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

Import Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
import re
import tensorflow as tf
import os
import datetime
from tensorflow.keras.layers import Embedding, LSTM, Dense
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import
ModelCheckpoint,EarlyStopping,LearningRateScheduler,ReduceLROnPlateau,
TensorBoard
from tqdm import tqdm
tqdm.pandas()
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import plot_model
```

Load the data

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

```
Downloading...
From: http://www.manythings.org/anki/ita-eng.zip
To: /content/ita-eng.zip
   0% 0.00/7.88M [00:00<?, ?B/s]   7% 524k/7.88M [00:00<00:01,
5.13MB/s] 100% 7.88M/7.88M [00:00<00:00, 39.9MB/s]
```

```
!mkdir dataset
! unzip /content/ita-eng.zip -d /content/dataset
```

```
Archive:  /content/ita-eng.zip
  inflating: /content/dataset/ita.txt
  inflating: /content/dataset/_about.txt
```

```
with open('/content/dataset/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()
```

```
(358373, 2)
```

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

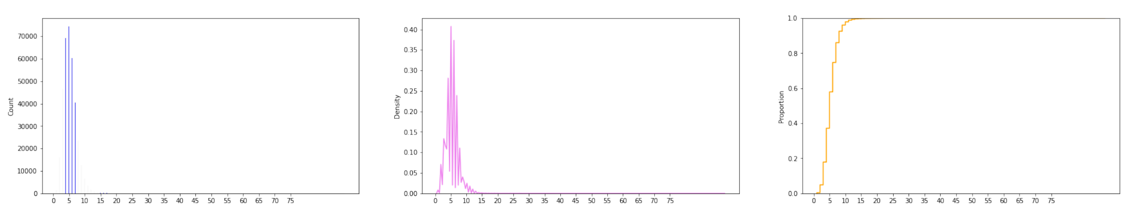
Preprocess data

```
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 between 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'].progress_apply(preprocess)
data['italian'] = data['italian'].progress_apply(preprocess_ita)
data.head()

100%|██████████| 358373/358373 [00:16<00:00, 21109.44it/s]
100%|██████████| 358373/358373 [00:07<00:00, 45963.34it/s]

  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)
```

Lets analyse length of italian sentences

```
fig, axs = plt.subplots(ncols=3,figsize=(30,5))
a = sns.histplot(data = ita_lengths,ax=axs[0],color = 'blue')
a.set_xticks(range(0,80,5))
b = sns.kdeplot(data = ita_lengths,ax=axs[1],color = 'violet')
b.set_xticks(range(0,80,5))
c = sns.ecdfplot(data = ita_lengths,ax=axs[2],color = 'orange')
c.set_xticks(range(00,80,5))
plt.show()
```




```
print('*'*30 + ' Percentiles with span of 10 '+ '*'*30)
for i in range(0,101,10):
    print(i,np.percentile(ita_lengths, i))
```

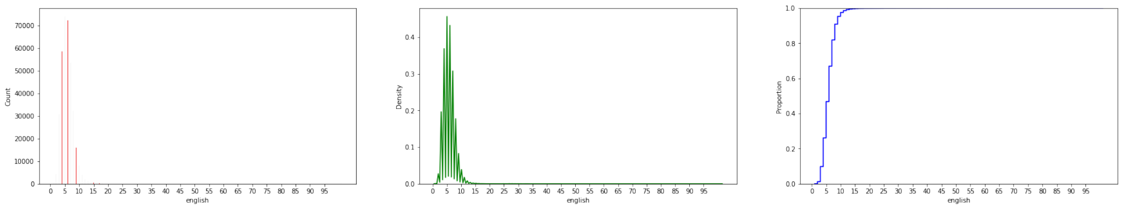
```
print('*'*30 + ' Percentiles fom 90 to 100 with span of 1 '+ '*'*30)
for i in range(90,101):
    print(i,np.percentile(ita_lengths, i))
print('*'*30 + ' Percentiles fom 99 to 100 with span of 0.1 '+ '*'*30)
for i in [99.1,99.2,99.3,99.4,99.5,99.6,99.7,99.8,99.9,100]:
    print(i,np.percentile(ita_lengths, i))
```

```
****************************** Percentiles with span of 10
******************************
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
****************************** Percentiles fom 90 to 100 with span of
1 ******************************
90 8.0
91 8.0
92 8.0
93 9.0
94 9.0
95 9.0
96 9.0
97 10.0
98 11.0
99 12.0
100 92.0
****************************** Percentiles fom 99 to 100 with span of
0.1 ******************************
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
```

Lets analyse length of english sentences

```
fig, axs = plt.subplots(ncols=3,figsize=(30,5))
a = sns.histplot(data = eng_lengths,ax=axs[0],color = 'red')
a.set_xticks(range(0,100,5))
```

```
b = sns.kdeplot(data = eng_lengths,ax=axs[1],color = 'green')
b.set_xticks(range(0,100,5))
c = sns.ecdfplot(data = eng_lengths,ax=axs[2],color = 'blue')
c.set_xticks(range(00,100,5))
plt.show()
```




```
print('*'*30 + ' Percentiles with span of 10 '+ '*'*30)
for i in range(0,101,10):
    print(i,np.percentile(eng_lengths, i))
print('*'*30 + ' Percentiles fom 90 to 100 with span of 1 '+ '*'*30)
for i in range(90,101):
    print(i,np.percentile(eng_lengths, i))
print('*'*30 + ' Percentiles fom 99 to 100 with span of 0.1 '+ '*'*30)
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))
```

```
****************************** Percentiles with span of 10
******************************
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
****************************** Percentiles fom 90 to 100 with span of
1 ******************************
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
****************************** Percentiles fom 99 to 100 with span of
0.1 ******************************
```

```
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
```

Lets select words less than 20 as for both italian and english we have 99.8% of values less than 20.

Let prepare the dataset for teacher forcing mechanism implementation

```
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()
```

```
<ipython-input-11-0af9942294a0>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data['english_inp'] = '<start> ' + data['english'].astype(str)
<ipython-input-11-0af9942294a0>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data['english_out'] = data['english'].astype(str) + ' <end>'

   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  \
15117                                      me la spedisca
13499                                    io volevo questo
269096                       una tigre è fuggita dallo zoo
183084                           vi dispiace se ci sediamo
249877                    fa più freddo del solito stasera
180536                                      con chi parlavi
240665                       tom si lavò il viso e le mani
302530  lufficio postale si trova a mezzo miglio di di...
70704                                        perché mi ami
231809                         lui è pignolo con le regole

                                        english_inp  \
15117                             <start> send it to me
13499                             <start> i wanted this
269096        <start> a tiger has escaped from the zoo
183084              <start> do you mind if we sit down
249877         <start> it is colder than usual tonight
180536               <start> who were you talking with
240665           <start> tom washed his face and hands
302530  <start> the post office is half a mile away
70704                      <start> why do you love me
231809          <start> he is a stickler for the rules

                                      english_out
15117                             send it to me <end>
13499                             i wanted this <end>
269096        a tiger has escaped from the zoo <end>
183084              do you mind if we sit down <end>
249877         it is colder than usual tonight <end>
180536               who were you talking with <end>
240665           tom washed his face and hands <end>
302530  the post office is half a mile away <end>
70704                      why do you love me <end>
231809          he is a stickler for the rules <end>
```

Getting train and test

```
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>'
```

```
(286292, 3) (71574, 3)
```

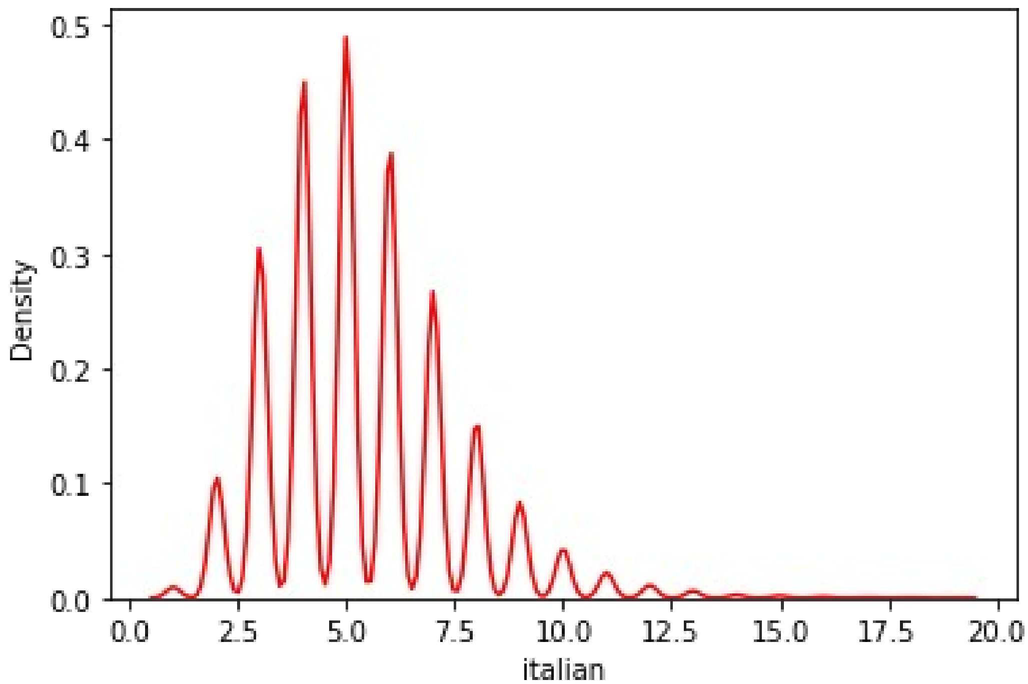
```
train.head()
```

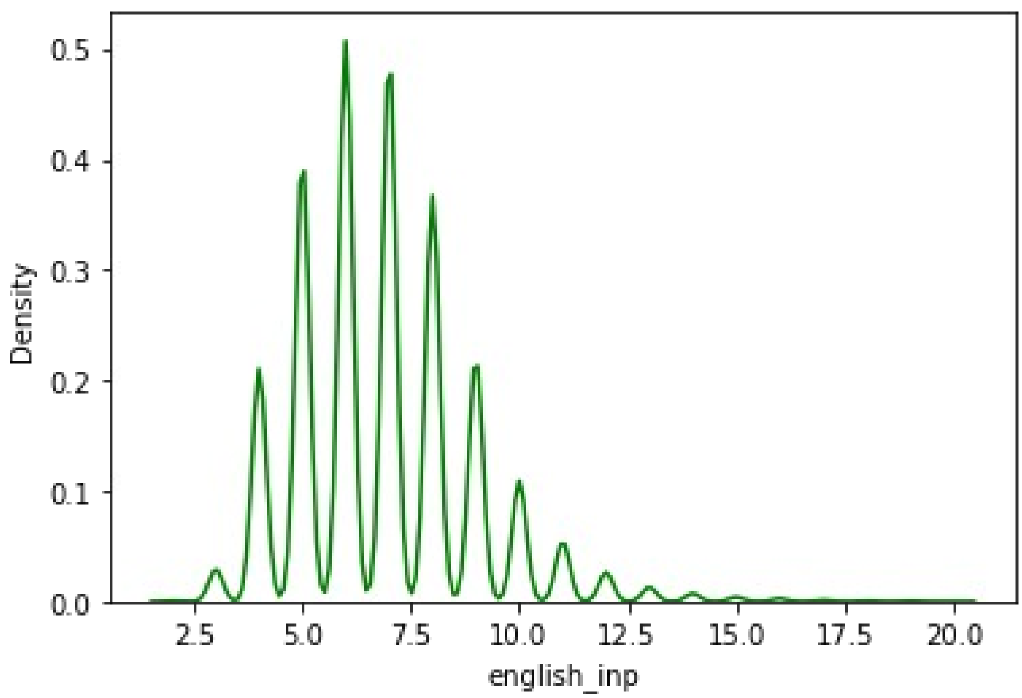
```
                                          italian  \
310857   tom si raddrizzò il nodo sulla cravatta
319680      non ho idea di cosa succederà domani
63199                                 starò con te
260952       penso di sapere cosa succede adesso
246299  io sono soddisfatta della mia nuova casa

                                               english_inp  \
310857  <start> tom straightened the knot on his tie <...
319680   <start> i have no idea what will happen tomorrow
63199                          <start> i will stay with you
260952              <start> i think i know what happens now
246299               <start> i am pleased with my new house

                                             english_out
310857  tom straightened the knot on his tie <end> <end>
319680    i have no idea what will happen tomorrow <end>
63199                          i will stay with you <end>
260952              i think i know what happens now <end>
246299               i am pleased with my new house <end>
```

```
validation.head()
```

```
                             italian
english_inp  \
8808                 io andrò a vedere              <start> i will go
look
185203  lho comprato a dieci dollari  <start> i bought it for 10
dollars
14571                   diventò virale               <start> it went
viral
6020             il tempo è tempestoso                <start> it is
stormy
39635              è così emozionante           <start> it is so
exciting

                             english_out
8808                 i will go look <end>
185203  i bought it for 10 dollars <end>
14571                 it went viral <end>
6020                   it is stormy <end>
39635             it is so exciting <end>
```

```
ita_lengths = train['italian'].str.split().apply(len)
eng_lengths = train['english_inp'].str.split().apply(len)
sns.kdeplot(ita_lengths,color = 'red')
plt.show()
sns.kdeplot(eng_lengths,color = 'green')
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 > from filters so that we can retain those symbols.

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

```
vocab_size_eng=len(tokenizer_eng.word_index.keys())
print(vocab_size_eng)
vocab_size_ita=len(tokenizer_ita.word_index.keys())
print(vocab_size_ita)
```

```
13139
26838
```

```
tokenizer_eng.word_index['<start>'], tokenizer_eng.word_index['<end>']
```

```
(1, 10396)
```

Creating embeddings for english sentences

```
#!gdown https://nlp.stanford.edu/data/glove.6B.zip
```

```
#!mkdir glove
#!unzip /content/glove.6B.zip -d glove
```

```
'''embeddings_index = dict()
f = open('/content/glove/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 tokenizer_eng.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector'''
```

```
{"type":"string"}
```

Implement custom encoder decoder

```
tf.keras.backend.clear_session()
```

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):
        super(Encoder,self).__init__()
        self.inp_vocab_size = inp_vocab_size
        self.lstm_size = lstm_size
        #Initialize Embedding layer
        self.embedding = Embedding(input_dim =
inp_vocab_size,output_dim = embedding_size,input_length =
input_length,name="embedding_layer_encoder")
        #Intialize Encoder LSTM layer
        self.lstm =
LSTM(lstm_size,return_state=True,return_sequences=True,kernel_initiali
zer = 'he_normal',kernel_regularizer=
tf.keras.regularizers.L2(),name="Encoder_LSTM")



    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_embed = self.embedding(input_sequence)
        self.lstm_output, self.lstm_state_h,self.lstm_state_c =
self.lstm(input_embed,initial_state = [states[0], states[1]])

        return self.lstm_output, self.lstm_state_h,self.lstm_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]
      '''
      self.hidden_state = tf.zeros((batch_size, self.lstm_size))
      self.cell_state = tf.zeros((batch_size, self.lstm_size))
```

```
        return self.hidden_state,self.cell_state
```

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):
        super(Decoder, self).__init__()
        self.out_vocab_size = out_vocab_size
        self.embedding_size = embedding_size
        self.lstm_size = lstm_size
        self.input_length = input_length
```

```
        #Initialize Embedding layer
        self.embedding = Embedding(input_dim =
self.out_vocab_size,output_dim = self.embedding_size,input_length =
self.input_length,name="embedding_layer_decoder")
        #Intialize Decoder LSTM layer
        self.lstm = LSTM(self.lstm_size,
return_sequences=True,return_state=True,kernel_initializer =
'he_normal',kernel_regularizer=
tf.keras.regularizers.L2(),name="Decoder_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
        '''
        target_embedd = self.embedding(input_sequence)

self.decoder_output,self.decoder_hidden_state,self.decoder_cell_state
= self.lstm(target_embedd, initial_state = [initial_states[0],
initial_states[1]])
        return self.decoder_output, self.decoder_hidden_state,
self.decoder_cell_state
```

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 Encoder_decoder(tf.keras.Model):

    def
__init__(self,encoder_inputs_length,decoder_inputs_length,encoder_voca
b_size,decoder_vocab_size,encoder_embed_size,decoder_embed_size,output
_vocab_size,):

        #Create encoder object
        #Create decoder object
        #Intialize Dense layer(out_vocab_size) with
activation='softmax'
        super().__init__()
        self.encoder = Encoder(inp_vocab_size =
encoder_vocab_size,embedding_size = encoder_embed_size,input_length =
encoder_inputs_length,lstm_size=256)
        self.decoder = Decoder(out_vocab_size =
decoder_vocab_size,embedding_size= decoder_embed_size,input_length =
decoder_inputs_length,lstm_size=256)
        self.dense = Dense(output_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
        '''
```

```
        encoder_input,decoder_input = data[0], data[1]
        #Intializing encoder initial states
        initial_state = self.encoder.initialize_states(1024)
        encoder_output, encoder_h, encoder_c =
self.encoder(encoder_input,initial_state)
        decoder_output,_,_ = self.decoder(decoder_input, [encoder_h,
encoder_c])
        output = self.dense(decoder_output)
        return output



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

    def __getitem__(self, i):
        self.encoder_seq =
self.tokenizer_ita.texts_to_sequences([self.encoder_inps[i]]) # need
to pass list of values
        self.decoder_inp_seq =
self.tokenizer_eng.texts_to_sequences([self.decoder_inps[i]])
        self.decoder_out_seq =
self.tokenizer_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, tokenizer_ita, tokenizer_eng, 20)
test_dataset  = Dataset(validation, tokenizer_ita, tokenizer_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)
```

```
#Create an object of encoder_decoder Model class,
model  = Encoder_decoder(encoder_inputs_length = 20,
                         decoder_inputs_length = 20,
                         encoder_vocab_size = vocab_size_ita + 1,
                         decoder_vocab_size = vocab_size_eng + 1,
                         encoder_embed_size = 100,
                         decoder_embed_size = 100,
                         output_vocab_size = vocab_size_eng)
# Compile the model and fit the model
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
metrics = ['accuracy'])

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

filepath="model_save/weights-{epoch:02d}-{val_loss:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss',
verbose=1, save_best_only=True, mode='auto')
```

```
earlystop = EarlyStopping(monitor='val_loss', min_delta=0.1,
patience=2, verbose=1)

# Load the TensorBoard notebook extension
%load_ext tensorboard
log_dir = os.path.join("logs",'fits',
datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback =
tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1,write_
graph=True)
%reload_ext tensorboard
```

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

```
model.fit(x=train_dataloader,validation_data = test_dataloader,epochs
= 15,steps_per_epoch = train_steps,validation_steps =
valid_steps,callbacks = [tensorboard_callback])
```

```
Epoch 1/15
279/279 [==============================] - 86s 299ms/step - loss:
14.0092 - accuracy: 0.6766 - val_loss: 2.9735 - val_accuracy: 0.7075
Epoch 2/15
279/279 [==============================] - 82s 293ms/step - loss:
2.0277 - accuracy: 0.7102 - val_loss: 1.7054 - val_accuracy: 0.7130
Epoch 3/15
279/279 [==============================] - 82s 293ms/step - loss:
1.6421 - accuracy: 0.7204 - val_loss: 1.6036 - val_accuracy: 0.7254
Epoch 4/15
279/279 [==============================] - 82s 293ms/step - loss:
1.5623 - accuracy: 0.7331 - val_loss: 1.5223 - val_accuracy: 0.7444
Epoch 5/15
279/279 [==============================] - 82s 293ms/step - loss:
1.4569 - accuracy: 0.7589 - val_loss: 1.4057 - val_accuracy: 0.7686
Epoch 6/15
279/279 [==============================] - 82s 294ms/step - loss:
1.3394 - accuracy: 0.7773 - val_loss: 1.2933 - val_accuracy: 0.7838
Epoch 7/15
279/279 [==============================] - 82s 293ms/step - loss:
1.2336 - accuracy: 0.7904 - val_loss: 1.1984 - val_accuracy: 0.7945
Epoch 8/15
279/279 [==============================] - 82s 294ms/step - loss:
1.1399 - accuracy: 0.8004 - val_loss: 1.1194 - val_accuracy: 0.8030
Epoch 9/15
279/279 [==============================] - 82s 294ms/step - loss:
1.0668 - accuracy: 0.8087 - val_loss: 1.0608 - val_accuracy: 0.8099
Epoch 10/15
279/279 [==============================] - 82s 294ms/step - loss:
1.0054 - accuracy: 0.8161 - val_loss: 1.0022 - val_accuracy: 0.8179
Epoch 11/15
279/279 [==============================] - 82s 294ms/step - loss:
```

```
0.9451 - accuracy: 0.8239 - val_loss: 0.9428 - val_accuracy: 0.8258
Epoch 12/15
279/279 [==============================] - 82s 294ms/step - loss:
0.8893 - accuracy: 0.8308 - val_loss: 0.9041 - val_accuracy: 0.8305
Epoch 12: early stopping
```

```
<keras.callbacks.History at 0x7fa497df9490>
```

```
#!mkdir saved_models
#model.save('/content/saved_models/encoder_decoder',save_format='tf')
```

```
#!zip -r /content/saved_models/encoder_decoder.zip
/content/saved_models/encoder_decoder
```

```
#from google.colab import files
#files.download("/content/saved_models/encoder_decoder.zip")
```

```
#!pip install -U --no-cache-dir gdown --pre
#!gdown 1a_eWynLJTJaWrGNNk_cyt7N5xeaHOz1c
```

```
#!mkdir encoder_decoder
```

```
 #!unzip  /content/encoder_decoder.zip -d /content/encoder_decoder
```

```
#model =
tf.keras.models.load_model('/content/encoder_decoder/content/saved_mod
els/encoder_decoder')
```

```
model.summary()
```

```
Model: "encoder_decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder_1 (Encoder)         multiple                  3049468

 decoder_1 (Decoder)         multiple                  1679568

 dense (Dense)               multiple                  3376723

=================================================================
Total params: 8,105,759
Trainable params: 8,105,759
Non-trainable params: 0
_________________________________________________________________
```

```
plot_model(model,show_shapes=True,show_layer_activations = True)
```

```
%tensorboard --logdir '/content/logs'
```

Output hidden; open in https://colab.research.google.com to view.

```
# make lookup table
eng_index_word={}
eng_word_index={}
for key,value in tokenizer_eng.word_index.items():
  eng_index_word[value]=key
  eng_word_index[key]=value


eng_word_index['<end>']
```

10396

```
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
  '''
  # convert sentence into intger encoding
  enc_in = tokenizer_ita.texts_to_sequences([input_sentence])
  # pad sequence
  enc_pad = pad_sequences(enc_in,maxlen=20, dtype='int32',
padding='post')
  # initialize state
  initial_state = model.layers[0].initialize_states(1)
  # get output and states from encoder
  enc_output, enc_state_h, enc_state_c = model.layers[0]
(enc_pad,initial_state)
  # hidden and cell states
  states_values = [enc_state_h, enc_state_c]
  # initialize empty target sentence
  sent = ''
```

```
  # <start>:1 , <end>:2 --
  target_word = np.array([[1]])
  # iterate to the range of decoder_sequence length
  for i in range(20):
      # get output and hidden,cell state from decoder model
      [infer_output, state_h, state_c] = model.layers[1](target_word,
states_values)
      # get output from dense layer
      infer_output = model.layers[2](infer_output)
      # store states_values
      states_values = [state_h, state_c]
      # np.argmax(infe_output) will be a single value, which
represents the the index of predicted word
      # but to pass this data into next time step embedding layer, we
are reshaping it into (1,1) shape
      target_word = np.reshape(np.argmax(infer_output), (1, 1))
      #print(target_word)
      # make sentence from predicted word by looking in lookup table
reverse dictionary
      if int(target_word) != 0:
        sent=sent +' '+ eng_index_word[int(target_word)]
      end_word = eng_word_index['<end>']
      if int(target_word) == end_word:
        sent = ' '.join(sent.split()[:-1])
        break
  return sent

predict('come va')
```

{"type":"string"}

```
predict('perché hai saltato il pranzo oggi')
```

{"type":"string"}

```
eng_index_word[int(1)]
```

{"type":"string"}

```
# 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.
import nltk.translate.bleu_score as bleu
import warnings
warnings.filterwarnings('ignore')
np.random.seed(42)
BLUE_Scores = []
for i in tqdm(range(1000)):
    index = np.random.randint(1,68677)
    # Original sentence
    original_sent = [validation['english_out'].iloc[index].split(),]
```

```
    # predicted sentence
    prediction = predict(validation['italian'].iloc[index]).split()
    # compute BLUE
    blue_score = bleu.sentence_bleu(original_sent, prediction)
    # append score
    BLUE_Scores.append(blue_score)
```

```
100%|██████████| 1000/1000 [00:43<00:00, 23.09it/s]
```

```
print('Average Blue Score is ==> ',np.average(BLUE_Scores))
```

```
Average Blue Score is ==>  0.04660040329081343
```

```
from prettytable import PrettyTable
import random
x = PrettyTable()

x.field_names = ["Input Sentence", "Original Sentence", "Predicted
Sentence"]

for i in range(10):
    # Original sentence
    original_sent = [train['english_out'].iloc[i].split(),]
    # predicted sentence
    prediction = predict(train['italian'].iloc[i]).split()
    # pretty table
    x.add_row([train['italian'].iloc[i],
               train['english_out'].iloc[i],
               predict(train['italian'].iloc[i])])
print(x)
```

```
+--------------------------------------------------------
+------------------------------------------------------
+------------------------------------------------------+
|                     Input Sentence                    |
Original Sentence                  |                 Predicted
Sentence                 |
+--------------------------------------------------------
+------------------------------------------------------
+------------------------------------------------------+
|        tom si raddrizzò il nodo sulla cravatta        |   tom
straightened the knot on his tie <end> <end>  |          tom opened
his hair on the table          |
|          non ho idea di cosa succederà domani          |    i have no
idea what will happen tomorrow <end>   |        i am not idea i am
going to do that          |
|                      starò con te                      |
i will stay with you <end>             |                  i will be
help                   |
|          penso di sapere cosa succede adesso          |        i
```

```
think i know what happens now <end>        |             i think you
should be tom              |
|        io sono soddisfatta della mia nuova casa       |         i am
pleased with my new house <end>        |              i am in my
friend of tom              |
|                  tom non ha un gatto                  |
tom does not have a cat <end>             |              tom does not
have a car               |
|               dovè laltra metà dei soldi              |      where
is the other half of the money <end>     |             where is the
nearest books              |
|                  tom non può contare                  |
tom can not count <end>                |                  tom can not
eat                      |
| penso che tom stia solamente facendo finta di dormire |  i think tom
is only pretending to be asleep <end>  |           i think tom is not
to be here               |
|   questo è il libro di cui stavamo parlando io e tom  | this is the
book tom and i were talking about <end> | this is the best who i have
to do that i have seen |
+--------------------------------------------------------
+------------------------------------------------------
+------------------------------------------------------+
```

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

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

    def 
__init__(self,inp_vocab_size,embedding_size,lstm_size,input_length):
      super(Encoder,self).__init__()
      self.inp_vocab_size = inp_vocab_size
      self.lstm_size = lstm_size
      #Initialize Embedding layer
      self.embedding = Embedding(input_dim = inp_vocab_size, 
output_dim = embedding_size, input_length = 
input_length,name="embedding_layer_encoder")
      #Intialize Encoder LSTM layer
      self.lstm = 
LSTM(lstm_size,return_state=True,return_sequences=True,kernel_initiali
zer = 'he_normal',kernel_regularizer=
tf.keras.regularizers.L2(),name="Attention_Encoder_LSTM")
```

```python
    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 -- All encoder_outputs, last time steps hidden and 
cell state
      '''
      input_embed = self.embedding(input_sequence)
      self.lstm_output, self.lstm_state_h,self.lstm_state_c = 
self.lstm(input_embed,initial_state = [states[0], states[1]])

      return self.lstm_output, self.lstm_state_h,self.lstm_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]
      '''
      self.h = tf.zeros((batch_size, self.lstm_size))
      self.c = tf.zeros((batch_size, self.lstm_size))
      return self.h,self.c
```

Grader function - 1

```python
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(Attention,self).__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 self.scoring_function == 'general':
      # Intialize variables needed for General score function here
      self.W1 = tf.keras.layers.Dense(att_units)

    elif self.scoring_function == 'concat':
      # Intialize variables needed for Concat score function here
      self.W1 = tf.keras.layers.Dense(att_units)
      self.W2 = tf.keras.layers.Dense(att_units)
      self.V = tf.keras.layers.Dense(1)


  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)
    '''

    decoder_hidden_state = tf.expand_dims(decoder_hidden_state,axis=1)

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


    elif self.scoring_function == 'general':
        # Implement General score function here
        score = self.W1(encoder_output)
        score = tf.keras.layers.Dot(axes=(2,2))([score,
decoder_hidden_state])

    elif self.scoring_function == 'concat':
        # Implement General score function here
        score = self.V(tf.nn.tanh(self.W1(decoder_hidden_state) +
self.W2(encoder_output)))


    # softmax layer to get attention weights
    # attention_weights shape == (batch_size, input_length, 1)
    attention_weights = tf.nn.softmax(score, axis=1)

    # multiply attention weights with encoder_output
    # context_vector shape after sum == (batch_size, att_units)
    context_vector = attention_weights * encoder_output
    # reduce sum to get final context vector
    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights
```

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(One_Step_Decoder,self).__init__()
      self.score_fun = score_fun
      self.att_units = att_units
      # Initialize decoder embedding layer, LSTM and any other objects
needed
      #Initialize Embedding layer
      self.embedding = Embedding(input_dim = tar_vocab_size,output_dim
= embedding_dim,input_length =
input_length,name="embedding_layer_decoder")
      #Intialize Decoder LSTM layer
      self.lstm1 = LSTM(dec_units,
return_sequences=True,return_state=True,kernel_initializer =
'he_normal',kernel_regularizer= tf.keras.regularizers.L2(),
name="Decoder_LSTM")
      self.lstm2 = LSTM(dec_units,
return_sequences=True,return_state=True,kernel_initializer =
'he_normal',kernel_regularizer= tf.keras.regularizers.L2(),
name="Decoder_LSTM")
      # Initialize fully connected
```

```
      self.fc = tf.keras.layers.Dense(tar_vocab_size)
      self.attention = Attention(self.score_fun,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
    '''
    # find embedding
    # shape of input_to_decoder after passing through embedding layer
== (batch_size, 1, embedding_dim)
    input_to_decoder = self.embedding(input_to_decoder)
    # find decoder output, hidden state and cell state
    # shape of decoder_hidden_state == (batch_size,dec_units)
    d_output,d_hidden_state,d_cell_state = self.lstm1(encoder_output,
initial_state = [state_h, state_c])

    # get context vector and attention weights
    # encoder_output shape == (batch_size, max_length, dec_units)
    # context_vector shape before concatenate ==
(batch_size,dec_units)
    context_vector, attention_weights = self.attention(d_hidden_state,
encoder_output)
    # concat context vector and input_to_decoder
    # shape after concat == (batch_size, 1, embedding_dim + dec_units)
    input_to_decoder = tf.concat([tf.expand_dims(context_vector,axis =
1),input_to_decoder], axis=-1)
    # passing concatenated vector through LSTM
    d_output , hidden_s, hidden_c = self.lstm2(input_to_decoder)
    # d_output shape == (batch_size * 1, dec_units)
    d_output = tf.reshape(d_output, (-1, d_output.shape[2]))
    # pass decoder output to dense layer
    # output shape == (batch_size, vocab)
    output = self.fc(d_output)
    return output,hidden_s,hidden_c,attention_weights,context_vector
```

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):
    #Intialize necessary variables and create an object from the class 
onestepdecoder
    super(Decoder,self).__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.onestep_decoder = 
One_Step_Decoder(self.out_vocab_size,self.embedding_dim,self.input_len
gth,self.dec_units,self.score_fun,self.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
    #Create a tensor array as shown in the reference notebook
    all_outputs = 
tf.TensorArray(tf.float32,size=self.input_length,name='output_array')

    #Iterate till the length of the decoder input
    for timestep in range(self.input_length):
      # Call onestepdecoder for each token in decoder_input
      
output,decoder_hidden_state,decoder_cell_state,attention_weights,conte
x_vector = 
self.onestep_decoder(input_to_decoder[:,timestep:timestep+1],encoder_o
utput,decoder_hidden_state,decoder_cell_state)
      # storing all outputs in output tensor array.
      all_outputs = all_outputs.write(timestep,output)

    all_outputs = tf.transpose(all_outputs.stack(),[1,0,2])
    # Return the tensor array
    return all_outputs
```

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,enc_vocab_size,enc_embedding_dim,enc_units,enc_input_len
```

```
gth,out_vocab_size, dec_embedding_dim,dec_input_length,
dec_units,score_fun ,att_units):
    #Intialize objects from encoder decoder
    super(encoder_decoder,self).__init__()

    # Intialize objects from encoder
    self.encoder =
Encoder(enc_vocab_size,enc_embedding_dim,enc_units,enc_input_length)

    # Intialize object from decoder
    self.decoder = Decoder(out_vocab_size,
dec_embedding_dim,dec_input_length, dec_units,score_fun, 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
    encoder_input,decoder_input = data[0], data[1]
    # initialize encoder state
    initial_state = self.encoder.initialize_states(1024)
    # getting encoder output and hidden states
    encoder_output, encoder_h, encoder_c =
self.encoder(encoder_input,initial_state)
    decoder_h = encoder_h
    decoder_c = encoder_c
    decoder_output  =
self.decoder(decoder_input,encoder_output,decoder_h, decoder_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_)
```

Lets generate train and test dataset

```
train_dataset = Dataset(train, tokenizer_ita, tokenizer_eng, 20)
test_dataset  = Dataset(validation, tokenizer_ita, tokenizer_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)
```

Training

```
tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()
```

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.


#Create an object of encoder_decoder Model class,
model  = encoder_decoder(enc_vocab_size = vocab_size_ita +
1,enc_embedding_dim = 100,enc_units = 256,enc_input_length =
20,out_vocab_size = vocab_size_eng + 1,
                         dec_embedding_dim = 100,dec_input_length =
20,dec_units = 256,score_fun = 'dot',att_units = 256)

# Compile the model and fit the model
model.compile(optimizer = 'adam',loss = loss_function,metrics =
['accuracy'])

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

# tensorboard callback
log_dir = '/content/drive/MyDrive/attention_assignment/logs2'
tensorboard_cb = TensorBoard(log_dir=log_dir)

# model checkpoint
checkpoint_filepath =
'/content/drive/MyDrive/attention_assignment/model_save/best_model.h5'
model_checkpoint_callback =
tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_filepath,save_w
eights_only=False,monitor='val_loss',save_best_only=True)
model.fit(x=train_dataloader,validation_data = test_dataloader,epochs
= 15,steps_per_epoch = train_steps,validation_steps =
valid_steps,callbacks = [tensorboard_cb])
model.summary()

Epoch 1/15
279/279 [==============================] - 294s 871ms/step - loss:
14.7172 - accuracy: 0.0497 - val_loss: 2.4690 - val_accuracy: 0.0500
Epoch 2/15
279/279 [==============================] - 228s 817ms/step - loss:
1.9063 - accuracy: 0.0597 - val_loss: 1.6983 - val_accuracy: 0.0661
Epoch 3/15
279/279 [==============================] - 227s 814ms/step - loss:
1.6203 - accuracy: 0.0740 - val_loss: 1.5623 - val_accuracy: 0.0812
Epoch 4/15
279/279 [==============================] - 227s 813ms/step - loss:
1.5775 - accuracy: 0.0837 - val_loss: 1.5176 - val_accuracy: 0.0850
Epoch 5/15
279/279 [==============================] - 227s 813ms/step - loss:
1.5046 - accuracy: 0.0870 - val_loss: 1.4896 - val_accuracy: 0.0880
Epoch 6/15
279/279 [==============================] - 227s 812ms/step - loss:
1.4796 - accuracy: 0.0887 - val_loss: 1.4700 - val_accuracy: 0.0896
```

```
Epoch 7/15
279/279 [==============================] - 227s 812ms/step - loss:
1.4611 - accuracy: 0.0897 - val_loss: 1.4540 - val_accuracy: 0.0900
Epoch 8/15
279/279 [==============================] - 227s 814ms/step - loss:
1.4476 - accuracy: 0.0902 - val_loss: 1.4437 - val_accuracy: 0.0905
Epoch 9/15
279/279 [==============================] - 227s 813ms/step - loss:
1.4365 - accuracy: 0.0907 - val_loss: 1.4329 - val_accuracy: 0.0911
Epoch 10/15
279/279 [==============================] - 227s 814ms/step - loss:
1.4272 - accuracy: 0.0910 - val_loss: 1.4244 - val_accuracy: 0.0911
Epoch 11/15
279/279 [==============================] - 227s 814ms/step - loss:
1.4187 - accuracy: 0.0915 - val_loss: 1.4175 - val_accuracy: 0.0907
Epoch 12/15
279/279 [==============================] - 227s 813ms/step - loss:
1.4111 - accuracy: 0.0921 - val_loss: 1.4141 - val_accuracy: 0.0915
Epoch 13/15
279/279 [==============================] - 227s 815ms/step - loss:
1.4033 - accuracy: 0.0928 - val_loss: 1.4035 - val_accuracy: 0.0924
Epoch 14/15
279/279 [==============================] - 229s 820ms/step - loss:
1.3868 - accuracy: 0.0978 - val_loss: 1.3681 - val_accuracy: 0.1050
Epoch 15/15
279/279 [==============================] - 229s 822ms/step - loss:
1.3458 - accuracy: 0.1077 - val_loss: 1.3331 - val_accuracy: 0.1105
Model: "encoder_decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder (Encoder)           multiple                  3049468

 decoder (Decoder)           multiple                  5844004

=================================================================
Total params: 8,893,472
Trainable params: 8,893,472
Non-trainable params: 0
_________________________________________________________________
```

```
%tensorboard --logdir
'/content/drive/MyDrive/attention_assignment/logs2'
```

```
Output hidden; open in https://colab.research.google.com to view.
```

Inference

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
  # function for plotting the attention weights
  fig = plt.figure(figsize=(10,10))
  ax = fig.add_subplot(1, 1, 1)
  ax.matshow(attention, cmap='viridis')

  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

```
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 onestepdecoder.
  D. till we reach max_length of decoder or till the model predicted
word <end>:
         predictions, input_states, attention_weights =
model.layers[1].onestepdecoder(input_to_decoder, encoder_output,
input_states)
         Save the attention weights
         And get the word using the tokenizer(word index) and then
store it in a string.
  E. Call plot_attention(#params)
  F. Return the predicted sentence
  '''

  # convert sentence into intger encoding
  encoder_in = tokenizer_ita.texts_to_sequences([input_sentence])

  # pad sequence
  encoder_in = pad_sequences(encoder_in,maxlen=20, dtype='int32',
padding='post')
  encoder_in = tf.convert_to_tensor(encoder_in)
```

```
  # initialize state
  initial_state = model.layers[0].initialize_states(1)

  # define empty sentence
  result = ''

  # Initialize index of <start>:1
  dec_input = np.array([[1]])

  # define empty array for attention plot
  attention_plot = np.zeros((20,20))

  # get output and states from encoder
  enc_output, enc_state_h, enc_state_c = model.layers[0]
(encoder_in,initial_state)
  decoder_h = enc_state_h
  decoder_c = enc_state_c

  for t in range(20):
      # call onestep decoder
      predictions, decoder_h, decoder_c,attention_weights,c_vector =
model.layers[1].onestep_decoder(
          dec_input, enc_output, decoder_h,decoder_c)

      # storing the attention weights to plot later on
      attention_weights = tf.reshape(attention_weights, (-1, ))
      attention_plot[t] = attention_weights.numpy()

      # predict index of word
      predicted_id = tf.argmax(predictions[0]).numpy()

      result += tokenizer_eng.index_word[predicted_id] + ' '

      if tokenizer_eng.index_word[predicted_id] == '<end>':
          return result, input_sentence , attention_plot

      # the predicted ID is fed back into the model
      dec_input = tf.expand_dims([predicted_id], 0)

  return result, input_sentence, attention_plot
```

Function to plot attention plot

```
def translate(sentence):
  result, sentence, attention_plot = predict(sentence)

  print('Input: %s' % (sentence))
  print('Predicted translation: {}'.format(result))

  attention_plot = attention_plot[:len(result.split('
```

```
')), :len(sentence.split(' '))]
  plot_attention(attention_plot, sentence.split(' '), result.split('
'))
```

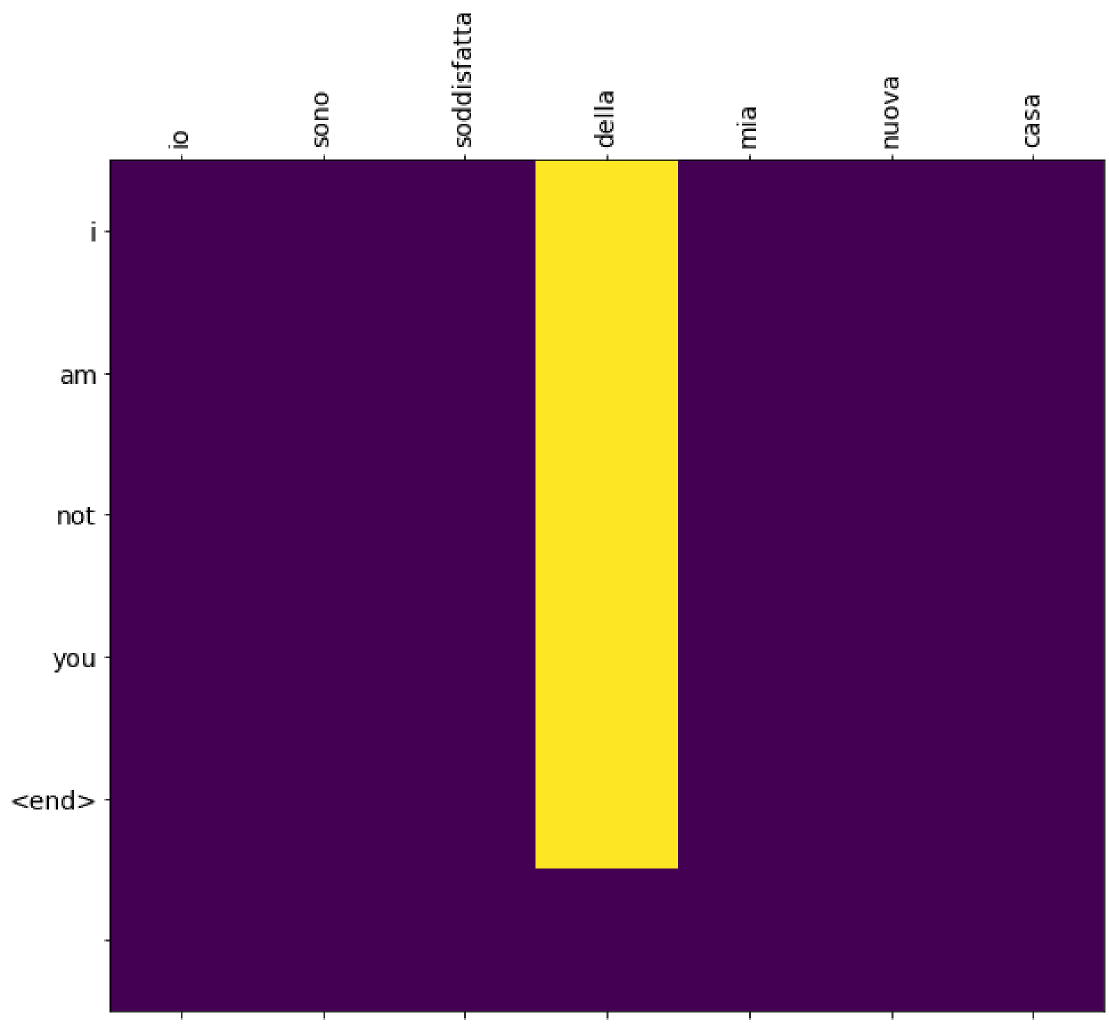
Example 1:

```
train.iloc[4]
```

```
italian        io sono soddisfatta della mia nuova casa
english_inp      <start> i am pleased with my new house
english_out        i am pleased with my new house <end>
Name: 246299, dtype: object
```

```
translate(train.iloc[4].italian)
```

```
Input: io sono soddisfatta della mia nuova casa
Predicted translation: i am not you <end>
```



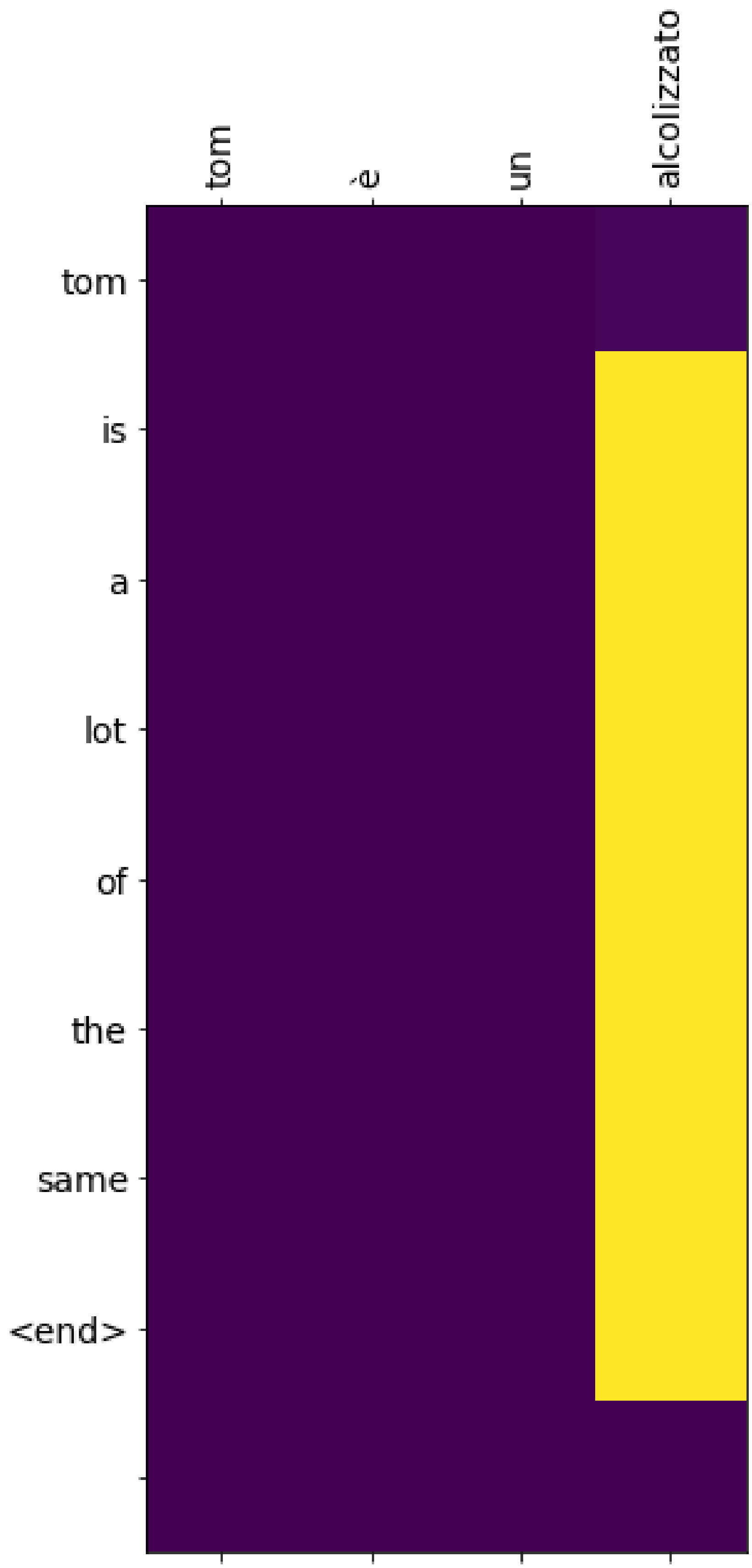
Example: 2

```
train.iloc[10]
```

```
italian                 tom è un alcolizzato
english_inp    <start> tom is an alcoholic
english_out      tom is an alcoholic <end>
Name: 82157, dtype: object
```

```
translate(train.iloc[10].italian)
```

```
Input: tom è un alcolizzato
Predicted translation: tom is a lot of the same <end>
```


tom
è
un
alcolizzato
tom
is
a
lot
of
the
same
<end>

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

#Sample example
import nltk.translate.bleu_score as bleu
reference = ['i am groot'.split(),] # the original
translation = 'it is ship'.split() # trasilated using model
print('BLEU score: {}'.format(bleu.sentence_bleu(reference,
translation)))
```

```
BLEU score: 0
```

Calculate Average BLEU for dot scoring attention model

```
np.random.seed(42)
import warnings
warnings.filterwarnings('ignore')
BLUE_Scores_dot= []
for i in tqdm(range(1000)):
    index = np.random.randint(1,68677)
    # Original sentence
    original_sent = [validation['english_out'].iloc[index].split(),]
    # predicted sentence
    prediction,_,_ = predict(validation['italian'].iloc[index])
    prediction = prediction.split()
    # compute BLUE
    blue_score = bleu.sentence_bleu(original_sent, prediction)
    # append score
    BLUE_Scores_dot.append(blue_score)

print('Average BLUE score using dot scoring is ===>
',np.mean(BLUE_Scores_dot))
```

```
100%|██████████| 1000/1000 [02:30<00:00,  6.64it/s]

Average BLUE score using dot scoring is ===>  7.38118681249641e-80
```

Repeat the same steps for General scoring function

```
tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()

#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

#Create an object of encoder_decoder Model class,
model_general  = encoder_decoder(enc_vocab_size = vocab_size_ita + 1,
                        enc_embedding_dim = 100,
                        enc_units = 256,
                        enc_input_length = 20,
                        out_vocab_size = vocab_size_eng + 1,
                        dec_embedding_dim = 100,
                        dec_input_length = 20,
                        dec_units = 256,
                        score_fun = 'general',
                        att_units = 256
                        )

# Compile the model and fit the model
optimizer = tf.keras.optimizers.Adam()
model_general.compile(optimizer = optimizer,loss =
loss_function,metrics = ['accuracy'])

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

# tensorboard callback
log_dir = '/content/drive/MyDrive/attention_assignment/logs_general'
tensorboard_cb = TensorBoard(log_dir=log_dir)
# model checkpoint
checkpoint_filepath =
'/content/drive/MyDrive/attention_assignment/model_save/general_model.
h5'
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor='val_loss',
    save_best_only=True)

model_general.fit(x=train_dataloader,
          validation_data = test_dataloader,
          epochs = 15,
          steps_per_epoch = train_steps,
          validation_steps = valid_steps,
          callbacks = [tensorboard_cb]
          )
model_general.summary()
```

```
Epoch 1/15
279/279 [==============================] - 305s 918ms/step - loss:
14.6580 - accuracy: 0.0497 - val_loss: 2.4478 - val_accuracy: 0.0500
Epoch 2/15
```

```
279/279 [==============================] - 243s 870ms/step - loss:
1.9533 - accuracy: 0.0610 - val_loss: 1.7180 - val_accuracy: 0.0680
Epoch 3/15
279/279 [==============================] - 242s 867ms/step - loss:
1.6324 - accuracy: 0.0743 - val_loss: 1.5725 - val_accuracy: 0.0793
Epoch 4/15
279/279 [==============================] - 242s 868ms/step - loss:
1.5632 - accuracy: 0.0827 - val_loss: 2.5535 - val_accuracy: 0.0843
Epoch 5/15
279/279 [==============================] - 241s 865ms/step - loss:
2.0034 - accuracy: 0.0871 - val_loss: 1.7372 - val_accuracy: 0.0882
Epoch 6/15
279/279 [==============================] - 242s 866ms/step - loss:
1.6710 - accuracy: 0.0886 - val_loss: 1.6173 - val_accuracy: 0.0891
Epoch 7/15
279/279 [==============================] - 242s 866ms/step - loss:
1.5800 - accuracy: 0.0893 - val_loss: 1.5473 - val_accuracy: 0.0896
Epoch 8/15
279/279 [==============================] - 242s 866ms/step - loss:
1.5212 - accuracy: 0.0903 - val_loss: 1.4994 - val_accuracy: 0.0905
Epoch 9/15
279/279 [==============================] - 241s 865ms/step - loss:
1.4804 - accuracy: 0.0908 - val_loss: 1.4657 - val_accuracy: 0.0908
Epoch 10/15
279/279 [==============================] - 243s 869ms/step - loss:
1.4515 - accuracy: 0.0914 - val_loss: 1.4416 - val_accuracy: 0.0920
Epoch 11/15
279/279 [==============================] - 242s 866ms/step - loss:
1.4307 - accuracy: 0.0920 - val_loss: 1.4252 - val_accuracy: 0.0921
Epoch 12/15
279/279 [==============================] - 242s 866ms/step - loss:
1.4157 - accuracy: 0.0924 - val_loss: 1.4147 - val_accuracy: 0.0924
Epoch 13/15
279/279 [==============================] - 242s 866ms/step - loss:
1.4053 - accuracy: 0.0929 - val_loss: 1.4052 - val_accuracy: 0.0923
Epoch 14/15
279/279 [==============================] - 241s 865ms/step - loss:
1.3975 - accuracy: 0.0930 - val_loss: 1.3970 - val_accuracy: 0.0929
Epoch 15/15
279/279 [==============================] - 241s 865ms/step - loss:
1.3908 - accuracy: 0.0933 - val_loss: 1.3926 - val_accuracy: 0.0930
Model: "encoder_decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder (Encoder)           multiple                  3049468

 decoder (Decoder)           multiple                  5909796

=================================================================
```

```
Total params: 8,959,264
Trainable params: 8,959,264
Non-trainable params: 0
_________________________________________________________________
```

```
%tensorboard --logdir
'/content/drive/MyDrive/attention_assignment/logs_general'
```

```
Output hidden; open in https://colab.research.google.com to view.
```

```
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 onestepdecoder.
    D. till we reach max_length of decoder or till the model predicted
word <end>:
            predictions, input_states, attention_weights =
model.layers[1].onestepdecoder(input_to_decoder, encoder_output,
input_states)
            Save the attention weights
            And get the word using the tokenizer(word index) and then
store it in a string.
    E. Call plot_attention(#params)
    F. Return the predicted sentence
    '''

    # convert sentence into intger encoding
    encoder_in = tokenizer_ita.texts_to_sequences([input_sentence])

    # pad sequence
    encoder_in = pad_sequences(encoder_in,maxlen=20, dtype='int32',
padding='post')
    encoder_in = tf.convert_to_tensor(encoder_in)

    # initialize state
    initial_state = model_general.layers[0].initialize_states(1)

    # define empty sentence
    result = ''

    # Initialize index of <start>:1
    dec_input = np.array([[1]])

    # define empty array for attention plot
    attention_plot = np.zeros((20,20))
```

```
    # get output and states from encoder
    enc_output, enc_state_h, enc_state_c = model_general.layers[0]
(encoder_in,initial_state)
    decoder_h = enc_state_h
    decoder_c = enc_state_c

    for t in range(20):
        # call onestep decoder
        predictions, decoder_h, decoder_c,attention_weights,c_vector =
model_general.layers[1].onestep_decoder(
            dec_input, enc_output, decoder_h,decoder_c)

        # storing the attention weights to plot later on
        attention_weights = tf.reshape(attention_weights, (-1, ))
        attention_plot[t] = attention_weights.numpy()

        # predict index of word
        predicted_id = tf.argmax(predictions[0]).numpy()

        result += tokenizer_eng.index_word[predicted_id] + ' '

        if tokenizer_eng.index_word[predicted_id] == '<end>':
            return result, input_sentence , attention_plot

        # the predicted ID is fed back into the model
        dec_input = tf.expand_dims([predicted_id], 0)

    return result, input_sentence, attention_plot
```

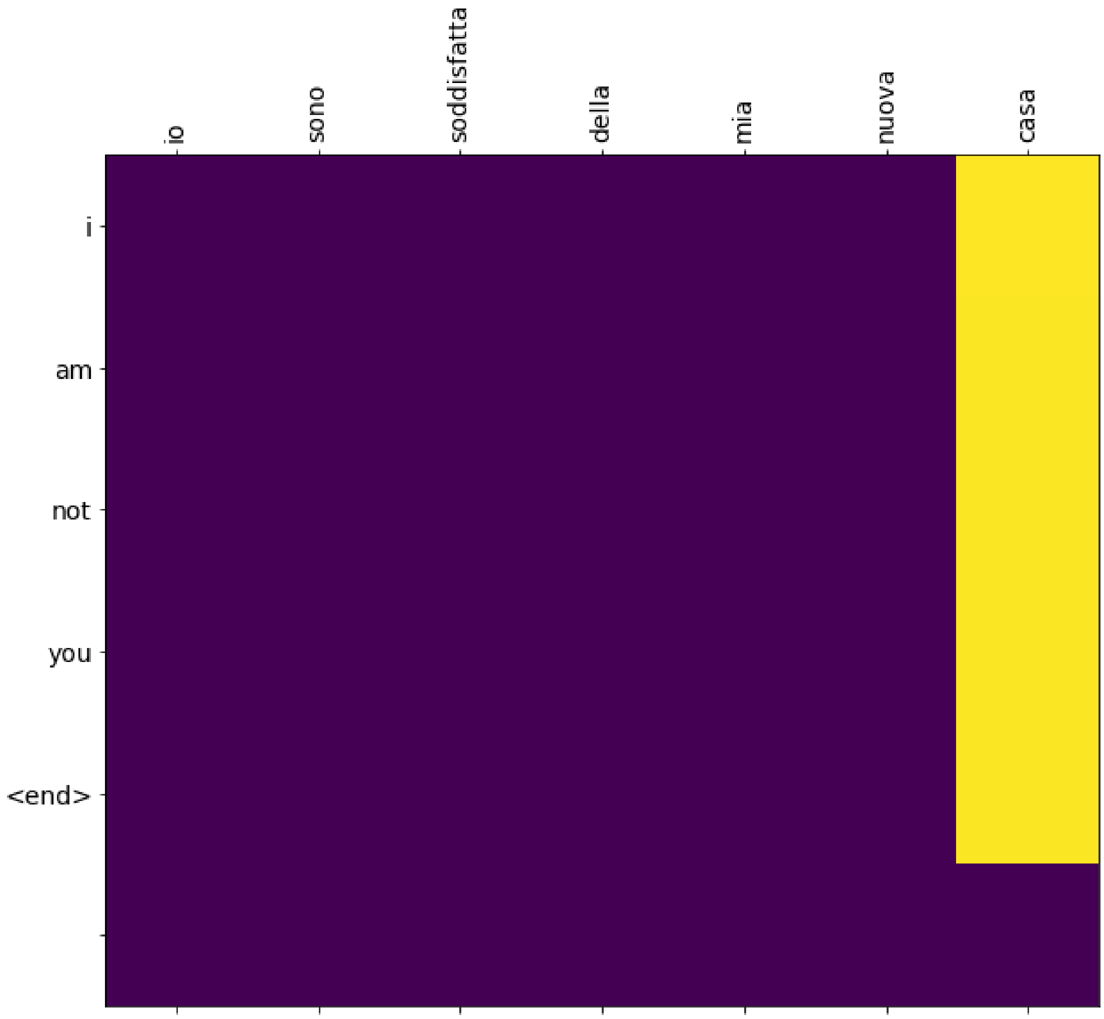
Example 1:

```
train.iloc[4]
```

```
italian        io sono soddisfatta della mia nuova casa
english_inp      <start> i am pleased with my new house
english_out        i am pleased with my new house <end>
Name: 246299, dtype: object
```

```
translate(train.iloc[4].italian)
```

```
Input: io sono soddisfatta della mia nuova casa
Predicted translation: i am not you <end>
```



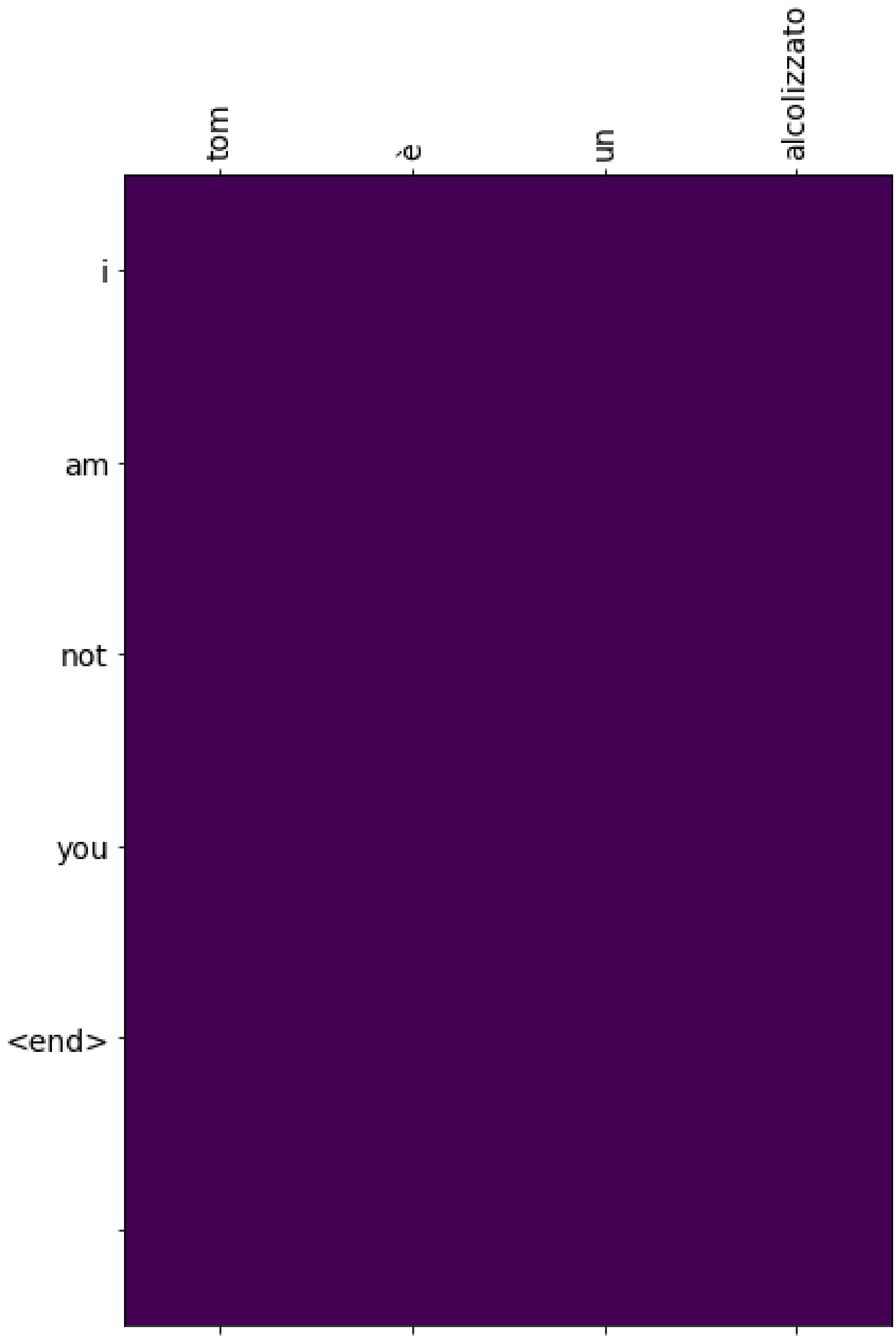
Example: 2

```
train.iloc[10]
```

```
italian                tom è un alcolizzato
english_inp    <start> tom is an alcoholic
english_out      tom is an alcoholic <end>
Name: 82157, dtype: object
```

```
translate(train.iloc[10].italian)
```

```
Input: tom è un alcolizzato
Predicted translation: i am not you <end>
```



Calculate Average BLEU score for random 1000 samples (general scoring)

```
import nltk.translate.bleu_score as bleu

np.random.seed(42)
BLEU_Scores_general= []
for i in tqdm(range(1000)):
    index = np.random.randint(1,68677)
    # Original sentence
    original_sent = [validation['english_out'].iloc[index].split(),]
    # predicted sentence
    prediction,_,_ = predict(validation['italian'].iloc[index])
    prediction = prediction.split()
    # compute BLUE
    blue_score = bleu.sentence_bleu(original_sent, prediction)
    # append score
    BLEU_Scores_general.append(blue_score)

print('Average BLUE score using general scoring is ===>
',np.mean(BLEU_Scores_general))
```

```
100%|██████████| 1000/1000 [01:16<00:00, 13.15it/s]

Average BLUE score using general scoring is ===>  5.95870057583763e-80
```

Repeat the same steps for Concat scoring function

```
#Compile and train your model on concat 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

tf.keras.backend.clear_session()
tf.compat.v1.reset_default_graph()

#Create an object of encoder_decoder Model class,
model_concat  = encoder_decoder(enc_vocab_size = vocab_size_ita + 1,
                         enc_embedding_dim = 100,
                         enc_units = 256,
                         enc_input_length = 20,
                         out_vocab_size = vocab_size_eng + 1,
                         dec_embedding_dim = 100,
                         dec_input_length = 20,
                         dec_units = 256,
                         score_fun = 'concat',
                         att_units = 256
                         )

# Compile the model and fit the model
optimizer = tf.keras.optimizers.Adam()
```

```
model_concat.compile(optimizer = optimizer,loss =
loss_function,metrics = ['accuracy'])

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

# tensorboard callback
log_dir = '/content/drive/MyDrive/attention_assignment/logs_concat'
tensorboard_cb = TensorBoard(log_dir=log_dir)
# model checkpoint
checkpoint_filepath =
'/content/drive/MyDrive/attention_assignment/model_save/concat_model.h
5'
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor='val_loss',
    save_best_only=True)

model_concat.fit(x=train_dataloader,
          validation_data = test_dataloader,
          epochs = 15,
          steps_per_epoch = train_steps,
          validation_steps = valid_steps,
          callbacks = [tensorboard_cb]
          )
model_concat.summary()
```

```
Epoch 1/15
279/279 [==============================] - 314s 945ms/step - loss:
14.7101 - accuracy: 0.0498 - val_loss: 2.4552 - val_accuracy: 0.0500
Epoch 2/15
279/279 [==============================] - 267s 957ms/step - loss:
1.9018 - accuracy: 0.0610 - val_loss: 1.6974 - val_accuracy: 0.0652
Epoch 3/15
279/279 [==============================] - 250s 896ms/step - loss:
1.6206 - accuracy: 0.0734 - val_loss: 1.5500 - val_accuracy: 0.0812
Epoch 4/15
279/279 [==============================] - 251s 899ms/step - loss:
1.5181 - accuracy: 0.0857 - val_loss: 1.4882 - val_accuracy: 0.0895
Epoch 5/15
279/279 [==============================] - 251s 899ms/step - loss:
1.4646 - accuracy: 0.0947 - val_loss: 1.4365 - val_accuracy: 0.0987
Epoch 6/15
279/279 [==============================] - 251s 899ms/step - loss:
1.4035 - accuracy: 0.1087 - val_loss: 1.3754 - val_accuracy: 0.1125
Epoch 7/15
279/279 [==============================] - 251s 899ms/step - loss:
1.3518 - accuracy: 0.1156 - val_loss: 1.3347 - val_accuracy: 0.1182
Epoch 8/15
```

```
279/279 [==============================] - 251s 900ms/step - loss:
1.3115 - accuracy: 0.1211 - val_loss: 1.2986 - val_accuracy: 0.1230
Epoch 9/15
279/279 [==============================] - 251s 899ms/step - loss:
1.2766 - accuracy: 0.1258 - val_loss: 1.2661 - val_accuracy: 0.1277
Epoch 10/15
279/279 [==============================] - 251s 899ms/step - loss:
1.2441 - accuracy: 0.1301 - val_loss: 1.2400 - val_accuracy: 0.1308
Epoch 11/15
279/279 [==============================] - 251s 900ms/step - loss:
1.2125 - accuracy: 0.1346 - val_loss: 1.1987 - val_accuracy: 0.1367
Epoch 12/15
279/279 [==============================] - 251s 900ms/step - loss:
1.1782 - accuracy: 0.1390 - val_loss: 1.1710 - val_accuracy: 0.1403
Epoch 13/15
279/279 [==============================] - 268s 961ms/step - loss:
1.1372 - accuracy: 0.1446 - val_loss: 1.1291 - val_accuracy: 0.1457
Epoch 14/15
279/279 [==============================] - 252s 903ms/step - loss:
1.0970 - accuracy: 0.1498 - val_loss: 1.1005 - val_accuracy: 0.1495
Epoch 15/15
279/279 [==============================] - 252s 902ms/step - loss:
1.1331 - accuracy: 0.1467 - val_loss: 1.0753 - val_accuracy: 0.1521
Model: "encoder_decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder (Encoder)           multiple                  3049468

 decoder (Decoder)           multiple                  5975845

=================================================================
Total params: 9,025,313
Trainable params: 9,025,313
Non-trainable params: 0
_________________________________________________________________
```

```
%tensorboard --logdir
'/content/drive/MyDrive/attention_assignment/logs_concat'
```

```
Output hidden; open in https://colab.research.google.com to view.
```

```
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 onestepdecoder.
    D. till we reach max_length of decoder or till the model predicted
```

```
word <end>:
            predictions, input_states, attention_weights =
model.layers[1].onestepdecoder(input_to_decoder, encoder_output,
input_states)
            Save the attention weights
            And get the word using the tokenizer(word index) and then
store it in a string.
    E. Call plot_attention(#params)
    F. Return the predicted sentence
    '''

    # convert sentence into intger encoding
    encoder_in = tokenizer_ita.texts_to_sequences([input_sentence])

    # pad sequence
    encoder_in = pad_sequences(encoder_in,maxlen=20, dtype='int32',
padding='post')
    encoder_in = tf.convert_to_tensor(encoder_in)

    # initialize state
    initial_state = model_concat.layers[0].initialize_states(1)

    # define empty sentence
    result = ''

    # Initialize index of <start>:1
    dec_input = np.array([[1]])

    # define empty array for attention plot
    attention_plot = np.zeros((20,20))

    # get output and states from encoder
    enc_output, enc_state_h, enc_state_c = model_concat.layers[0]
(encoder_in,initial_state)
    decoder_h = enc_state_h
    decoder_c = enc_state_c

    for t in range(20):
        # call onestep decoder
        predictions, decoder_h, decoder_c,attention_weights,c_vector =
model_concat.layers[1].onestep_decoder(
            dec_input, enc_output, decoder_h,decoder_c)

        # storing the attention weights to plot later on
        attention_weights = tf.reshape(attention_weights, (-1, ))
        attention_plot[t] = attention_weights.numpy()

        # predict index of word
        predicted_id = tf.argmax(predictions[0]).numpy()
```

```
        result += tokenizer_eng.index_word[predicted_id] + ' '

        if tokenizer_eng.index_word[predicted_id] == '<end>':
            return result, input_sentence , attention_plot

        # the predicted ID is fed back into the model
        dec_input = tf.expand_dims([predicted_id], 0)

    return result, input_sentence, attention_plot
```

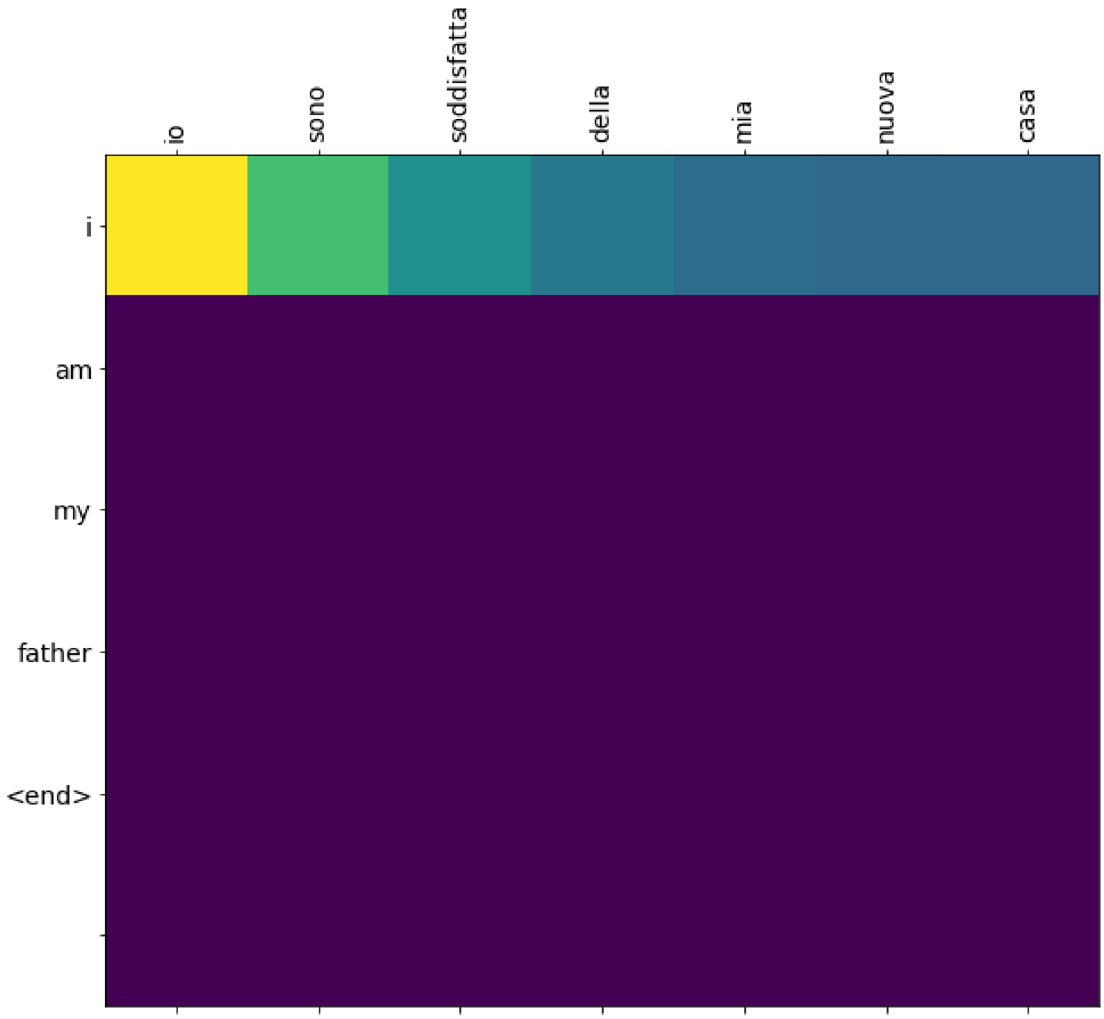
Example 1:

```
train.iloc[4]
```

```
italian            io sono soddisfatta della mia nuova casa
english_inp          <start> i am pleased with my new house
english_out            i am pleased with my new house <end>
Name: 246299, dtype: object
```

```
translate(train.iloc[4].italian)
```

```
Input: io sono soddisfatta della mia nuova casa
Predicted translation: i am my father <end>
```



Example: 2

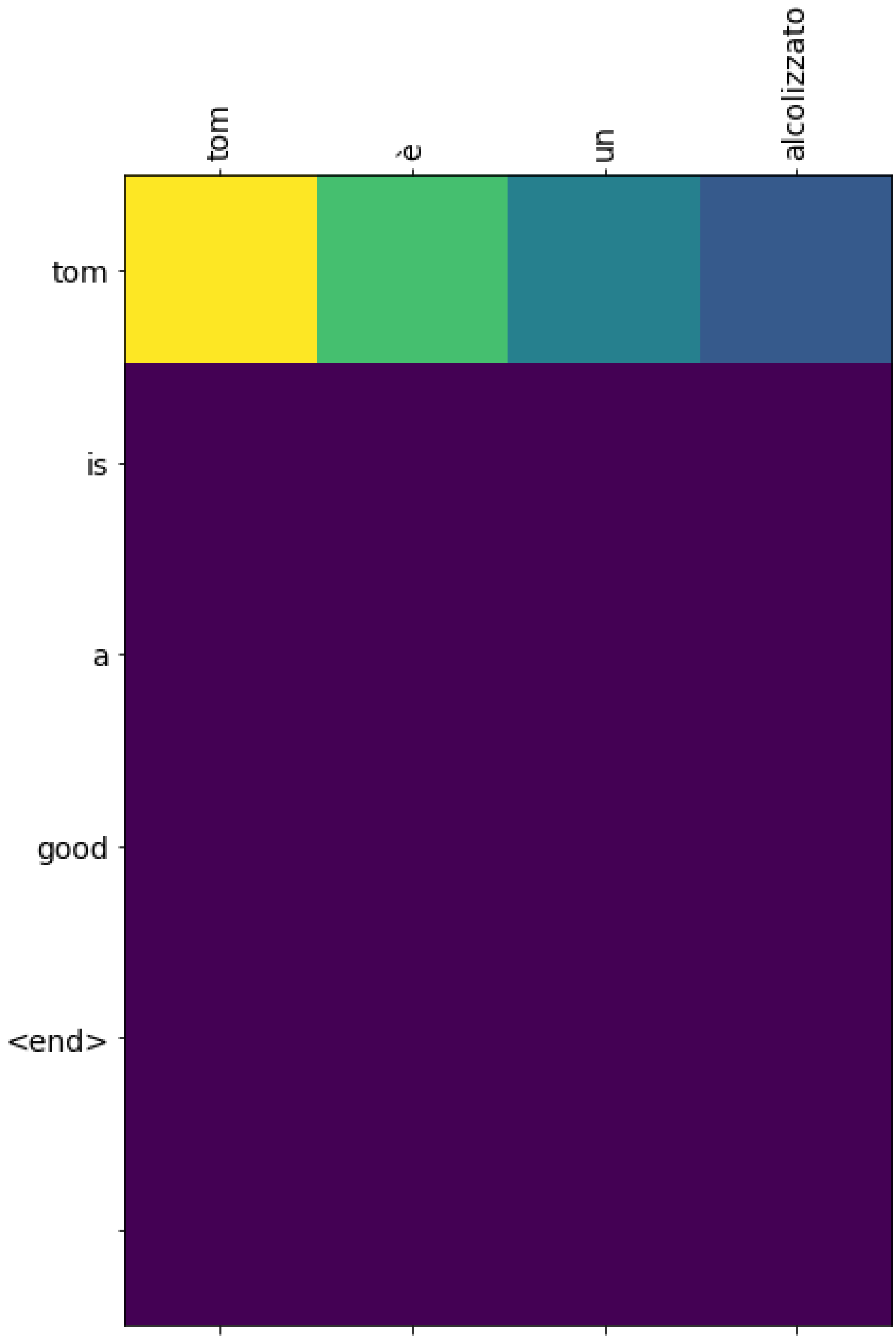
```
train.iloc[10]
```

```
italian                  tom è un alcolizzato
english_inp    <start> tom is an alcoholic
english_out      tom is an alcoholic <end>
Name: 82157, dtype: object
```

```
italian                  tom è un alcolizzato
english_inp    <start> tom is an alcoholic
english_out      tom is an alcoholic <end>
Name: 82157, dtype: object
```

```
translate(train.iloc[10].italian)
```

```
Input: tom è un alcolizzato
Predicted translation: tom is a good <end>
```


tom
è
un
alcolizzato
tom
is
a
good
<end>

Calculate Average BLEU score fot 100 random sentence (concat scoring)

```
np.random.seed(42)
BLEU_Scores_concat = []
for i in tqdm(range(1000)):
    index = np.random.randint(1,68677)
    # Original sentence
    original_sent = [validation['english_out'].iloc[index].split(),]
    # predicted sentence
    prediction,_,_ = predict(validation['italian'].iloc[index])
    prediction = prediction.split()
    # compute BLUE
    blue_score = bleu.sentence_bleu(original_sent, prediction)
    # append score
    BLEU_Scores_concat.append(blue_score)

print('Average BLUE score using concat scoring is ===>
',np.mean(BLEU_Scores_concat))
```

```
100%|██████████| 1000/1000 [02:15<00:00,  7.38it/s]

Average BLUE score using concat scoring is ===>  0.014918195637248258
```

```
# Write your observations on each of the scoring
```

Observations

###Dot Scoring

- In dot scoring i have ran the model for 15 epochs and we got the bleu score of 7.381186812496 41e-80 and in Dot very less number of words are attenting atention.Attention wise dot scoring is better than general scoring.

general scoring

- In general scoring first i have ran the model for 15 epochs and we got the bleu score of 5.95870057583763e-80 which is decreased when compared to dot scoring. but we can see that attention plots shows some attention compared to dot scoring and we can say general scoring takes more time for training because of difference in scoring function.

Concat scoring

- In concat scoring model i have trained model only for 15 epochs and got Highest BLEU score then rest of the scoring function and is 0.014918195637248258.
- Also training time is more than Dot scoring and almost same as general scoring
- In concat scoring attention plots significant nnumber of words are getting attention in sentence translation.