words of the input language,
        embedding_size: output embedding dimension for each word after
embedding layer,
        lstm_size: Number of lstm units,
        input_length: Length of the input sentence,
        batch_size
    '''
    vocab_size=10
    embedding_size=20
    lstm_size=32
    input_length=10
    batch_size=16
    #Intialzing encoder
    encoder=Encoder(vocab_size,embedding_size,lstm_size,input_length)

input_sequence=tf.random.uniform(shape=[batch_size,input_length],maxva
l=vocab_size,minval=0,dtype=tf.int32)
    #Intializing encoder initial states
    initial_state=encoder.initialize_states(batch_size)


encoder_output,state_h,state_c=encoder(input_sequence,initial_state)

    assert(encoder_output.shape==(batch_size,input_length,lstm_size)
and state_h.shape==(batch_size,lstm_size) and
state_c.shape==(batch_size,lstm_size))
    return True
print(grader_check_encoder())
```

```
True
```

```
class Decoder(tf.keras.Model):
    '''
    Encoder model -- That takes a input sequence and returns output
sequence
    '''

    def
__init__(self,out_vocab_size,embedding_size,lstm_size,input_length):

        #Initialize Embedding layer
```

```
        #Intialize Decoder LSTM layer
        super().__init__()
        self.out_vocab_size=out_vocab_size
        self.embedding_size=embedding_size
        self.lstm_size=lstm_size
        self.input_length=input_length


self.embedding=Embedding(input_dim=self.out_vocab_size,output_dim=self
.embedding_size,input_length=self.input_length,mask_zero=True,name="em
bedding_layer_encoder")

self.lstm=LSTM(self.lstm_size,return_sequences=True,return_state=True,
name="Encoder_LSTM")



    def call(self,input_sequence,initial_states):
        '''
          This function takes a sequence input and the initial states
of the encoder.
          Pass the input_sequence input to the Embedding layer, Pass
the embedding layer ouput to decoder_lstm

          returns --
decoder_output,decoder_final_state_h,decoder_final_state_c
        '''
        input_embedd=self.embedding(input_sequence)
        decoder_out,decoder_final_state_c,decoder_final_state_h
=self.lstm(input_embedd)
        return decoder_out,decoder_final_state_c,decoder_final_state_h

#self.lstm_out,self.lstm_encoder_final_state_h,self.lstm_encoder_final
_state_c=self.lstm(input_embedd)
        #return
self.lstm_out,self.lstm_encoder_final_state_h,self.lstm_encoder_final_
state_c
```

italicized text**Grader function - 2**

```
def grader_decoder():
    '''
        out_vocab_size: Unique words of the target language,
        embedding_size: output embedding dimension for each word after
embedding layer,
        dec_units: Number of lstm units in decoder,
        input_length: Length of the input sentence,
        batch_size
```

```
    '''
    out_vocab_size=13
    embedding_dim=12
    input_length=10
    dec_units=16
    batch_size=32


target_sentences=tf.random.uniform(shape=(batch_size,input_length),max
val=10,minval=0,dtype=tf.int32)

encoder_output=tf.random.uniform(shape=[batch_size,input_length,dec_un
its])
    state_h=tf.random.uniform(shape=[batch_size,dec_units])
    state_c=tf.random.uniform(shape=[batch_size,dec_units])
    states=[state_h,state_c]
    decoder=Decoder(out_vocab_size, embedding_dim,
dec_units,input_length )
    output,_,_=decoder(target_sentences, states)
    assert(output.shape==(batch_size,input_length,dec_units))
    return True
print(grader_decoder())
```

```
True
```

```
class Dataset:
    def __init__(self, data, tknizer_ita, tknizer_eng, max_len):
        self.encoder_inps = data['italian'].values
        self.decoder_inps = data['english_inp'].values
        self.decoder_outs = data['english_out'].values
        self.tknizer_eng = tknizer_eng
        self.tknizer_ita = tknizer_ita
        self.max_len = max_len

    def __getitem__(self, i):
        self.encoder_seq =
self.tknizer_ita.texts_to_sequences([self.encoder_inps[i]]) # need to
pass list of values
        self.decoder_inp_seq =
self.tknizer_eng.texts_to_sequences([self.decoder_inps[i]])
        self.decoder_out_seq =
self.tknizer_eng.texts_to_sequences([self.decoder_outs[i]])

        self.encoder_seq = pad_sequences(self.encoder_seq,
maxlen=self.max_len, dtype='int32', padding='post')
        self.decoder_inp_seq = pad_sequences(self.decoder_inp_seq,
maxlen=self.max_len, dtype='int32', padding='post')
        self.decoder_out_seq = pad_sequences(self.decoder_out_seq,
```

```
maxlen=self.max_len, dtype='int32', padding='post')
        return self.encoder_seq, self.decoder_inp_seq,
self.decoder_out_seq

    def __len__(self): # your model.fit_gen requires this function
        return len(self.encoder_inps)


class Dataloder(tf.keras.utils.Sequence):
    def __init__(self, dataset, batch_size=1):
        self.dataset = dataset
        self.batch_size = batch_size
        self.indexes = np.arange(len(self.dataset.encoder_inps))


    def __getitem__(self, i):
        start = i * self.batch_size
        stop = (i + 1) * self.batch_size
        data = []
        for j in range(start, stop):
            data.append(self.dataset[j])

        batch = [np.squeeze(np.stack(samples, axis=1), axis=0) for
samples in zip(*data)]
        # we are creating data like ([italian, english_inp],
english_out) these are already converted into seq
        return tuple([[batch[0],batch[1]],batch[2]])

    def __len__(self):  # your model.fit_gen requires this function
        return len(self.indexes) // self.batch_size

    def on_epoch_end(self):
        self.indexes = np.random.permutation(self.indexes)


train_dataset = Dataset(train, tknizer_ita, tknizer_eng, 20)
test_dataset  = Dataset(validation, tknizer_ita, tknizer_eng, 20)

train_dataloader = Dataloder(train_dataset, batch_size=1024)
test_dataloader = Dataloder(test_dataset, batch_size=1024)


print(train_dataloader[0][0][0].shape, train_dataloader[0][0]
[1].shape, train_dataloader[0][1].shape)
```

```
(1024, 20) (1024, 20) (1024, 20)
```

```
class Encoder_decoder(tf.keras.Model):
```

```
    def
__init__(self,encoder_inputs_length=20,decoder_inputs_length=20,inp_vo
cab_size=vocab_size_ita, out_vocab_size=vocab_size_eng,lstm_size=128):

        #Create encoder object
        #Create decoder object
        #Intialize Dense layer(out_vocab_size) with
activation='softmax'
        super().__init__() #
https://stackoverflow.com/a/27134600/4084039
        self.encoder = Encoder(inp_vocab_size=vocab_size_ita + 1,
embedding_size=50, input_length=encoder_inputs_length,lstm_size=256)
        self.decoder = Decoder(out_vocab_size=vocab_size_eng + 1,
embedding_size=100, input_length=decoder_inputs_length,lstm_size=256)
        self.dense   = Dense(out_vocab_size, activation='softmax')



    def call(self,data):
        '''
        A. Pass the input sequence to Encoder layer -- Return
encoder_output,encoder_final_state_h,encoder_final_state_c
        B. Pass the target sequence to Decoder layer with intial
states as encoder_final_state_h,encoder_final_state_C
        C. Pass the decoder_outputs into Dense layer

        Return decoder_outputs
        '''
        input,output = data[0], data[1]
        initial_state = self.encoder.initialize_states(len(input))
        encoder_output, encoder_h, encoder_c =
self.encoder(input,initial_state)
        decoder_output ,__,__                      =
self.decoder(output,initial_states=[ encoder_h, encoder_c])
        output                                 =
self.dense(decoder_output)
        return output


#Create an object of encoder_decoder Model class,
# Compile the model and fit the model

import datetime
```

Model training 1

Simple encoder decoder model

Without Attention model

```
import datetime
%load_ext tensorboard
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M
%S")
tensorboard = tf.keras.callbacks.TensorBoard(log_dir=log_dir,

histogram_freq=1)
model  =
Encoder_decoder(encoder_inputs_length=20,decoder_inputs_length=20,inp_
vocab_size=vocab_size_ita,
                         out_vocab_size=vocab_size_eng,lstm_size=128)
train_steps=train.shape[0]//1024
valid_steps=validation.shape[0]//1024

optimizer = tf.keras.optimizers.Adam()
model.compile(optimizer=optimizer,loss='sparse_categorical_crossentrop
y',metrics=['accuracy'])

model.fit_generator(train_dataloader,
                    steps_per_epoch=train_steps, epochs=20,
                    validation_data=test_dataloader,

validation_steps=valid_steps,callbacks=[tensorboard])
model.summary()
```

```
/usr/local/lib/python3.7/dist-packages/keras/engine/training.py:1972:
UserWarning: `Model.fit_generator` is deprecated and will be removed
in a future version. Please use `Model.fit`, which supports
generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/20
WARNING:tensorflow:Gradients do not exist for variables
['encoder_decoder/encoder_1/embedding_layer_encoder/embeddings:0',
'encoder_decoder/encoder_1/Encoder_LSTM/lstm_cell_2/kernel:0',
'encoder_decoder/encoder_1/Encoder_LSTM/lstm_cell_2/recurrent_kernel:0
', 'encoder_decoder/encoder_1/Encoder_LSTM/lstm_cell_2/bias:0'] when
minimizing the loss.
WARNING:tensorflow:Gradients do not exist for variables
['encoder_decoder/encoder_1/embedding_layer_encoder/embeddings:0',
'encoder_decoder/encoder_1/Encoder_LSTM/lstm_cell_2/kernel:0',
'encoder_decoder/encoder_1/Encoder_LSTM/lstm_cell_2/recurrent_kernel:0
', 'encoder_decoder/encoder_1/Encoder_LSTM/lstm_cell_2/bias:0'] when
minimizing the loss.
274/274 [==============================] - 131s 450ms/step - loss:
1.9540 - accuracy: 0.1849 - val_loss: 1.6277 - val_accuracy: 0.2208
Epoch 2/20
274/274 [==============================] - 123s 447ms/step - loss:
1.5579 - accuracy: 0.2385 - val_loss: 1.4779 - val_accuracy: 0.2636
Epoch 3/20
```

```
274/274 [==============================] - 124s 451ms/step - loss: 
1.4186 - accuracy: 0.2865 - val_loss: 1.3613 - val_accuracy: 0.3018
Epoch 4/20
274/274 [==============================] - 123s 449ms/step - loss: 
1.3313 - accuracy: 0.3090 - val_loss: 1.3004 - val_accuracy: 0.3152
Epoch 5/20
274/274 [==============================] - 123s 450ms/step - loss: 
1.2787 - accuracy: 0.3205 - val_loss: 1.2599 - val_accuracy: 0.3238
Epoch 6/20
274/274 [==============================] - 123s 449ms/step - loss: 
1.2404 - accuracy: 0.3286 - val_loss: 1.2286 - val_accuracy: 0.3311
Epoch 7/20
274/274 [==============================] - 123s 450ms/step - loss: 
1.2097 - accuracy: 0.3351 - val_loss: 1.2039 - val_accuracy: 0.3354
Epoch 8/20
274/274 [==============================] - 123s 449ms/step - loss: 
1.1842 - accuracy: 0.3403 - val_loss: 1.1829 - val_accuracy: 0.3404
Epoch 9/20
274/274 [==============================] - 124s 452ms/step - loss: 
1.1620 - accuracy: 0.3451 - val_loss: 1.1656 - val_accuracy: 0.3441
Epoch 10/20
274/274 [==============================] - 123s 450ms/step - loss: 
1.1428 - accuracy: 0.3497 - val_loss: 1.1497 - val_accuracy: 0.3486
Epoch 11/20
274/274 [==============================] - 124s 451ms/step - loss: 
1.1254 - accuracy: 0.3538 - val_loss: 1.1361 - val_accuracy: 0.3517
Epoch 12/20
274/274 [==============================] - 123s 451ms/step - loss: 
1.1096 - accuracy: 0.3577 - val_loss: 1.1241 - val_accuracy: 0.3552
Epoch 13/20
274/274 [==============================] - 124s 451ms/step - loss: 
1.0951 - accuracy: 0.3614 - val_loss: 1.1122 - val_accuracy: 0.3581
Epoch 14/20
274/274 [==============================] - 124s 451ms/step - loss: 
1.0818 - accuracy: 0.3648 - val_loss: 1.1022 - val_accuracy: 0.3605
Epoch 15/20
274/274 [==============================] - 124s 450ms/step - loss: 
1.0692 - accuracy: 0.3681 - val_loss: 1.0929 - val_accuracy: 0.3638
Epoch 16/20
274/274 [==============================] - 123s 450ms/step - loss: 
1.0576 - accuracy: 0.3716 - val_loss: 1.0840 - val_accuracy: 0.3664
Epoch 17/20
274/274 [==============================] - 124s 451ms/step - loss: 
1.0466 - accuracy: 0.3747 - val_loss: 1.0756 - val_accuracy: 0.3691
Epoch 18/20
274/274 [==============================] - 123s 450ms/step - loss: 
1.0362 - accuracy: 0.3781 - val_loss: 1.0679 - val_accuracy: 0.3712
Epoch 19/20
274/274 [==============================] - 124s 451ms/step - loss: 
```

```
1.0264 - accuracy: 0.3811 - val_loss: 1.0609 - val_accuracy: 0.3738
Epoch 20/20
274/274 [==============================] - 124s 451ms/step - loss: 
1.0171 - accuracy: 0.3841 - val_loss: 1.0543 - val_accuracy: 0.3766
Model: "encoder_decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
encoder_1 (Encoder)          multiple                  1642568   
_________________________________________________________________
decoder_1 (Decoder)          multiple                  1671668   
_________________________________________________________________
dense (Dense)                multiple                  3356420   
=================================================================
Total params: 6,670,656
Trainable params: 6,670,656
Non-trainable params: 0
_________________________________________________________________
```

```
!kill 426
```

```
/bin/bash: line 0: kill: (426) - No such process
```

```
from IPython.display import Image
%load_ext tensorboard
%tensorboard --logdir logs/fit
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
<IPython.core.display.Javascript object>
```

```
import tensorflow as tf
tf.compat.v1.enable_eager_execution()
from tensorflow.keras.layers import TimeDistributed
tf.keras.backend.clear_session()
from tensorflow.keras.layers import Input, Softmax, RNN, Dense, 
Embedding, LSTM
from tensorflow.keras.models import Model
import numpy as np
```

```
# This function predict Translated sentence and return Translated 
sentence and attention weights
def predict(input_sentence):

  '''
  A. Given input sentence, convert the sentence into integers using 
tokenizer used earlier
  B. Pass the input_sequence to encoder. we get encoder_outputs, last 
time step hidden and cell state
  C. Initialize index of <start> as input to decoder. and encoder
```

```
final states as input_states to decoder
  D. till we reach max_length of decoder or till the model predicted
word <end>:
         predicted_out,state_h,state_c=model.layers[1]
(dec_input,states)
         pass the predicted_out to the dense layer
         update the states=[state_h,state_c]
         And get the index of the word with maximum probability of the
dense layer output, using the tokenizer(word index) get the word and
then store it in a string.
         Update the input_to_decoder with current predictions
  F. Return the predicted sentence
  '''

  encoder_seq = tknizer_ita.texts_to_sequences([input_sentence])
  encoder_seq =
pad_sequences(encoder_seq,maxlen=20,dtype='int32',padding='post')
  initial_state=model.layers[0].initialize_states(1)
  encoder_output, encoder_state_h, encoder_state_c = model.layers[0]
(encoder_seq,initial_state)
  states_values = [encoder_state_h,encoder_state_c]
  pred = []
  cur_vec = tf.expand_dims([tknizer_eng.word_index['<start>']], 0)


  for i in range(DECODER_SEQ_LEN):
    cur_emb = model.layers[1].embedding(cur_vec)
    infe_output, state_h, state_c =
model.layers[1].lstm(cur_emb,initial_state=states_values)
    infe_output=model.layers[2](infe_output)
    states_values = [state_h, state_c]
    if cur_vec == end_index:
      return pred
    cur_vec = np.reshape(np.argmax(infe_output), (1, 1))
    pred.append(cur_vec)
  return pred

import nltk.translate.bleu_score as bleu
DECODER_SEQ_LEN = 20
end_index = tknizer_eng.word_index['<end>']
blue_scores=[]
for i in range(1000):
  acutal_sentence = validation['italian'].sample().item()
  #predicted_sentence=predict(acutal_sentence)
  pred = predict(acutal_sentence)
  sent_predicted = []

  for j in pred:
    sent_predicted.append(tknizer_eng.sequences_to_texts(j))
```

```
  sent_predicted = list(map(''.join,sent_predicted))
  blue_scores.append(bleu.sentence_bleu(acutal_sentence,
sent_predicted))

print("Average BLUE Score: ",np.average(np.array(blue_scores)))
```

```
/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)

Average BLUE Score:  0.5286025626187095
```

Task -2: Including Attention mechanisum

1. Use the preprocessed data from Task-1

2. You have to implement an Encoder and Decoder architecture with attention as discussed in the reference notebook.

    - Encoder - with 1 layer LSTM
    - Decoder - with 1 layer LSTM
    - attention - (Please refer the **reference notebook** to know more about the attention mechanism.)

3. In Global attention, we have 3 types of scoring functions(as discussed in the reference notebook). As a part of this assignment **you need to create 3 models for each scoring function**

    - In model 1 you need to implemnt "dot" score function
    - In model 2 you need to implemnt "general" score function
    - In model 3 you need to implemnt "concat" score function.

Please do add the markdown titles for each model so that we can have a better look at the code and verify.

1. It is mandatory to train the model with simple model.fit() only, Donot train the model with custom GradientTape()

2. Using attention weights, you can plot the attention plots, please plot those for 2-3 examples. You can check about those in this

3. The attention layer has to be written by yourself only. The main objective of this assignment is to read and implement a paper on yourself so please do it yourself.

4. Please implement the class **onestepdecoder** as mentioned in the assignment instructions.

5. You can use any tf.Keras highlevel API's to build and train the models. Check the reference notebook for better understanding.

6. Use BLEU score as metric to evaluate your model. You can use any loss function you need.

7. You have to use Tensorboard to plot the Graph, Scores and histograms of gradients.

8. Resources:

   a. Check the reference notebook

   b. Resource 1

   c. Resource 2

   d. Resource 3

Implement custom encoder decoder and attention layers

Encoder

Grader function - 1

```
def grader_check_encoder():
    '''
        vocab-size: Unique words of the input language,
        embedding_size: output embedding dimension for each word after
embedding layer,
        lstm_size: Number of lstm units in encoder,
        input_length: Length of the input sentence,
        batch_size
    '''

    vocab_size=10
    embedding_size=20
    lstm_size=32
    input_length=10
    batch_size=16
    encoder=Encoder(vocab_size,embedding_size,lstm_size,input_length)

input_sequence=tf.random.uniform(shape=[batch_size,input_length],maxva
l=vocab_size,minval=0,dtype=tf.int32)
    initial_state=encoder.initialize_states(batch_size)

encoder_output,state_h,state_c=encoder(input_sequence,initial_state)

    assert(encoder_output.shape==(batch_size,input_length,lstm_size)
and state_h.shape==(batch_size,lstm_size) and
state_c.shape==(batch_size,lstm_size))
    return True
print(grader_check_encoder())
```

```
True
```

Attention

```
class Attention(tf.keras.layers.Layer):
  '''
    Class the calculates score based on the scoring_function using
Bahdanu attention mechanism.
  '''
  def __init__(self,scoring_function, att_units):
    super().__init__()
    self.scoring_function=scoring_function
    self.att_units=att_units

    # Please go through the reference notebook and research paper to
complete the scoring functions

    if self.scoring_function=='dot':
      # Intialize variables needed for Dot score function here
      pass

    if scoring_function == 'general':
      # Intialize variables needed for General score function here
      self.dens_1=Dense(att_units,name='general_layer')

    elif scoring_function == 'concat':
      # Intialize variables needed for Concat score function here
      self.dens_1=Dense(att_units,name='concat_layer1')
      self.dens_2=Dense(att_units,name='concat_layer2')
      self.dens_3 = Dense(1,name='concat_layer3')


  def call(self,decoder_hidden_state,encoder_output):
    '''
      Attention mechanism takes two inputs current step --
decoder_hidden_state and all the encoder_outputs.
      * Based on the scoring function we will find the score or
similarity between decoder_hidden_state and encoder_output.
        Multiply the score function with your encoder_outputs to get
the context vector.
        Function returns context vector and attention weights(softmax
- scores)
    '''

    if self.scoring_function == 'dot':
        # Implement Dot score function here
        hidden_ = tf.expand_dims(decoder_hidden_state,1)
        out = Dot(axes=(2,2))([encoder_output,hidden_])

    elif self.scoring_function == 'general':
```

```
        # Implement General score function here
        out_hidd_ = tf.expand_dims(decoder_hidden_state,1)
        out = Dot((2,2))([encoder_output , self.dens_1(out_hidd_)])

    elif self.scoring_function == 'concat':
        # Implement General score function here
        out_hidd_1 = tf.expand_dims(decoder_hidden_state,1)
        out =
self.dens_3(tf.nn.tanh(self.dens_1(out_hidd_1)+self.dens_2(encoder_out
put)))

    wt_attentn  = tf.nn.softmax(out,axis=1)
    vec_con = wt_attentn  * encoder_output
    vec_con = tf.reduce_sum(vec_con,axis=1)
    return vec_con,wt_attentn
```

Grader function - 2

```
def grader_check_attention(scoring_fun):

    '''
        att_units: Used in matrix multiplications for scoring
functions,
        input_length: Length of the input sentence,
        batch_size
    '''

    input_length=10
    batch_size=16
    att_units=32

    state_h=tf.random.uniform(shape=[batch_size,att_units])

encoder_output=tf.random.uniform(shape=[batch_size,input_length,att_un
its])
    attention=Attention(scoring_fun,att_units)
    context_vector,attention_weights=attention(state_h,encoder_output)
    assert(context_vector.shape==(batch_size,att_units) and
attention_weights.shape==(batch_size,input_length,1))
    return True
print(grader_check_attention('dot'))
print(grader_check_attention('general'))
print(grader_check_attention('concat'))
```

```
True
True
True
```

OneStepDecoder

```
class One_Step_Decoder(tf.keras.Model):
  def __init__(self,tar_vocab_size, embedding_dim, input_length,
dec_units ,score_fun ,att_units):

      # Initialize decoder embedding layer, LSTM and any other objects
needed
      super().__init__()
      self.tar_vocab_size=tar_vocab_size
      self.embedding_dim=embedding_dim
      self.input_length=input_length
      self.dec_units=dec_units
      self.score_fun=score_fun
      self.att_units=att_units

      self.embedding =
Embedding(input_dim=self.tar_vocab_size,output_dim=self.embedding_dim,
input_length=self.input_length,

mask_zero=True,name='embedding_layers')
      self.lstm =
LSTM(self.dec_units,return_sequences=True,return_state=True,name='ones
tepdecoder_layers')
      self.dense =
Dense(self.tar_vocab_size,name='One_step_Decoder_Dense_layer')
      self.attention =
Attention(scoring_function=self.score_fun,att_units=self.att_units)


  def call(self,input_to_decoder, encoder_output, state_h,state_c):
    '''
        One step decoder mechanisim step by step:
      A. Pass the input_to_decoder to the embedding layer and then get
the output(batch_size,1,embedding_dim)
      B. Using the encoder_output and decoder hidden state, compute
the context vector.
      C. Concat the context vector with the step A output
      D. Pass the Step-C output to LSTM/GRU and get the decoder output
and states(hidden and cell state)
      E. Pass the decoder output to dense layer(vocab size) and store
the result into output.
      F. Return the states from step D, output from Step E, attention
weights from Step -B
    '''
    embb_ = self.embedding(input_to_decoder)
    vec_cont,wt_attentn = self.attention(state_h,encoder_output)
    conc =
tf.keras.layers.concatenate([tf.expand_dims(vec_cont,1),embb_],axis=-
1)
```

```
    out_deco,state_decoder_h,state_decoder_c = self.lstm(conc)
    out_deco_flatt = tf.keras.layers.Flatten()(out_deco)
    out = self.dense(out_deco_flatt)
    return out,state_decoder_h,state_decoder_c,wt_attentn,vec_cont
```

Grader function - 3

```
def grader_onestepdecoder(score_fun):

    '''
        tar_vocab_size: Unique words of the target language,
        embedding_dim: output embedding dimension for each word after
embedding layer,
        dec_units: Number of lstm units in decoder,
        att_units: Used in matrix multiplications for scoring
functions in attention class,
        input_length: Length of the target sentence,
        batch_size


    '''

    tar_vocab_size=13
    embedding_dim=12
    input_length=10
    dec_units=16
    att_units=16
    batch_size=32
    onestepdecoder=One_Step_Decoder(tar_vocab_size, embedding_dim,
input_length, dec_units ,score_fun ,att_units)

input_to_decoder=tf.random.uniform(shape=(batch_size,1),maxval=10,minv
al=0,dtype=tf.int32)

encoder_output=tf.random.uniform(shape=[batch_size,input_length,dec_un
its])
    state_h=tf.random.uniform(shape=[batch_size,dec_units])
    state_c=tf.random.uniform(shape=[batch_size,dec_units])

output,state_h,state_c,attention_weights,context_vector=onestepdecoder
(input_to_decoder,encoder_output,state_h,state_c)
    assert(output.shape==(batch_size,tar_vocab_size))
    assert(state_h.shape==(batch_size,dec_units))
    assert(state_c.shape==(batch_size,dec_units))
    assert(attention_weights.shape==(batch_size,input_length,1))
    assert(context_vector.shape==(batch_size,dec_units))
    return True

print(grader_onestepdecoder('dot'))
```

```
print(grader_onestepdecoder('general'))
print(grader_onestepdecoder('concat'))
```

```
True
True
True
```

Decoder

```
class Decoder(tf.keras.Model):
    def __init__(self,out_vocab_size, embedding_dim, input_length,
dec_units ,score_fun ,att_units):

      super().__init__()
      #Intialize necessary variables and create an object from the
class onestepdecoder
      self.out_vocab_size=out_vocab_size
      self.embedding_dim=embedding_dim
      self.input_length=input_length
      self.dec_units=dec_units
      self.score_fun=score_fun
      self.att_units=att_units

self.onestepdecoder=One_Step_Decoder(out_vocab_size,embedding_dim,inpu
t_length,dec_units,score_fun,att_units)


    def call(self,
input_to_decoder,encoder_output,decoder_hidden_state,decoder_cell_stat
e ):

        #Initialize an empty Tensor array, that will store the outputs
at each and every time step
        arr_out = tf.TensorArray(tf.float32,size =
tf.shape(input_to_decoder)[1])
        for timestep in range(tf.shape(input_to_decoder)[1]):
          out,state_h,state_c,wt_attentn,vec_cont =
self.onestepdecoder(input_to_decoder[:,timestep:timestep+1],

encoder_output,decoder_hidden_state,decoder_cell_state)

          arr_out = arr_out.write(timestep,out)
        #Create a tensor array as shown in the reference notebook

        #Iterate till the length of the decoder input
            # Call onestepdecoder for each token in decoder_input
            # Store the output in tensorarray
        arr_out = tf.transpose(arr_out.stack(),[1,0,2])
        return arr_out
        # Return the tensor array
```

Grader function - 4

```
def grader_decoder(score_fun):

    '''
        out_vocab_size: Unique words of the target language,
        embedding_dim: output embedding dimension for each word after
embedding layer,
        dec_units: Number of lstm units in decoder,
        att_units: Used in matrix multiplications for scoring
functions in attention class,
        input_length: Length of the target sentence,
        batch_size


    '''

    out_vocab_size=13
    embedding_dim=12
    input_length=11
    dec_units=16
    att_units=16
    batch_size=32


target_sentences=tf.random.uniform(shape=(batch_size,input_length),max
val=10,minval=0,dtype=tf.int32)

encoder_output=tf.random.uniform(shape=[batch_size,input_length,dec_un
its])
    state_h=tf.random.uniform(shape=[batch_size,dec_units])
    state_c=tf.random.uniform(shape=[batch_size,dec_units])

    decoder=Decoder(out_vocab_size, embedding_dim, input_length,
dec_units ,score_fun ,att_units)
    output=decoder(target_sentences,encoder_output, state_h, state_c)
    assert(output.shape==(batch_size,input_length,out_vocab_size))
    return True
print(grader_decoder('dot'))
print(grader_decoder('general'))
print(grader_decoder('concat'))
```

```
True
True
True
```

Encoder Decoder model

```
class encoder_decoder(tf.keras.Model):
  def
__init__(self,encoder_inputs_length,decoder_inputs_length,out_vocab_si
ze,score_fun,att_units):
    #Intialize objects from encoder decoder
    super().__init__()
    self.encoder_inputs_length=encoder_inputs_length
    #self.embedding_dim=embedding_dim
    self.decoder_inputs_length=decoder_inputs_length
    self.out_vocab_size=out_vocab_size
    self.score_fun=score_fun
    self.att_units=att_units

    self.encoder = Encoder(inp_vocab_size=vocab_size_ita+1,
                           embedding_size=100,
                           input_length=self.encoder_inputs_length,
                           lstm_size=self.att_units)
    self.decoder = Decoder(out_vocab_size=vocab_size_eng+1,
                           embedding_dim=100,
                           input_length=self.decoder_inputs_length,
                           dec_units=self.att_units,
                           score_fun=self.score_fun,
                           att_units=self.att_units)


  def call(self,data):
    #Intialize encoder states, Pass the encoder_sequence to the
embedding layer
    # Decoder initial states are encoder final states, Initialize it
accordingly
    # Pass the decoder sequence,encoder_output,decoder states to
Decoder
    # return the decoder output
    initial_state = self.encoder.initialize_states(batch_size)

    inputs,outputs = data[0],data[1]

    encoder_output,encoder_final_state_h,encoder_final_state_c =
self.encoder(inputs,initial_state)
    decoder_output                                             =
self.decoder(outputs,encoder_output,

encoder_final_state_h,encoder_final_state_c)

    return decoder_output
```

Custom loss function

```
#https://www.tensorflow.org/tutorials/text/image_captioning#model
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')


def loss_function(real, pred):
    """ Custom loss function that will not consider the loss for padded zeros.
    why are we using this, can't we use simple sparse categorical crossentropy?
    Yes, you can use simple sparse categorical crossentropy as loss like we did in task-1. But in this loss function we are ignoring the loss
    for the padded zeros. i.e when the input is zero then we donot need to worry what the output is. This padded zeros are added from our end
    during preprocessing to make equal length for all the sentences.

    """


    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)
```

Training

Implement dot function here.

```
# Implement teacher forcing while training your model. You can do it two ways.
# Prepare your data, encoder_input,decoder_input and decoder_output
# if decoder input is
# <start> Hi how are you
# decoder output should be
# Hi How are you <end>
# i.e when you have send <start>-- decoder predicted Hi, 'Hi' decoder predicted 'How' .. e.t.c

# or

# model.fit([train_ita,train_eng],train_eng[:,1:]..)
# Note: If you follow this approach some grader functions might return false and this is fine.
```

```
tf.keras.backend.clear_session()
import datetime
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

Dot Function

```
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M
%S")
tensorboard = tf.keras.callbacks.TensorBoard(log_dir=log_dir,

histogram_freq=1)

model =
encoder_decoder(encoder_inputs_length=20,decoder_inputs_length=20,out_
vocab_size=vocab_size_eng,
                          score_fun='dot',att_units=312)

batch_size = 1024
optimizer = tf.keras.optimizers.Adam()

train_steps = train.shape[0]//1024
valid_steps = validation.shape[0]//1024

callback = [tensorboard]
model.compile(optimizer=optimizer,loss=loss_function)


model.fit(train_dataloader,steps_per_epoch=train_steps,
          epochs=20, validation_data=test_dataloader,
          callbacks=[tensorboard])
model.summary()
```

```
Epoch 1/20
274/274 [==============================] - 281s 987ms/step - loss:
1.9824 - val_loss: 1.7646
Epoch 2/20
274/274 [==============================] - 265s 966ms/step - loss:
1.7084 - val_loss: 1.6280
Epoch 3/20
274/274 [==============================] - 265s 965ms/step - loss:
1.5704 - val_loss: 1.5153
Epoch 4/20
274/274 [==============================] - 264s 963ms/step - loss:
1.4890 - val_loss: 1.4550
Epoch 5/20
274/274 [==============================] - 265s 965ms/step - loss:
1.4162 - val_loss: 1.3660
```

```
Epoch 6/20
274/274 [==============================] - 264s 964ms/step - loss:
1.3202 - val_loss: 1.2727
Epoch 7/20
274/274 [==============================] - 265s 966ms/step - loss:
1.2223 - val_loss: 1.1776
Epoch 8/20
274/274 [==============================] - 264s 962ms/step - loss:
1.1257 - val_loss: 1.0841
Epoch 9/20
274/274 [==============================] - 264s 964ms/step - loss:
1.0308 - val_loss: 0.9967
Epoch 10/20
274/274 [==============================] - 264s 964ms/step - loss:
0.9429 - val_loss: 0.9190
Epoch 11/20
274/274 [==============================] - 264s 964ms/step - loss:
0.8632 - val_loss: 0.8470
Epoch 12/20
274/274 [==============================] - 264s 964ms/step - loss:
0.7898 - val_loss: 0.7828
Epoch 13/20
274/274 [==============================] - 265s 965ms/step - loss:
0.7221 - val_loss: 0.7248
Epoch 14/20
274/274 [==============================] - 264s 965ms/step - loss:
0.6599 - val_loss: 0.6694
Epoch 15/20
274/274 [==============================] - 264s 962ms/step - loss:
0.6015 - val_loss: 0.6202
Epoch 16/20
274/274 [==============================] - ETA: 0s - loss: 0.5495Epoch
17/20
274/274 [==============================] - 263s 959ms/step - loss:
0.5038 - val_loss: 0.5409
Epoch 18/20
274/274 [==============================] - 262s 957ms/step - loss:
0.4629 - val_loss: 0.5073
Epoch 19/20
274/274 [==============================] - 263s 959ms/step - loss:
0.4267 - val_loss: 0.4790
Epoch 20/20
274/274 [==============================] - 263s 960ms/step - loss:
0.3950 - val_loss: 0.4571
Model: "encoder_decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder (Encoder)            multiple                  3171824
```

```
_________________________________________________________________
decoder (Decoder)            multiple                  6298993
=================================================================
Total params: 9,470,817
Trainable params: 9,470,817
Non-trainable params: 0
_________________________________________________________________
```

```
from IPython.display import Image
%load_ext tensorboard
%tensorboard --logdir logs/fit
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6006 (pid 826), started 1:30:10 ago. (Use
'!kill 826' to kill it.)
```

```
<IPython.core.display.Javascript object>
```

```
#https://towardsdatascience.com/intuitive-understanding-of-attention-
mechanism-in-deep-lear
#Refer:
https://www.tensorflow.org/tutorials/text/nmt_with_attention#translate
# refrence taken from
https://www.tensorflow.org/text/tutorials/nmt_with_attention
#refrence taken from https://blog.floydhub.com/attention-mechanism/

# This function plot attention weights
import matplotlib.ticker as ticker
def plot_attention(attention,sentence,predicted_sentence):
  #Refer:
https://www.tensorflow.org/tutorials/text/nmt_with_attention#translate
  fig = plt.figure(figsize=(15,10))
  ax = fig.add_subplot(1,1,1)
  ax.matshow(attention)
  fontdict = {'fontsize':14}

  ax.set_xticklabels(['']+sentence,fontdict=fontdict,rotation=90)
  ax.set_yticklabels(['']+predicted_sentence,fontdict=fontdict)
  ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
  ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

  plt.show()
```

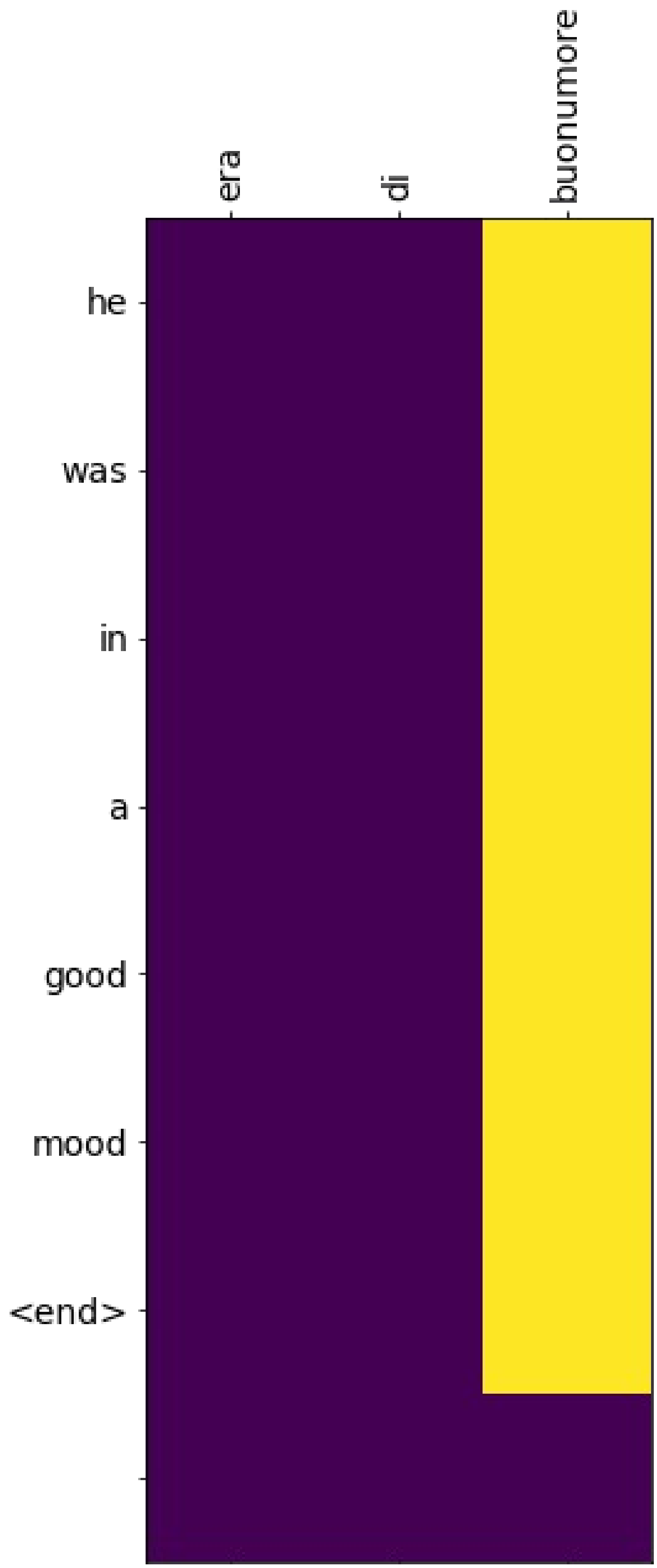
Predict the sentence translation

```
# This function predict Translated sentence and return Translated
sentence and attention weights
def predict(input_sentence):
```

```
  '''
  A. Given input sentence, convert the sentence into integers using
tokenizer used earlier
  B. Pass the input_sequence to encoder. we get encoder_outputs, last
time step hidden and cell state
  C. Initialize index of <start> as input to decoder. and encoder
final states as input_states to decoder
  D. till we reach max_length of decoder or till the model predicted
word <end>:
         predicted_out,state_h,state_c=model.layers[1]
(dec_input,states)
         pass the predicted_out to the dense layer
         update the states=[state_h,state_c]
         And get the index of the word with maximum probability of the
dense layer output, using the tokenizer(word index) get the word and
then store it in a string.
         Update the input_to_decoder with current predictions
  F. Return the predicted sentence
  '''

  encoder_seq = tknizer_ita.texts_to_sequences([input_sentence])
  encoder_seq =
pad_sequences(encoder_seq,maxlen=20,dtype='int32',padding='post')
  initial_state=model.layers[0].initialize_states(1)
  encoder_output, encoder_state_h, encoder_state_c = model.layers[0]
(encoder_seq,initial_state)
  states_values = [encoder_state_h,encoder_state_c]
  pred = []
  cur_vec = tf.expand_dims([tknizer_eng.word_index['<start>']], 0)
  attention_plot = np.zeros((20, 20))


  for i in range(DECODER_SEQ_LEN):
    cur_emb = model.layers[1].embedding(cur_vec)
    infe_output, state_h, state_c =
model.layers[1].lstm(cur_emb,initial_state=states_values)
    attention_weights = tf.reshape(attention_weights,(-1,))
    attention_plot[i] = attention_weights.numpy()
    infe_output=model.layers[2](infe_output)
    states_values = [state_h, state_c]
    if cur_vec == end_index:
      return pred,attention_plot
    cur_vec = np.reshape(np.argmax(infe_output), (1, 1))
    pred.append(cur_vec)
  return pred,attention_plot

DECODER_SEQ_LEN = 20

input_sentence = validation['italian'].sample().item()
print(input_sentence)
```

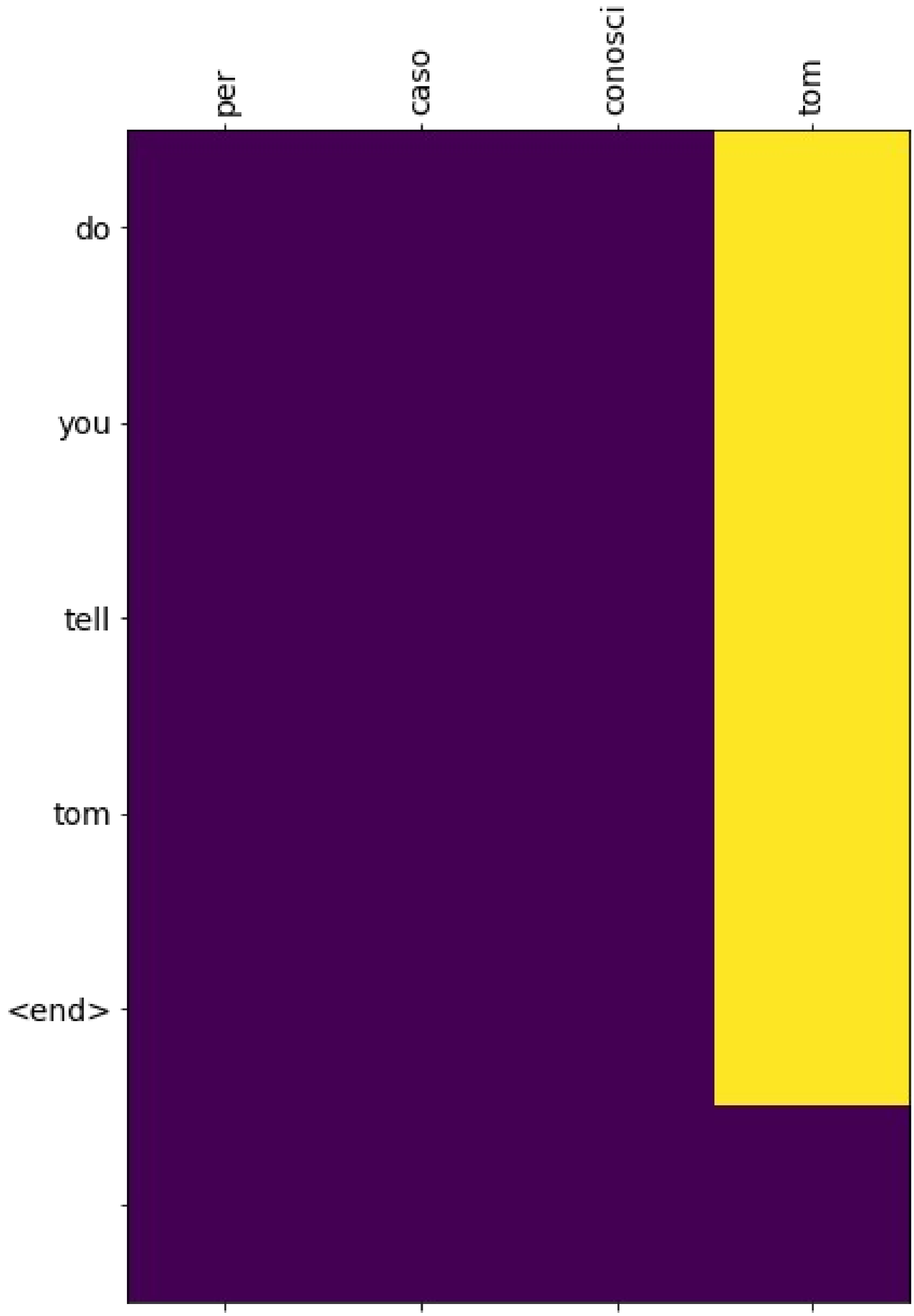
```
pred_sent,attention_plot = predict(input_sentence)

attention_plot = attention_plot[:len(pred_sent.split('
')),:len(input_sentence.split(' '))]
plot_attention(attention_plot,input_sentence.split('
'),pred_sent.split(' '))

# ab=validation['italian'].iloc[2]
# sentence=ab.split()
# predicted_sentence=predicted_sentence.split()
# attention=attention
# plot=plot_attention(attention, sentence, predicted_sentence)
```

```
era di buonumore
```


era
di
buonumore
he
was
in
a
good
mood
<end>

```
English Translated Sentence:  he was in a good mood <end>
```

```
print('\n')
print('English Translated Sentence: ',pred_sent)

DECODER_SEQ_LEN = 20

input_sentence = validation['italian'].sample().item()
pred_sent,attention_plot = predict(input_sentence)

attention_plot = attention_plot[:len(pred_sent.split('
')),:len(input_sentence.split(' '))]
plot_attention(attention_plot,input_sentence.split('
'),pred_sent.split(' '))
```


per
caso
conosci
tom
do
you
tell
tom
<end>

```
******************************************************************************************************************************************************
Italian Sentence:  per caso conosci tom
English Translated Sentence:  do you tell tom <end>

print('\n')
print('*'*150)
print('Italian Sentence: ',input_sentence)
print('English Translated Sentence: ',pred_sent)
```

Calculate BLEU score

```
#Create an object of your custom model.
#Compile and train your model on dot scoring function.
# Visualize few sentences randomly in Test data
# Predict on 1000 random sentences on test data and calculate the average BLEU score of these sentences.
# https://www.nltk.org/_modules/nltk/translate/bleu_score.html
import nltk.translate.bleu_score as bleu
blue_scores=[]
for i in range(1000):
  acutal_sentence = validation['italian'].sample().item()
  predicted_sentence,_=predict(acutal_sentence)
  blue_scores.append(bleu.sentence_bleu(acutal_sentence, predicted_sentence))

print("Average BLUE Score: ",np.average(np.array(blue_scores)))

/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)

Average BLUE Score:  0.712629745688181
```

Model 2

Repeat the same steps for General scoring function

```
#Compile and train your model on general scoring function.
# Visualize few sentences randomly in Test data
# Predict on 1000 random sentences on test data and calculate the average BLEU score of these sentences.
# https://www.nltk.org/_modules/nltk/translate/bleu_score.html

batch_size = 1024
```

```
tf.keras.backend.clear_session()
import datetime
%load_ext tensorboard

The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

General function

```
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M
%S")
tensorboard = tf.keras.callbacks.TensorBoard(log_dir=log_dir,

histogram_freq=1)
model =
encoder_decoder(encoder_inputs_length=20,decoder_inputs_length=20,out_
vocab_size=vocab_size_eng,
                          score_fun='general',att_units=312)
train_steps = train.shape[0]//1024
valid_steps = validation.shape[0]//1024

optimizer = tf.keras.optimizers.Adam()
model.compile(optimizer=optimizer,loss=loss_function)

model.fit(train_dataloader,
          steps_per_epoch=train_steps,
          epochs=20,
          validation_data=test_dataloader,
          callbacks=[tensorboard])
model.summary()

Epoch 1/20
274/274 [==============================] - 285s 1s/step - loss: 1.9817
- val_loss: 1.7631
Epoch 2/20
274/274 [==============================] - 272s 993ms/step - loss:
1.7064 - val_loss: 1.6175
Epoch 3/20
274/274 [==============================] - 272s 993ms/step - loss:
1.5485 - val_loss: 1.4908
Epoch 4/20
274/274 [==============================] - 271s 988ms/step - loss:
1.4615 - val_loss: 1.4228
Epoch 5/20
274/274 [==============================] - 270s 987ms/step - loss:
1.3761 - val_loss: 1.3274
Epoch 6/20
274/274 [==============================] - 270s 986ms/step - loss:
1.2797 - val_loss: 1.2315
Epoch 7/20
```

```
274/274 [==============================] - 271s 988ms/step - loss:
1.1796 - val_loss: 1.1330
Epoch 8/20
274/274 [==============================] - 271s 990ms/step - loss:
1.0800 - val_loss: 1.0409
Epoch 9/20
274/274 [==============================] - 271s 989ms/step - loss:
0.9906 - val_loss: 0.9614
Epoch 10/20
274/274 [==============================] - 271s 987ms/step - loss:
0.9093 - val_loss: 0.8880
Epoch 11/20
274/274 [==============================] - 271s 990ms/step - loss:
0.8328 - val_loss: 0.8216
Epoch 12/20
274/274 [==============================] - 271s 988ms/step - loss:
0.7626 - val_loss: 0.7593
Epoch 13/20
274/274 [==============================] - 271s 990ms/step - loss:
0.6948 - val_loss: 0.6984
Epoch 14/20
274/274 [==============================] - 272s 992ms/step - loss:
0.6326 - val_loss: 0.6447
Epoch 15/20
274/274 [==============================] - 272s 992ms/step - loss:
0.5744 - val_loss: 0.5962
Epoch 16/20
274/274 [==============================] - 271s 989ms/step - loss:
0.5225 - val_loss: 0.5529
Epoch 17/20
274/274 [==============================] - 271s 988ms/step - loss:
0.4762 - val_loss: 0.5162
Epoch 18/20
274/274 [==============================] - 272s 993ms/step - loss:
0.4353 - val_loss: 0.4835
Epoch 19/20
274/274 [==============================] - 271s 989ms/step - loss:
0.3993 - val_loss: 0.4557
Epoch 20/20
274/274 [==============================] - 272s 993ms/step - loss:
0.3686 - val_loss: 0.4328
Model: "encoder_decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
encoder (Encoder)            multiple                  3171824   
_________________________________________________________________
decoder (Decoder)            multiple                  6396649   
=================================================================
```

```
Total params: 9,568,473
Trainable params: 9,568,473
Non-trainable params: 0
_________________________________________________________________
```

```
from IPython.display import Image
%load_ext tensorboard
%tensorboard --logdir logs/fit
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

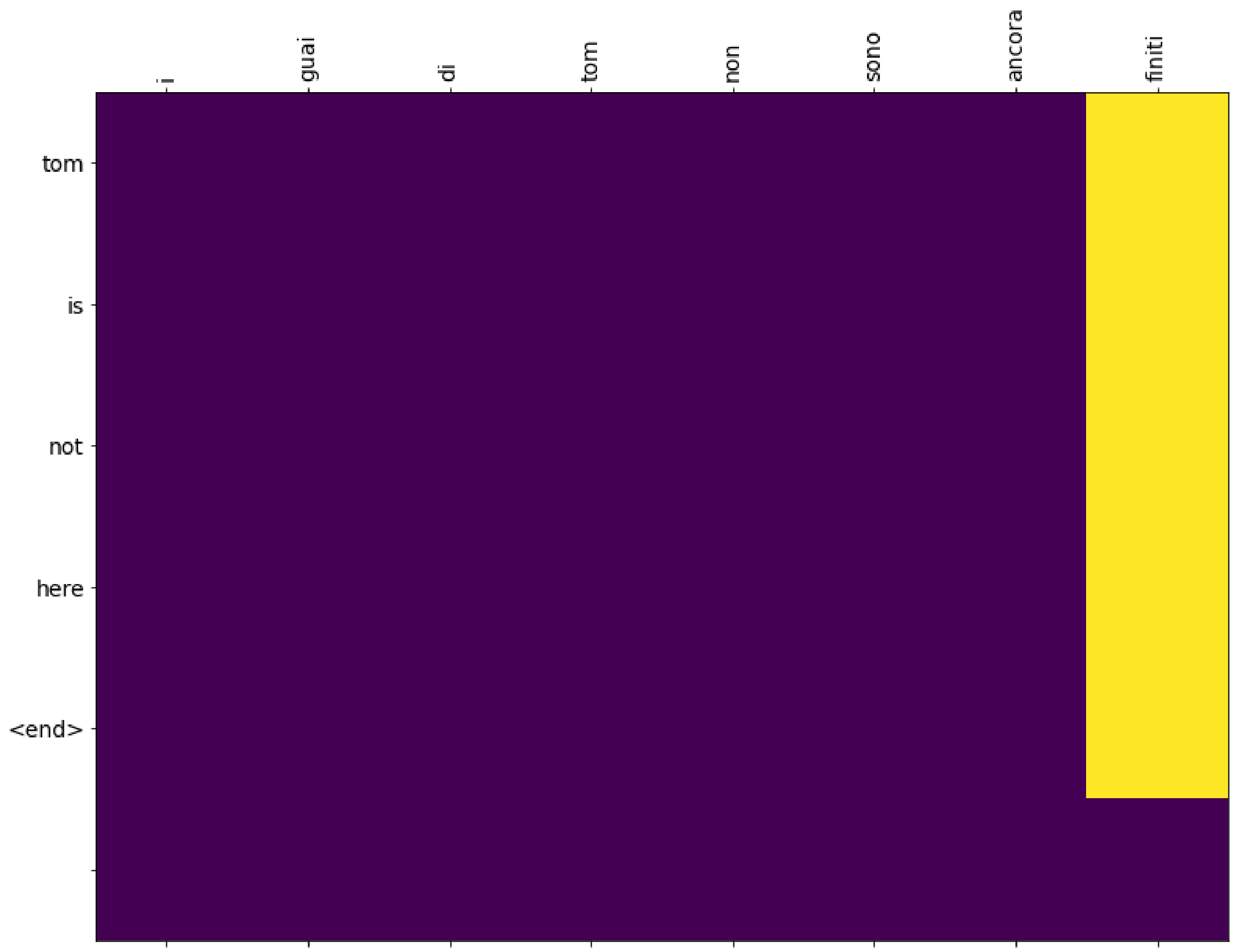
```
Reusing TensorBoard on port 6006 (pid 826), started 3:05:14 ago. (Use
'!kill 826' to kill it.)
```

```
<IPython.core.display.Javascript object>
```

```
# This function predict Translated sentence and return Translated
sentence and attention weights
def predict(input_sentence):

  '''
  A. Given input sentence, convert the sentence into integers using
tokenizer used earlier
  B. Pass the input_sequence to encoder. we get encoder_outputs, last
time step hidden and cell state
  C. Initialize index of <start> as input to decoder. and encoder
final states as input_states to decoder
  D. till we reach max_length of decoder or till the model predicted
word <end>:
         predicted_out,state_h,state_c=model.layers[1]
(dec_input,states)
         pass the predicted_out to the dense layer
         update the states=[state_h,state_c]
         And get the index of the word with maximum probability of the
dense layer output, using the tokenizer(word index) get the word and
then store it in a string.
         Update the input_to_decoder with current predictions
  F. Return the predicted sentence
  '''

  encoder_seq = tknizer_ita.texts_to_sequences([input_sentence])
  encoder_seq =
pad_sequences(encoder_seq,maxlen=20,dtype='int32',padding='post')
  initial_state=model.layers[0].initialize_states(1)
  encoder_output, encoder_state_h, encoder_state_c = model.layers[0]
(encoder_seq,initial_state)
  states_values = [encoder_state_h,encoder_state_c]
  pred = []
  cur_vec = tf.expand_dims([tknizer_eng.word_index['<start>']], 0)
  attention_plot = np.zeros((20, 20))
```

```
  for i in range(DECODER_SEQ_LEN):
    cur_emb = model.layers[1].embedding(cur_vec)
    infe_output, state_h, state_c =
model.layers[1].lstm(cur_emb,initial_state=states_values)
    attention_weights = tf.reshape(attention_weights,(-1,))
    attention_plot[i] = attention_weights.numpy()
    infe_output=model.layers[2](infe_output)
    states_values = [state_h, state_c]
    if cur_vec == end_index:
      return pred,attention_plot
    cur_vec = np.reshape(np.argmax(infe_output), (1, 1))
    pred.append(cur_vec)
  return pred,attention_plot

DECODER_SEQ_LEN = 20

input_sentence = validation['italian'].sample().item()
print(input_sentence)
pred_sent,attention_plot = predict(input_sentence)

attention_plot = attention_plot[:len(pred_sent.split('
')),:len(input_sentence.split(' '))]
plot_attention(attention_plot,input_sentence.split('
'),pred_sent.split(' '))
```

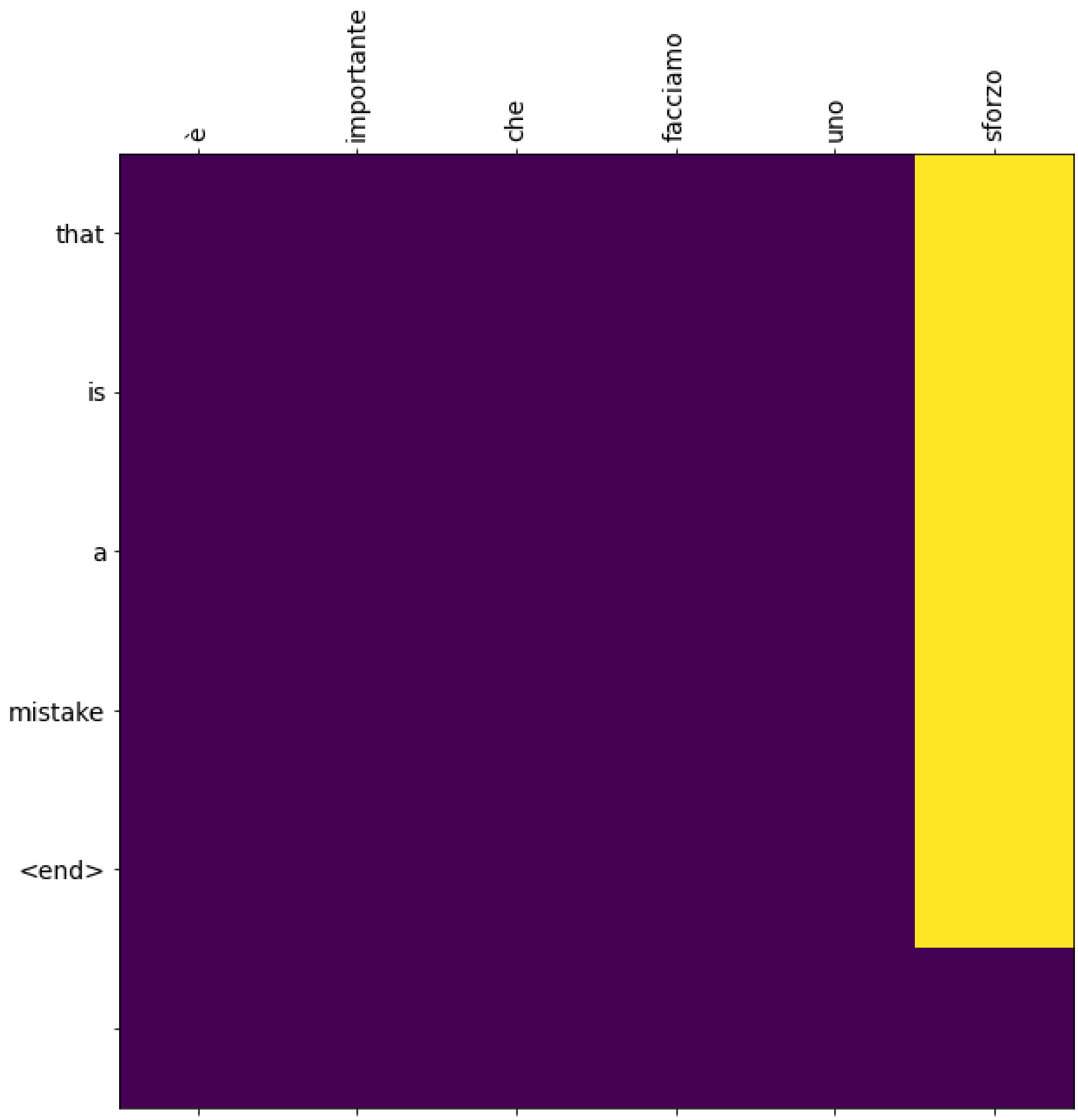
```
i guai di tom non sono ancora finiti
```




```
English Translated Sentence:  tom is not here <end>
```

```
print('\n')
print('English Translated Sentence: ',pred_sent)

DECODER_SEQ_LEN = 20

input_sentence = validation['italian'].sample().item()
print(input_sentence)
pred_sent,attention_plot = predict(input_sentence)

attention_plot = attention_plot[:len(pred_sent.split('
')),:len(input_sentence.split(' '))]
plot_attention(attention_plot,input_sentence.split('
'),pred_sent.split(' '))
```

```
è importante che facciamo uno sforzo
```




```
English Translated Sentence:  that is a mistake <end>
```

```python
print('\n')
print('English Translated Sentence: ',pred_sent)

import nltk.translate.bleu_score as bleu
blue_scores=[]
for i in range(1000):
  acutal_sentence = validation['italian'].sample().item()
```

```
  predicted_sentence,_=predict(acutal_sentence)
  blue_scores.append(bleu.sentence_bleu(acutal_sentence,
predicted_sentence))

print("Average BLUE Score: ",np.average(np.array(blue_scores)))
```

```
/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)

Average BLUE Score:  0.7160862122833141
```

CONCAT Function

```
tf.keras.backend.clear_session()
import datetime
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
batch_size = 1024

log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M
%S")
tensorboard = tf.keras.callbacks.TensorBoard(log_dir=log_dir,

histogram_freq=1)

model =
encoder_decoder(encoder_inputs_length=20,decoder_inputs_length=20,out_
vocab_size=vocab_size_eng,
                          score_fun='concat',att_units=312)

train_steps = train.shape[0]//1024
valid_steps = validation.shape[0]//1024

optimizer = tf.keras.optimizers.Adam()
model.compile(optimizer=optimizer,loss=loss_function)

model.fit(train_dataloader,steps_per_epoch=train_steps,
          epochs=20, validation_data=test_dataloader,
          callbacks=[tensorboard])
model.summary()
```

```
Epoch 1/20
274/274 [==============================] - 340s 1s/step - loss: 1.9861
- val_loss: 1.7736
Epoch 2/20
```

```
274/274 [==============================] - 324s 1s/step - loss: 1.6926
- val_loss: 1.7062
Epoch 3/20
274/274 [==============================] - 324s 1s/step - loss: 1.5086
- val_loss: 1.6299
Epoch 4/20
274/274 [==============================] - 323s 1s/step - loss: 1.3940
- val_loss: 1.5434
Epoch 5/20
274/274 [==============================] - 324s 1s/step - loss: 1.3190
- val_loss: 1.4939
Epoch 6/20
274/274 [==============================] - 324s 1s/step - loss: 1.2302
- val_loss: 1.4273
Epoch 7/20
274/274 [==============================] - 325s 1s/step - loss: 1.1376
- val_loss: 1.3731
Epoch 8/20
274/274 [==============================] - 324s 1s/step - loss: 1.0493
- val_loss: 1.3208
Epoch 9/20
274/274 [==============================] - 324s 1s/step - loss: 0.9655
- val_loss: 1.2794
Epoch 10/20
274/274 [==============================] - 325s 1s/step - loss: 0.8870
- val_loss: 1.2375
Epoch 11/20
274/274 [==============================] - 325s 1s/step - loss: 0.8117
- val_loss: 1.1820
Epoch 12/20
274/274 [==============================] - 324s 1s/step - loss: 0.7400
- val_loss: 1.1437
Epoch 13/20
274/274 [==============================] - 325s 1s/step - loss: 0.6717
- val_loss: 1.1185
Epoch 14/20
274/274 [==============================] - 325s 1s/step - loss: 0.6093
- val_loss: 1.0893
Epoch 15/20
274/274 [==============================] - 324s 1s/step - loss: 0.5529
- val_loss: 1.0494
Epoch 16/20
274/274 [==============================] - 324s 1s/step - loss: 0.5030
- val_loss: 1.0298
Epoch 17/20
274/274 [==============================] - 324s 1s/step - loss: 0.4589
- val_loss: 0.9932
Epoch 18/20
274/274 [==============================] - 324s 1s/step - loss: 0.4208
```

```
- val_loss: 0.9744
Epoch 19/20
274/274 [==============================] - 323s 1s/step - loss: 0.3875
- val_loss: 0.9648
Epoch 20/20
274/274 [==============================] - ETA: 0s - loss:
0.3583Model: "encoder_decoder_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_1 (Encoder)          multiple                  3171824
_________________________________________________________________
decoder_1 (Decoder)          multiple                  6494618
=================================================================
Total params: 9,666,442
Trainable params: 9,666,442
Non-trainable params: 0
_________________________________________________________________
```

```
from IPython.display import Image
%load_ext tensorboard
%tensorboard --logdir logs/fit
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

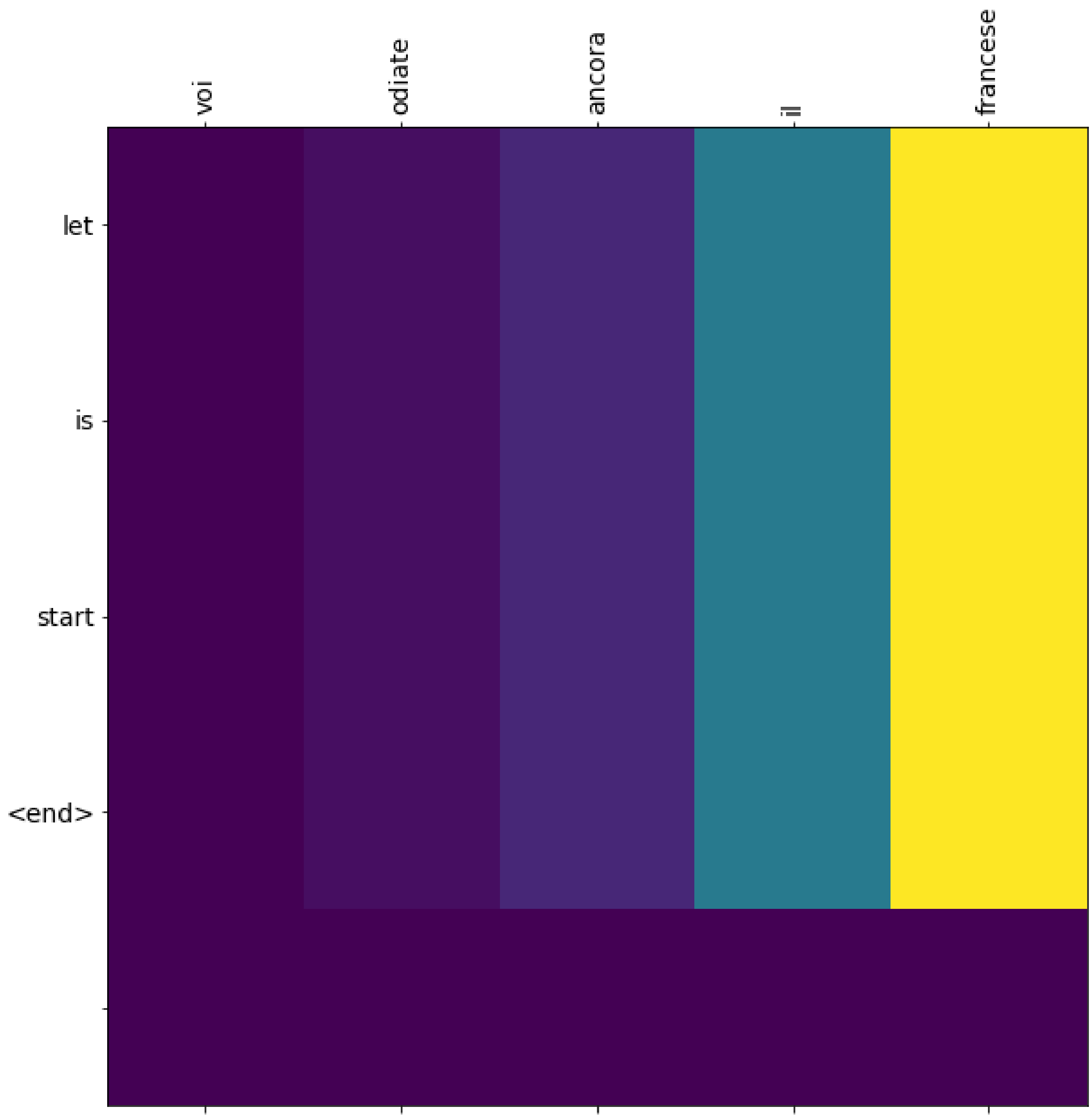
```
Reusing TensorBoard on port 6006 (pid 826), started 5:10:33 ago. (Use
'!kill 826' to kill it.)
```

```
<IPython.core.display.Javascript object>
```

```
# This function predict Translated sentence and return Translated
sentence and attention weights
def predict(input_sentence):

  '''
  A. Given input sentence, convert the sentence into integers using
tokenizer used earlier
  B. Pass the input_sequence to encoder. we get encoder_outputs, last
time step hidden and cell state
  C. Initialize index of <start> as input to decoder. and encoder
final states as input_states to decoder
  D. till we reach max_length of decoder or till the model predicted
word <end>:
         predicted_out,state_h,state_c=model.layers[1]
(dec_input,states)
         pass the predicted_out to the dense layer
         update the states=[state_h,state_c]
         And get the index of the word with maximum probability of the
dense layer output, using the tokenizer(word index) get the word and
```

```
then store it in a string.
         Update the input_to_decoder with current predictions
  F. Return the predicted sentence
  '''

  encoder_seq = tknizer_ita.texts_to_sequences([input_sentence])
  encoder_seq =
pad_sequences(encoder_seq,maxlen=20,dtype='int32',padding='post')
  initial_state=model.layers[0].initialize_states(1)
  encoder_output, encoder_state_h, encoder_state_c = model.layers[0]
(encoder_seq,initial_state)
  states_values = [encoder_state_h,encoder_state_c]
  pred = []
  cur_vec = tf.expand_dims([tknizer_eng.word_index['<start>']], 0)
  attention_plot = np.zeros((20, 20))


  for i in range(DECODER_SEQ_LEN):
    cur_emb = model.layers[1].embedding(cur_vec)
    infe_output, state_h, state_c =
model.layers[1].lstm(cur_emb,initial_state=states_values)
    attention_weights = tf.reshape(attention_weights,(-1,))
    attention_plot[i] = attention_weights.numpy()
    infe_output=model.layers[2](infe_output)
    states_values = [state_h, state_c]
    if cur_vec == end_index:
      return pred,attention_plot
    cur_vec = np.reshape(np.argmax(infe_output), (1, 1))
    pred.append(cur_vec)
  return pred,attention_plot

DECODER_SEQ_LEN = 20

input_sentence = validation['italian'].sample().item()
print(input_sentence)
pred_sent,attention_plot = predict(input_sentence)

attention_plot = attention_plot[:len(pred_sent.split('
')),:len(input_sentence.split(' '))]
plot_attention(attention_plot,input_sentence.split('
'),pred_sent.split(' '))
```

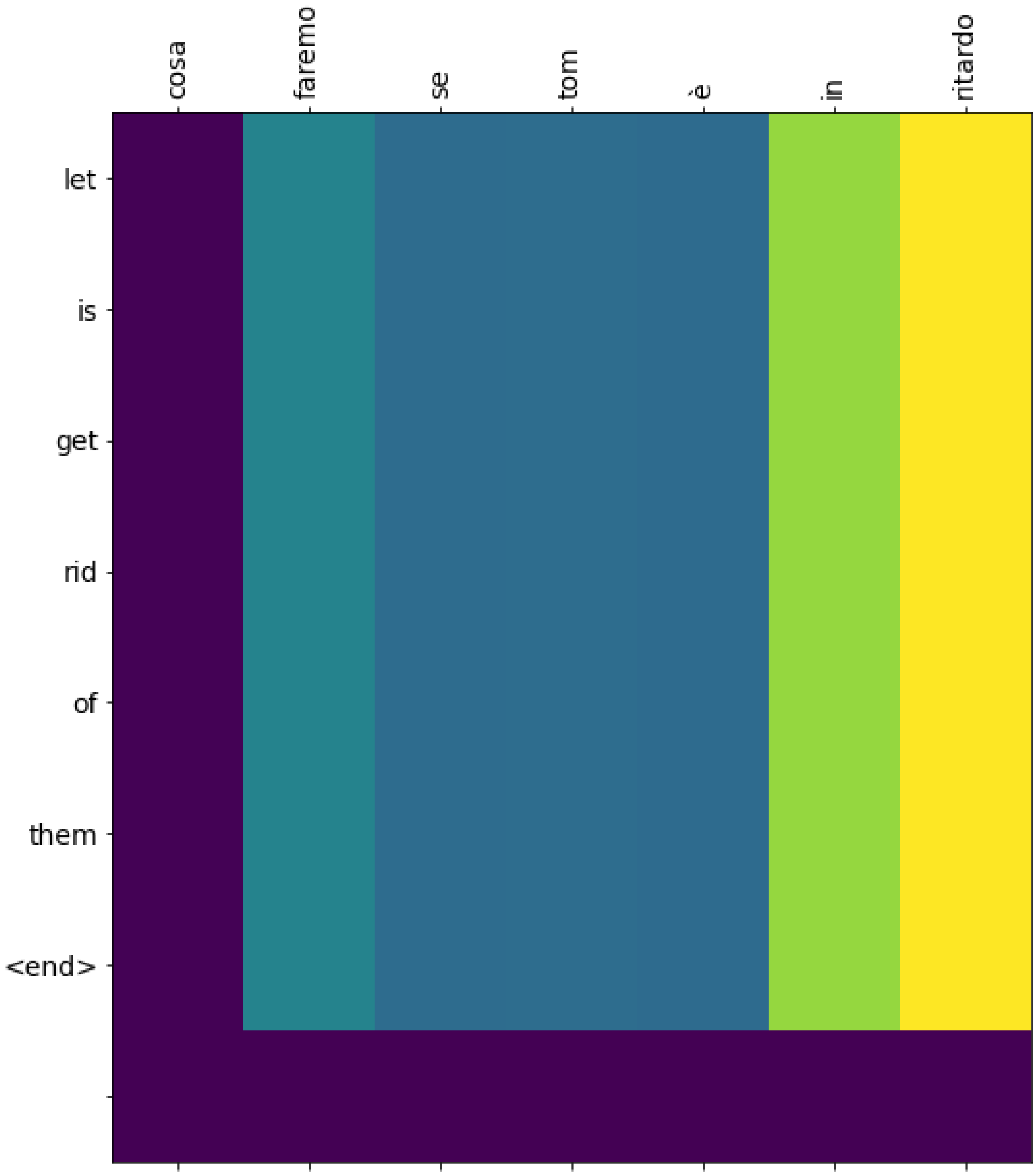
```
voi odiate ancora il francese
```




```
English Translated Sentence:  let is start <end>

print('\n')
print('English Translated Sentence: ',pred_sent)

DECODER_SEQ_LEN = 20

input_sentence = validation['italian'].sample().item()
print(input_sentence)
pred_sent,attention_plot = predict(input_sentence)
```

```
attention_plot = attention_plot[:len(pred_sent.split('
')),:len(input_sentence.split(' '))]
plot_attention(attention_plot,input_sentence.split('
'),pred_sent.split(' '))
```

```
cosa faremo se tom è in ritardo
```



```
English Translated Sentence:  let is get rid of them <end>
```

```python
print('\n')
print('English Translated Sentence: ',pred_sent)
```

```python
#Create an object of your custom model.
#Compile and train your model on dot scoring function.
# Visualize few sentences randomly in Test data
# Predict on 1000 random sentences on test data and calculate the
average BLEU score of these sentences.
# https://www.nltk.org/_modules/nltk/translate/bleu_score.html
import nltk.translate.bleu_score as bleu
blue_scores=[]
for i in range(1000):
  acutal_sentence = validation['italian'].sample().item()
  predicted_sentence,_=predict(acutal_sentence)
  blue_scores.append(bleu.sentence_bleu(acutal_sentence,
predicted_sentence))

print("Average BLUE Score: ",np.average(np.array(blue_scores)))
```

```
/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)
```

```
Average BLUE Score:  0.7558977346895056
```

```python
from prettytable import PrettyTable

p = PrettyTable()

p.field_names = ["Model",'Bleu Score']
p.add_row(["Encoder decoder model", '0.528'])
p.add_row(["Scoring function With Attention-Dot Score",'0.712'])
p.add_row(["Scoring function With Attention-General Score",'0.716'])
p.add_row(["Scoring function With Attention-Concat Score", '0.755'])

print(p)
```

```
+-----------------------------------------------+------------+
|                     Model                     | Bleu Score |
+-----------------------------------------------+------------+
|             Encoder decoder model             |   0.528    |
|   Scoring function With Attention-Dot Score   |   0.712    |
| Scoring function With Attention-General Score |   0.716    |
|  Scoring function With Attention-Concat Score |   0.755    |
+-----------------------------------------------+------------+
```