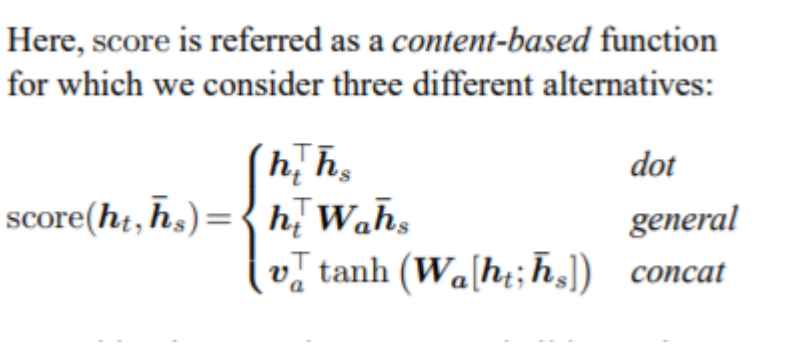
Task 2 (Dot Attention Model)

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** (https://drive.google.com/file/d/1z_bnc-3aubKawbR6q8wyl6Mh5ho2R1aZ/view?usp=sharing) 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.
4. It is mandatory to train the model with simple model.fit() only, Donot train the model with custom GradientTape()
5. Using attention weights, you can plot the attention plots, please plot those for 2-3 examples. You can check about those in this (https://www.tensorflow.org/tutorials/text/nmt_with_attention#translate)
6. 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.
7. Please implement the class **onestepdecoder** as mentioned in the assignment instructions.
8. You can use any tf.Keras highlevel API's to build and train the models. Check the reference notebook for better understanding.
9. Use BLEU score as metric to evaluate your model. You can use any loss function you need.
10. You have to use Tensorboard to plot the Graph, Scores and histograms of gradients.
11. Resources: a. Check the reference notebook b. Resource 1 (https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/) c. Resource 2 (https://www.tensorflow.org/tutorials/text/nmt_with_attention) d. Resource 3

(https://stackoverflow.com/questions/44238154/what-is-the-difference-between-luong-attention-and-bahdanau-attention#:~:text=Luong%20attention%20used%20top%20hidden,hidden%20state%20at%20tim

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

```
Mounted at /content/drive
```

Note:

- In Task 2 we have to train three models.
- I have made a seperate notebook for each model to avoid any confusion.
- This notebook contains the Dot Attention Model.

In [1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import re
```

Loading the Data

In [3]:
```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/ass-28/ita.txt', delimit
df
```

Out[3]:

	english	italian
0	Hi.	Ciao!
1	Run!	Corri!
2	Run!	Corra!
3	Run!	Correte!
4	Who?	Chi?
...	...	...
345239	If you want to sound like a native speaker, yo...	Se vuoi sembrare un madrelingua, devi essere d...
345240	If you want to sound like a native speaker, yo...	Se vuoi sembrare un madrelingua, devi essere d...
345241	If someone who doesn't know your background sa...	Se qualcuno che non conosce il tuo background ...
345242	Doubtless there exists in this world precisely...	Senza dubbio esiste in questo mondo proprio la...
345243	Doubtless there exists in this world precisely...	Senza dubbio esiste in questo mondo proprio la...

345244 rows × 2 columns

Preprocess Data

In [4]:
```
def decontracted(phrase):
    # specific
    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)
    return phrase
def preprocessor(text):
    text = text.lower()
    text = decontracted(text)
    text = re.sub("[^A-Za-z0-9 ]+",'',text)
    return text
def preprocessor_ita(text):
    text = text.lower()
    text = decontracted(text)
    text = re.sub("[^A-Za-z0-9 ]+",'',text)
    return text
```

In [5]:
```
# PREPROCESSED DATA INTO A DATAFRAME
df["english"] = df.english.apply(preprocessor)
df["italian"] = df.italian.apply(preprocessor_ita)
df
```

Out[5]:

	english	italian
0	hi	ciao
1	run	corri
2	run	corra
3	run	correte
4	who	chi
...	...	...
345239	if you want to sound like a native speaker you...	se vuoi sembrare un madrelingua devi essere di...
345240	if you want to sound like a native speaker you...	se vuoi sembrare un madrelingua devi essere di...
345241	if someone who does not know your background s...	se qualcuno che non conosce il tuo background ...
345242	doubtless there exists in this world precisely...	senza dubbio esiste in questo mondo proprio la...
345243	doubtless there exists in this world precisely...	senza dubbio esiste in questo mondo proprio la...

345244 rows × 2 columns

In [6]:
```
# REMOVING SENTENCES WITH LENGTH GREATER THAN 20
df["eng_len"] = df.english.apply(lambda x: len(x.split()))
df = df[df.eng_len<=20]
df["ita_len"] = df.italian.apply(lambda x: len(x.split()))
df  = df[df.ita_len<=20]
# ADDING <start> TO THE BEGINING OF ENGLISH SENTENCES
df["english_inp"]= "<start> "+ df.english
df["english_out"]= df.english+ " <end>"
df.drop(["english","eng_len","ita_len"],axis=1,inplace=True)
df
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopy
Warning:
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/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  This is separate from the ipykernel package so we can avoid doing imports unt
il
```

Out[6]:

	italian	english_inp	english_out
0	ciao	<start> hi	hi <end>
1	corri	<start> run	run <end>
2	corra	<start> run	run <end>
3	correte	<start> run	run <end>
4	chi	<start> who	who <end>
...	...	...	...
344915	charles moore cre il forth nel tentativo di au...	<start> charles moore created forth in an atte...	charles moore created forth in an attempt to i...
344998	se la tua azienda opera principalmente con lam...	<start> if your company primarily does busines...	if your company primarily does business with a...
344999	se la sua azienda opera principalmente con lam...	<start> if your company primarily does busines...	if your company primarily does business with a...
345000	se la vostra azienda opera principalmente con ...	<start> if your company primarily does busines...	if your company primarily does business with a...
345001	lintelligenza fondata nella capacit di ricono...	<start> intelligence is found in the capacity ...	intelligence is found in the capacity to recog...

344860 rows × 3 columns

Train Test Split

In [2]:
```
from sklearn.model_selection import train_test_split
```

In [8]:
```
train , validation = train_test_split(df,test_size = 0.2,random_state=33)
#ADDING <end> RO THE END OF FIRST ENGLISH SENTENCE IN "english_inp"
train.iloc[0]["english_inp"]  = train.iloc[0]["english_inp"] + " <end>"
```

In [8]:
```
```

Tokenization

In [3]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from  tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [10]:
```
# TOKENIZE ENGLISH SENTENCES
tk_eng = Tokenizer(filters = '!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n')
tk_eng.fit_on_texts(train.english_inp.values)
# TOKENIZING ITALIAN SENTENCES
tk_ita = Tokenizer()
tk_ita.fit_on_texts(train.italian)
```

Data Generator

In [4]:
```
from  tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [12]:
```
class Dataset :
    def __init__(self, data,tk_eng,tk_ita,max_len):
        self.encoder_inp = data["italian"].values
        self.decoder_inp = data["english_inp"].values
        self.decoder_out = data["english_out"].values
        self.tk_eng = tk_eng
        self.tk_ita = tk_ita
        self.max_len = max_len

    def __getitem__(self,i):
        # ITALIAN TO INTEGER SEQUENCES
        self.encoder_seq = self.tk_ita.texts_to_sequences([self.encoder_inp[i]])
        # ENGLISH TO INTEGER SEQUENCES
        self.decoder_inp_seq = self.tk_eng.texts_to_sequences([self.decoder_inp[i
        # ENGLISH TO INTEGER SEQUENCES
        self.decoder_out_seq = self.tk_eng.texts_to_sequences([self.decoder_out[i

        # PADDING THE ENCODER INPUT SEQUENCES
        self.encoder_seq = pad_sequences(self.encoder_seq,maxlen = self.max_len,p
        # PADDING THE DECODER INPUT SEQUENCES
        self.decoder_inp_seq = pad_sequences(self.decoder_inp_seq,maxlen = self.m
        # PADDING DECODER OUTPUT SEQUENCES
        self.decoder_out_seq = pad_sequences(self.decoder_out_seq,maxlen = self.m
        return self.encoder_seq ,  self.decoder_inp_seq,  self.decoder_out_seq

    def __len__(self):
        # RETURN THE LEN OF INPUT ENDODER
        return len(self.encoder_inp)
```

In [13]:
```
class Dataloader(tf.keras.utils.Sequence):
    def __init__(self,batch_size,dataset):
        # INTIALIZING THE REQUIRED VARIABLES
        self.dataset = dataset
        self.batch_size = batch_size
        self.totl_points = self.dataset.encoder_inp.shape[0]

    def __getitem__(self,i):
        # STATING THE START AND STOP VATIABLE CONTAINGING INDEX VALUES FOR EACH E
        start = i * self.batch_size
        stop = (i+1)*self.batch_size

        # PLACEHOLDERS FOR BATCHED DATA
        batch_ita =[]
        batch_eng_input = []
        batch_eng_out =[]

        for j in range(start,stop): # FOR EACH VALUE IN START TO STOP

            a,b,c = self.dataset[j] # DATASET RETURNS ITALIAN , ENGLIGH_INPUT, EN
            batch_ita.append(a[0]) # APPENDING ITALIAN TO batch_ita
            batch_eng_input.append(b[0]) # APPENGIND ENGLISH INPUT TO batch_eng_i
            batch_eng_out.append(c[0]) # APPENDING ENGLISH OUTPUT TO batch_eng_ou

        # Conveting list to array
        batch_ita = (np.array(batch_ita))
        batch_eng_input = np.array(batch_eng_input)
        batch_eng_out = np.array(batch_eng_out)

        return [batch_ita , batch_eng_input],batch_eng_out

    def __len__(self):
        # Returning the number of batches
        return int(self.totl_points/self.batch_size)
```

Encoder

In [7]:
```
from tensorflow.keras import layers
```

In [8]:
```
class Encoder(tf.keras.layers.Layer):
    '''
    Encoder model -- That takes a input sequence and returns encoder-outputs,encc
    '''

    def __init__(self, vocab_size,emb_dims, enc_units, input_length,batch_size):
        super().__init__()
        # INITIALIZING THE REQUIRED VARIABLES
        self.batch_size=batch_size # BATHCH SIZE
        self.enc_units = enc_units # ENCODER UNITS

        # EMBEDDING LAYER
        self.embedding= layers.Embedding(vocab_size ,emb_dims)
        # LSTM LAYER WITH RETURN SEQ AND RETURN STATES
        self.lstm = layers.LSTM(self.enc_units,return_state= True,return_sequence
    def call(self, enc_input , states):
        '''
          This function takes a sequence input and the initial states of the encc
          Pass the input_sequence input to the Embedding layer, Pass the embeddir
          returns -- encoder_output, last time step's hidden and cell state
        '''
        # FORMING THE EMBEDDED VECTOR
        emb = self.embedding(enc_input)
        # PASSING THE EMBEDDED VECTIO THROUGH LSTM LAYERS
        enc_output,state_h,state_c = self.lstm(emb,initial_state=states)
        #RETURNING THE OUTPUT OF LSTM LAYER
        return enc_output,state_h,state_c

    def initialize(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 st
      '''
        return tf.zeros(shape=(batch_size,self.enc_units)),tf.zeros(shape=(batch_
```

In [13]:
```
vocab_size=10
embedding_size=20
lstm_size=32
input_length=10
batch_size=16

enc = Encoder(vocab_size,embedding_size,lstm_size,input_length,batch_size)
a,b,c = enc(tf.random.uniform((batch_size,input_length)),enc.initialize(batch_siz
print("ENCODER OUTPUT SHAPE=",a.shape)
print("ENCODER STATE H SHAPE=",b.shape)
print("ENCODER STATE C SHAPE=",c.shape)
```

```
ENCODER OUTPUT SHAPE= (16, 10, 32)
ENCODER STATE H SHAPE= (16, 32)
ENCODER STATE C SHAPE= (16, 32)
```

Grander Function 1

In [10]:
```
def grader_check_encoder():

    '''
        vocab-size: Unique words of the input language,
        embedding_size: output embedding dimension for each word after embedding
        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],maxval=vocab
    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.
    return True
print(grader_check_encoder())
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-11b658172760> in <module>
     21     assert(encoder_output.shape==(batch_size,input_length,lstm_size) an
d state_h.shape==(batch_size,lstm_size) and state_c.shape==(batch_size,lstm_siz
e))
     22     return True
---> 23 print(grader_check_encoder())

<ipython-input-10-11b658172760> in grader_check_encoder()
     14     input_length=10
     15     batch_size=16
---> 16     encoder=Encoder(vocab_size,embedding_size,lstm_size,input_length)
     17     input_sequence=tf.random.uniform(shape=[batch_size,input_length],ma
xval=vocab_size,minval=0,dtype=tf.int32)
     18     initial_state=encoder.initialize_states(batch_size)

TypeError: __init__() missing 1 required positional argument: 'batch_size'
```

Attention for Dot Model

In [14]:
```
# THIS IS ATTNETION LAYER FOR DOT MODEL
class Attention(tf.keras.layers.Layer):
    '''THIS FUNCTION RETURNS THE CONTEXT VECTOR AND ATTENTION WEIGHTS (ALPHA VALU
    def __init__(self,units):
        super().__init__()
        # INITIALIZING THE NUMBER OF UNITS IN ATTENTION MODEL
        self.units =  units

    def call(self,enc_output,dec_state):
        # EXPANDING THE DIMENSION OF DECODER STATE  EG. FROM (16,32) TO (16,32,1)
        dec_state =  tf.expand_dims(dec_state,axis=-1)

        # MATRIX MULTIPLICATION OF ENCODER OUTPUT AND MODIFIED DECODER STATE
        # (16,32,1)*(16,13,32) = (16,13,1)
        score = tf.matmul(enc_output,dec_state)

        # APPLYING SOFTMAX TO THE AXIS 1
        # OUPUT SHAPE = (16,13,1)
        att_weights = tf.nn.softmax(score,axis=1)

        # CALCULATING THE CONTEXT VECTOR BY FIRST ELEMENTWISE MULTIPLICATION AND
        # (16,13,1)*(16,13,32)=(16,13,32)
        context_vec  = att_weights* enc_output
        # (16,13,32) SUM AND REDUCE THE DIMENSION AT AXIS 1 => (16,32)
        context_vec = tf.reduce_sum(context_vec,axis=1)

        # RETURNING THE CONTEXT VECTOR AND ATTENTION WEIGHTS
        return context_vec,att_weights
```

In [17]:
```
input_length=10
batch_size=16
att_units=32

att = Attention(att_units)
encoder_output =  tf.random.uniform((batch_size,input_length,att_units))
dec_state_h=tf.random.uniform(shape=[batch_size,att_units])
x1, x2 = att(encoder_output,dec_state_h)
print("CONTEXT VECTOR SHAPE=",x1.shape)
print("ATTENTION WEIGHTS SHAPE=",x2.shape)
```

```
CONTEXT VECTOR SHAPE= (16, 32)
ATTENTION WEIGHTS SHAPE= (16, 10, 1)
```

Grader Function 2

In [18]:
```
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_units])
    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.sha
    return True
print(grader_check_attention('dot'))
print(grader_check_attention('general'))
print(grader_check_attention('concat'))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-18-1b8722fdfe69> in <module>
     17     assert(context_vector.shape==(batch_size,att_units) and attention_w
eights.shape==(batch_size,input_length,1))
     18     return True
---> 19 print(grader_check_attention('dot'))
     20 print(grader_check_attention('general'))
     21 print(grader_check_attention('concat'))

<ipython-input-18-1b8722fdfe69> in grader_check_attention(scoring_fun)
     13     state_h=tf.random.uniform(shape=[batch_size,att_units])
     14     encoder_output=tf.random.uniform(shape=[batch_size,input_length,att
_units])
---> 15     attention=Attention(scoring_fun,att_units)
     16     context_vector,attention_weights=attention(state_h,encoder_output)
     17     assert(context_vector.shape==(batch_size,att_units) and attention_w
eights.shape==(batch_size,input_length,1))

TypeError: __init__() takes 2 positional arguments but 3 were given
```

One Step Decoder

In [25]:
```
class Onestepdecoder(tf.keras.Model):
    '''THIS MODEL OUTPUTS THE RESULT OF DECODER FOR ONE TIME SETP GIVEN THE INPUT

    def __init__(self, vocab_size,emb_dims, dec_units, input_len,att_units,batch_
        super().__init__()
        # INTITALIZING THE REQUIRED VARIABLES
        # EMBEDDING LAYERS
        self.emb = layers.Embedding(vocab_size,emb_dims,input_length= input_len)
        # ATTENTION LAYER
        self.att = Attention(att_units)
        # LSTM LAYER
        self.lstm = layers.LSTM(dec_units,return_sequences=True,return_state=True
        # DENSE LAYER
        self.dense = layers.Dense(vocab_size)

    def call(self, encoder_output , input , state_h):
        # FORMING THE EMBEDDED VECTOR FOR THE WORD
        # (32,1)=>(32,1,12)
        emb = self.emb(input)

        # GETTING THE CONTEXT VECTOR AND ATTENTION WEIGHTS BASED ON THE ENCODER O
        context_vec,alphas = self.att(encoder_output,state_h)

        # CONCATINATING THE CONTEXT VECTOR(BY EXPANDING DIMENSION) AND ENBEDDED V
        # (32,1,16)|(32,1,12)=>(32,1,28)
        dec_input =  tf.concat([tf.expand_dims(context_vec,1),emb],axis=-1)

        # PASSING THE CONTEXT VECTOR THROUGH LSTM UNIT
        dec_output,dec_state_h,state_c = self.lstm(dec_input)

        # PASSING THE DECODER OUTPUT THROUGH DENSE LAYER WITH UNITS EQUAL TO VOCA
        fc = self.dense(dec_output)

        # RETURNING THE OUTPUT
        return fc , dec_state_h , alphas
```

In [26]:
```
tar_vocab_size=13
embedding_dim=12
input_length=10
dec_units=16
att_units=16
batch_size=32

onestepdecoder=Onestepdecoder(tar_vocab_size, embedding_dim, input_length, dec_ur
input_to_decoder=tf.random.uniform(shape=(batch_size,1),maxval=10,minval=0,dtype=
encoder_output=tf.random.uniform(shape=[batch_size,input_length,dec_units])
state_h=tf.random.uniform(shape=[batch_size,dec_units])

output,state_h,attention_weights=onestepdecoder(encoder_output,input_to_decoder,s

print("one_step_dec output shape",output.shape)
print("state_h",state_h.shape)
print("attention_weight",attention_weights.shape)
```

```
one_step_dec output shape (32, 1, 13)
state_h (32, 10)
attention_weight (32, 10, 1)
```

Grader Function 3

In [38]:
```
def grader_onestepdecoder(score_fun):

    '''
        tar_vocab_size: Unique words of the target language,
        embedding_dim: output embedding dimension for each word after embedding l
        dec_units: Number of lstm units in decoder,
        att_units: Used in matrix multiplications for scoring functions in attent
        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,
    input_to_decoder=tf.random.uniform(shape=(batch_size,1),maxval=10,minval=0,dt
    encoder_output=tf.random.uniform(shape=[batch_size,input_length,dec_units])
    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_
    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'))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-38-bc4618541f5d> in <module>
     31     return True
     32
---> 33 print(grader_onestepdecoder('dot'))
     34 print(grader_onestepdecoder('general'))
     35 print(grader_onestepdecoder('concat'))

<ipython-input-38-bc4618541f5d> in grader_onestepdecoder(score_fun)
     18     att_units=16
     19     batch_size=32
---> 20     onestepdecoder=One_Step_Decoder(tar_vocab_size, embedding_dim, in
put_length, dec_units ,score_fun ,att_units)
     21     input_to_decoder=tf.random.uniform(shape=(batch_size,1),maxval=10
,minval=0,dtype=tf.int32)
     22     encoder_output=tf.random.uniform(shape=[batch_size,input_length,d
ec_units])
```

```
NameError: name 'One_Step_Decoder' is not defined
```

Decoder

In [28]:
```
class Decoder(tf.keras.Model):
    '''THIS MODEL PERFORMS THE WHOLE DECODER OPERATION FOR THE COMPLETE SENTENCE'
    def __init__(self, vocab_size,emb_dims, dec_units, input_len,att_units,batch_
        super().__init__()
        # INITIALIZING THE VARIABLES
        # LENGTH OF INPUT SENTENCE
        self.input_len = input_len
        # ONE STEP DECODER
        self.onestepdecoder = Onestepdecoder(vocab_size,emb_dims, dec_units, inpu

    def call(self,dec_input,enc_output,state_h):
        # THIS VATIABLE STORES THE VALUE OF STATE_H FOR THE PREVIOUS STATE
        current_state_h = state_h
        # THIS STORES THE DECODER OUTPUT FOR EACH TIME STEP
        pred = []
        # THIS STORED THE ALPHA VALUES
        alpha_values = []
        # FOR EACH WORD IN THE INPUT SENTENCE
        for i in range(self.input_len):

            # CURRENT WORD TO INPUT TO ONE STEP DECODER
            current_vec = dec_input[:,i]
            # EXPANDING THE DIMENSION FOR THE WORD
            current_vec = tf.expand_dims(current_vec,axis=-1)
            # PERFORMING THE ONE STEP DECODER OPERATION
            dec_output,dec_state_h,alphas = self.onestepdecoder(enc_output ,curre
            #UPDATING THE CURRENT STATE_H
            current_state_h = dec_state_h
            #APPENDING THE DECODER OUTPUT TO "pred" LIST
            pred.append(dec_output)
            # APPENDING THE ALPHA VALUES
            alpha_values.append(alphas)

        # CONCATINATING ALL THE VALUES IN THE LIST
        output = tf.concat(pred,axis=1)
        # CONCATINATING ALL THE ALPHA VALUES IN THE LIST
        alpha_values = tf.concat(alpha_values,axis = -1)
        # RETURNING THE OUTPUT
        return output , alpha_values
```

In [36]:
```
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),maxval=10,minv
encoder_output=tf.random.uniform(shape=[batch_size,input_length,dec_units])
state_h=tf.random.uniform(shape=[batch_size,dec_units])

decoder =Decoder(out_vocab_size, embedding_dim,dec_units , input_length, att_unit
output,alpha_values = decoder(target_sentences,encoder_output, state_h)
print("Decoder Output=",output.shape)
print("Alpha Values=",alpha_values.shape)
```

```
Decoder Output= (32, 11, 13)
Alpha Values= (32, 11, 11)
```

Grader Function 4

In [39]:
```
def grader_decoder(score_fun):

    '''
        out_vocab_size: Unique words of the target language,
        embedding_dim: output embedding dimension for each word after embedding l
        dec_units: Number of lstm units in decoder,
        att_units: Used in matrix multiplications for scoring functions in attent
        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),maxval=10,
    encoder_output=tf.random.uniform(shape=[batch_size,input_length,dec_units])
    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
    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'))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-ca3f12bb002e> in <module>
     28     assert(output.shape==(batch_size,input_length,out_vocab_size))
     29     return True
---> 30 print(grader_decoder('dot'))
     31 print(grader_decoder('general'))
     32 print(grader_decoder('concat'))

<ipython-input-39-ca3f12bb002e> in grader_decoder(score_fun)
     25
     26     decoder=Decoder(out_vocab_size, embedding_dim, input_length, dec_
units ,score_fun ,att_units)
---> 27     output=decoder(target_sentences,encoder_output, state_h, state_c)
     28     assert(output.shape==(batch_size,input_length,out_vocab_size))
     29     return True

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow_core\python\keras\engin
e\base_layer.py in __call__(self, inputs, *args, **kwargs)
    820           with base_layer_utils.autocast_context_manager(
    821               self._compute_dtype):
--> 822             outputs = self.call(cast_inputs, *args, **kwargs)
    823           self._handle_activity_regularization(inputs, outputs)
    824           self._set_mask_metadata(inputs, outputs, input_masks)
```

```
TypeError: call() takes 4 positional arguments but 5 were given
```

Encoder Decoder

In [33]:
```
class encoder_decoder(tf.keras.Model):
    '''THIS MODEL COMBINES ALL THE LAYERS AND FORM IN ENCODER DECODER MODEL WITH
    def __init__(self,enc_vocab_size,enc_emb_dim,enc_units,enc_input_length,
                 dec_vocab_size,dec_emb_dim,dec_units,dec_input_length ,att_units, ba
        # INITAILIZING ALL VARIABLES
        super().__init__()
        # BATCH SIZE
        self.batch_size = batch_size
        # INITIALIZING ENCODER LAYER
        self.encoder = Encoder(enc_vocab_size, enc_emb_dim,enc_units, enc_input_l
        # INITALIZING DECODER LAYER
        self.decoder = Decoder(dec_vocab_size ,dec_emb_dim,dec_units,dec_input_le

    def call(self,data):
        # THE INPUT OF DATALOADER IS IN A LIST FORM FOR EACH BATCH IT GIVER TWO I
        # INPUT1 IS FOR ENCODER
        # INPUT2 IS FOR DECODER
        inp1 , inp2 = data
        # PASSING THE INPUT1 TO ENCODER LAYER
        enc_output, enc_state_h, enc_state_c = self.encoder(inp1,self.encoder.ini
        # PASSING INPUT2 TO THE DECODER LAYER
        dec_output , alphas = self.decoder(inp2 , enc_output,enc_state_h)
        # THE OUTPUT OF MODEL IS ONLY DECODER OUTPUT THE ALPHA VALUES ARE IGNORED
        return dec_output
```

In [37]:
```
enc_vocab_size=13
enc_emb_dim=12
enc_units=16
enc_input_length=10
dec_vocab_size=13
dec_emb_dim=12
dec_units=16
dec_input_length=10
batch_size=32
att_units=16

e_d = encoder_decoder(enc_vocab_size,enc_emb_dim,enc_units,enc_input_length,
                      dec_vocab_size,dec_emb_dim,dec_units,dec_input_length, att_

data1 = tf.random.uniform(shape=[batch_size , enc_input_length])
data2 = tf.random.uniform(shape=[batch_size , dec_input_length])
output = e_d([data1,data2])
print("MODEL OUPUT",output.shape)
```

```
MODEL OUPUT (32, 10, 13)
```

In [25]:
```
# FORMING THE BATCHED DATASET FOR TRAIN DATA
train_dataset = Dataset(train,tk_eng,tk_ita,20)
train_dataloader = Dataloader( batch_size = 1000 , dataset=train_dataset)
#FORMING THE BATCHED DATASET FOR TEST DATA
val_dataset = Dataset(validation,tk_eng,tk_ita,20)
val_dataloader = Dataloader(batch_size=1000,dataset=val_dataset)
```

In [26]:
```
# INITAILZING THE MODEL
model = encoder_decoder(enc_vocab_size=len(tk_ita.word_index)+1,
                        enc_emb_dim = 50,
                        enc_units=512,enc_input_length=20,
                        dec_vocab_size =len(tk_eng.word_index)+1,
                        dec_emb_dim =50,
                        dec_units=512,
                        dec_input_length = 20,

                        att_units=512,
                        batch_size=1000)
```

In [29]:
```
# TRAIN STEPS
train_steps = train_dataloader.__len__()
# VALIDATION STEPS
val_steps  = val_dataloader.__len__()
# CALLBACKS FOR SAVING THE MODEL
callback = [tf.keras.callbacks.ModelCheckpoint("/content/drive/MyDrive/Colab Note
]
# COMPILIG THE MODEL WITH ADAM OPTIMIZER AND LOSS AS SPARSE CROSS ENROPY WITH LOG
model.compile(optimizer="adam",loss=tf.keras.losses.SparseCategoricalCrossentropy
# FITTING THE MODEL
model.fit(train_dataloader, steps_per_epoch=train_steps,epochs= 50,validation_dat
```

```
Epoch 1/50
275/275 [==============================] - 345s 1s/step - loss: 2.6876 - val_
loss: 2.2299

Epoch 00001: val_loss improved from inf to 2.22990, saving model to /content/
drive/MyDrive/Colab Notebooks/ass-28/test_save2/best.h5
Epoch 2/50
275/275 [==============================] - 306s 1s/step - loss: 1.9486 - val_
loss: 1.7672

Epoch 00002: val_loss improved from 2.22990 to 1.76725, saving model to /cont
ent/drive/MyDrive/Colab Notebooks/ass-28/test_save2/best.h5
Epoch 3/50
275/275 [==============================] - 306s 1s/step - loss: 1.6852 - val_
loss: 1.5979

Epoch 00003: val_loss improved from 1.76725 to 1.59789, saving model to /cont
ent/drive/MyDrive/Colab Notebooks/ass-28/test_save2/best.h5
Epoch 4/50
```

In [29]:
```
model.load_weights("/content/drive/MyDrive/Colab Notebooks/ass-28/test_save2/best
```

Prediction

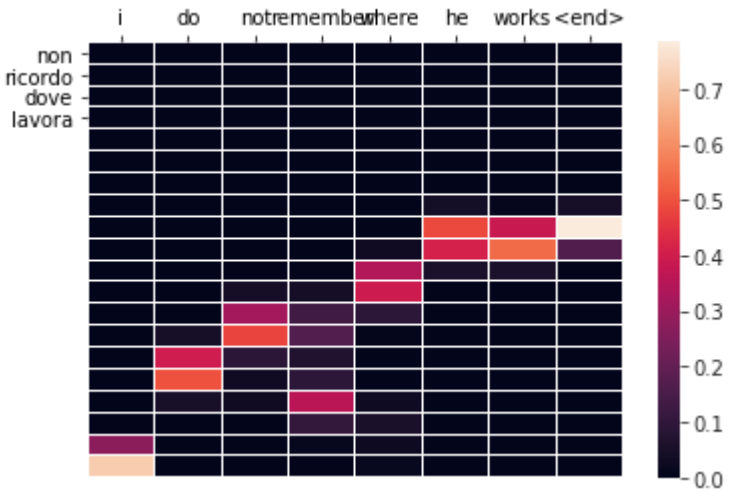
In [30]:
```
def predict(ita_text,model):
    '''THIS FUNCTION IS USED IN INFERENCE TIME WHICH GIVEN ANY SENTENCE IN ITALIA
    # FORMING TOKENIZED SEQUENCES FOR INPUT SENTENCE
    seq = tk_ita.texts_to_sequences([ita_text])

    # PADDING THE SEQUENCES
    seq = pad_sequences(seq,maxlen = 20 , padding="post")

    # INITIALIZING THE STATES FOR INPUTING TO ENCODER
    state = model.layers[0].initialize(1)

    # GETTING THE ENCODED OUTPUT
    enc_output,state_h,state_c= model.layers[0](seq,state)


    # VARIABLE TO STORE PREDICTED SENTENCE
    pred = []

    # THIS VARIABLE STORES THE STATE TO BE INPUTED TO ONE STEP ENCODER
    input_state_h = state_h
    input_state_c = state_c

    # THIS VARIABLE STORES THE VECTOR TO VE INPUTED TO ONE STEP ENCODER
    current_vec = tf.ones((1,1))
    # THIS VARIABLE WILL STORE ALL THE ALPHA VALUES OUTPUTS
    alpha_values = []

    for i in range(20):
        # PASSING THE REQUIRED VARIABLE TO ONE STEP ENCODER LAYER
        fc , dec_state_h , alphas = model.layers[1].layers[0](enc_output , currer
        #APPENDING THE ALPHA VALUES TO THE LIST "alpha_values"
        alpha_values.append(alphas)

        # UPDATING THE CURRENT VECTOR
        current_vec = np.argmax(fc , axis = -1)
        # UPDATING THE INPUT STATE
        input_state_h = dec_state_h
        # GETTING THE ACTUAL WORDS FRO THE TOKENIZED INDEXES
        pred.append(tk_eng.index_word[current_vec[0][0]])
        # IF THE WORD "<end>" COMES THE LOOP WILL BREAK
        if tk_eng.index_word[current_vec[0][0]]=="<end>":
            break
    # JOINING THE PREDICTED WORDS
    pred_sent = " ".join(pred)

    # CONCATINATING ALL THE ALPHA VALUES
    alpha_values = tf.squeeze(tf.concat(alpha_values,axis=-1),axis=0)

    # RETURNING THE PREDICTED SENTENCE AND ALPHA VALUES
    return  pred_sent , alpha_values
```

In [31]:
```
import seaborn as sns
import matplotlib.pyplot as plt


def plot( input_sent , output_sent , alpha ) :
    '''THIS FUNCTION PLOTS THE ALPHA VALUES IN FORM OF HEAT MAPS'''

    input_words = input_sent.split() # SPLITTING THE INPUT SENTENCE
    output_words = output_sent.split() # SPLITTING THE OUTPUT SENTENCE

    fig, ax = plt.subplots()
    sns.set_style("darkgrid")
    # HEAT MAP WITH ALPHA VALURS
    # X LABELS ARE THE OUTPUT WORDS
    # T LABELS ARE THE INPUT WORDS
    sns.heatmap(alpha, xticklabels= output_words , yticklabels=input_words,linewi
    # PLACING THE TICKS ON  THE TOP
    ax.xaxis.tick_top( )
    plt.show()
```

Predicting Some Random Results

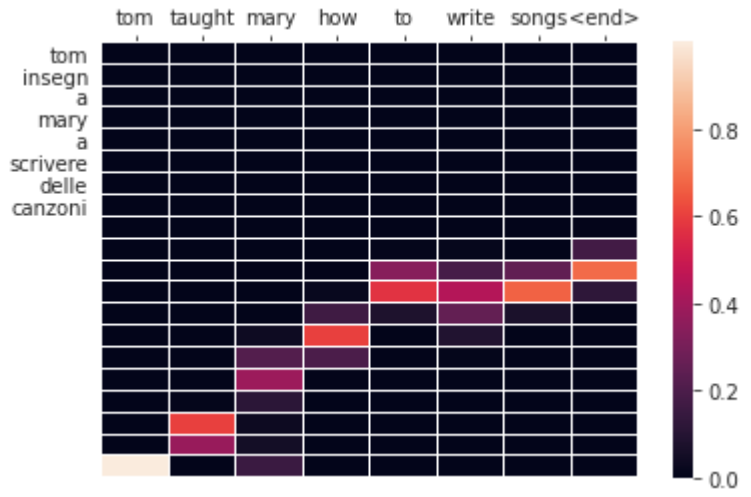
In [32]:
```
random = np.random.randint(0,2000,1)[0]
print("Input=>",validation.italian.values[random])
print("Predicted=>", predict( validation.italian.values[random] , model)[0])
print("Actual=>",validation.english_out.values[random])
print("="*50)
plot( validation.italian.values[random] , predict( validation.italian.values[ranc
```

```
Input=> non ricordo dove lavora
Predicted=> i do not remember where he works <end>
Actual=> i do not remember where you work <end>
==================================================
```



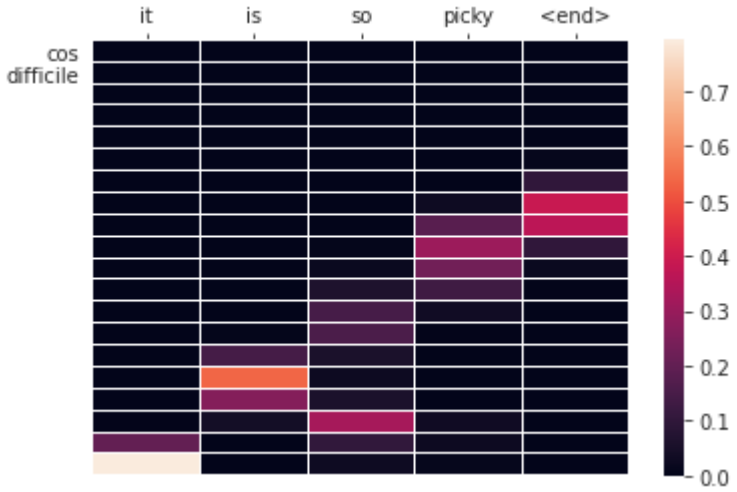
In [33]:
```
random = np.random.randint(0,2000,1)[0]
print("Input=>",validation.italian.values[random])
print("Predicted=>", predict( validation.italian.values[random] , model)[0])
print("Actual=>",validation.english_out.values[random])
print("="*50)
plot( validation.italian.values[random] , predict( validation.italian.values[ranc
```

```
Input=> tom insegn a mary a scrivere delle canzoni
Predicted=> tom taught mary how to write songs <end>
Actual=> tom taught mary how to write songs <end>
==================================================
```



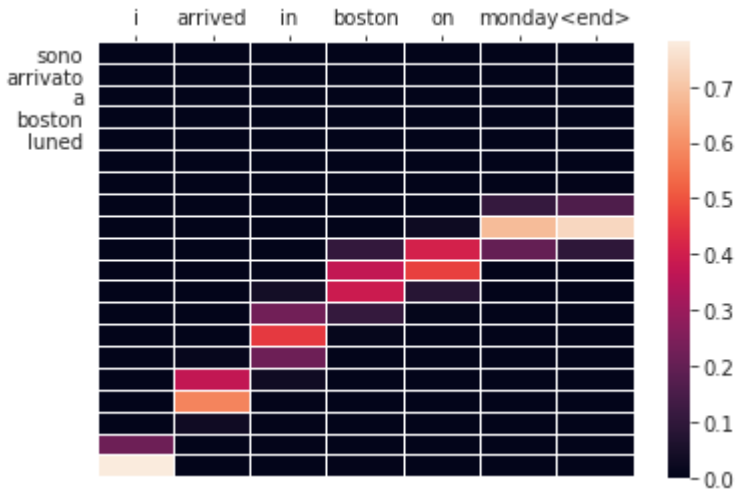
In [34]:
```
random = np.random.randint(0,2000,1)[0]
print("Input=>",validation.italian.values[random])
print("Predicted=>", predict( validation.italian.values[random] , model)[0])
print("Actual=>",validation.english_out.values[random])
print("="*50)
plot( validation.italian.values[random] , predict( validation.italian.values[ranc
```

```
Input=>  cos difficile
Predicted=> it is so picky <end>
Actual=> is that so difficult <end>
==================================================
```



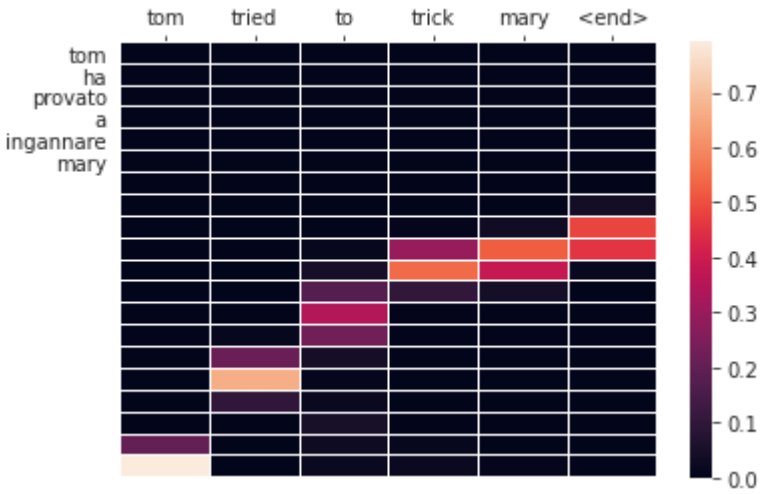
In [43]:
```
random = np.random.randint(0,2000,1)[0]
print("Input=>",validation.italian.values[random])
print("Predicted=>", predict( validation.italian.values[random] , model)[0])
print("Actual=>",validation.english_out.values[random])
print("="*50)
plot( validation.italian.values[random] , predict( validation.italian.values[ranc
```

```
Input=> sono arrivato a boston luned
Predicted=> i arrived in boston on monday <end>
Actual=> i arrived in boston on monday <end>
==================================================
```



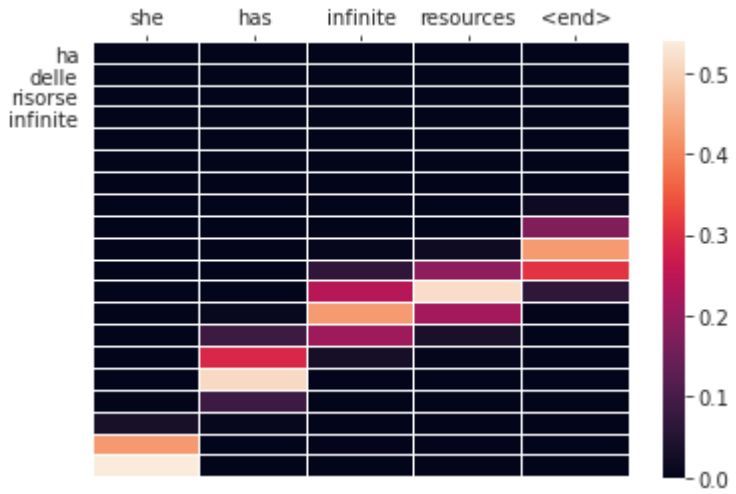
In [42]:
```
random = np.random.randint(0,2000,1)[0]
print("Input=>",validation.italian.values[random])
print("Predicted=>", predict( validation.italian.values[random] , model)[0])
print("Actual=>",validation.english_out.values[random])
print("="*50)
plot( validation.italian.values[random] , predict( validation.italian.values[ranc
```

```
Input=> tom ha provato a ingannare mary
Predicted=> tom tried to trick mary <end>
Actual=> tom tried to deceive mary <end>
==================================================
```



In [41]:
```
random = np.random.randint(0,2000,1)[0]
print("Input=>",validation.italian.values[random])
print("Predicted=>", predict( validation.italian.values[random] , model)[0])
print("Actual=>",validation.english_out.values[random])
print("="*50)
plot( validation.italian.values[random] , predict( validation.italian.values[ranc
```

```
Input=> ha delle risorse infinite
Predicted=> she has infinite resources <end>
Actual=> he has infinite resources <end>
==================================================
```



BLEU Score for Dot Model

In [39]:
```
import nltk.translate.bleu_score as bleu
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')
```

In [40]:
```
BLEU = []

test_data = validation.loc[np.random.choice(validation.index,size = 1000)][["ital
for ind,i in tqdm(test_data.iterrows()):
    pred = predict(i.italian , model)[0]
    act = i.english_out
    b =bleu.sentence_bleu(act,pred)
    BLEU.append(b)

print("BLEU DOT Score = ",np.mean(BLEU))
```

```
1000it [01:51,  8.96it/s]

BLEU DOT Score =  0.8377369268279129
```

In [ ]:
```

```

In [ ]: