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.

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.
3. 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.)
4. 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**

    Here, score is referred as a *content-based* function for which we consider three different alternatives:

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W}_a \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_a[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

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

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.

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import nltk
import string
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve,auc

import re
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from tqdm import tqdm

from collections import Counter
from tensorflow.keras.layers import Input,Embedding,LSTM,Dense
from tensorflow.keras.models import Model
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Concatenate
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Reshape

from scipy import sparse
import tensorflow as tf
from numpy import asarray
from numpy import zeros
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.callbacks import Callback,EarlyStopping
import datetime,os
from scipy.sparse import hstack
from tensorflow.keras.layers import TimeDistributed
from tensorflow.keras.layers import Bidirectional
from tensorflow.keras.layers import MaxPooling1D
from sklearn.metrics import roc_auc_score
from tensorflow.keras import optimizers
from tensorflow.compat.v1.keras.layers import CuDNNLSTM
```

Load the data

In [2]:

```
!wget --header="Host: www.manythings.org" --header="User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64
) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.132 Safari/537.36" --header="Accept: text/htm
l,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchan
ge;v=b3" --header="Accept-Language: en-GB,en-US;q=0.9,en;q=0.8" --header="Referer: https://colab.resear
ch.google.com/" --header="Cookie: __cfduid=d458f2fe234659a2d899b89fedfc54e8f1594543845; __utmz=3028652.
1594543810.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none); __utma=3028652.2060216133.1594543810.1594
627888.1594630131.3" --header="Connection: keep-alive" "http://www.manythings.org/anki/ita-eng.zip" -c
-O 'ita-eng.zip'
```

```
--2020-07-18 07:21:32--  http://www.manythings.org/anki/ita-eng.zip
Resolving www.manythings.org (www.manythings.org)... 104.24.108.196, 172.67.173.198, 104.24.109.196, ..
.
Connecting to www.manythings.org (www.manythings.org)|104.24.108.196|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7441562 (7.1M) [application/zip]
Saving to: 'ita-eng.zip'

ita-eng.zip         100%[===================>]   7.10M  4.11MB/s    in 1.7s

2020-07-18 07:21:34 (4.11 MB/s) - 'ita-eng.zip' saved [7441562/7441562]
```

In [3]:

```
!unzip 'ita-eng.zip'
```

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

Preprocess data

In [4]:

```
data = pd.read_csv('ita.txt', header=None, sep='\t')
data.columns = ['english','italy','attribute']
len(data)
```

Out[4]:

```
340432
```

In [5]:

```
data.columns
```

Out[5]:

```
Index(['english', 'italy', 'attribute'], dtype='object')
```

In [6]:

```
data.head(2)
```

Out[6]:

	english	italy	attribute
0	Hi.	Ciao!	CC-BY 2.0 (France) Attribution: tatoeba.org #5...
1	Run!	Corri!	CC-BY 2.0 (France) Attribution: tatoeba.org #9...

Processing data to decontracted form

In [7]:

```
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
```

Pre-Processing data using several techniques

In [8]:

```
#https://machinelearningmastery.com/prepare-french-english-dataset-machine-translation/

def clean_lines(lines):

    line = decontracted(lines)
    # prepare regex for char filtering
    re_print = re.compile('[^%s]'%re.escape(string.printable))
   # prepare translation table for removing punctuation
    table = str.maketrans('','',string.punctuation)
   # normalize unicode characters
    line = unicodedata.normalize('NFD',line).encode('ascii','ignore')
    line = line.decode('UTF-8')
    # tokenize on white space
    line = line.split()
  # convert to lower case
    line = [word.lower() for word in line]
  # remove punctuation from each token
    line = [word.translate(table) for word in line]
  # remove non-printable chars form each token
    line = [re_print.sub('',w) for w in line]
  # remove tokens with numbers in them
    line = [word for word in line if word.isalpha()]
  #Adding <start> and <end> tag to the preprocessed sentence.
    str1='<start>'
    for elem in line:
      str1+=' '+str(elem)
    str1+=' '+'<end>'

    return str1
```

Processing English and Italian sentences and storing them as seperate lists

In [9]:

```
import unicodedata
# load English data
english_data=[]
for i in range(len(data)):
    sentences = clean_lines(data['english'][i])
    english_data.append(sentences)

#load Italian data
italian_data=[]
for i in range(len(data)):
    sentences = clean_lines(data['italy'][i])
    italian_data.append(sentences)
```

In [10]:

```
print(english_data[-1])
print(data['english'][340431])
print(italian_data[-1])
print(data['italy'][340431])
```

```
<start> doubtless there exists in this world precisely the right woman for any given man to marry and v
ice versa but when you consider that a human being has the opportunity of being acquainted with only a
few hundred people and out of the few hundred that there are but a dozen or less whom he knows intimate
ly and out of the dozen one or two friends at most it will easily be seen when we remember the number o
```

```
ly and out of the dozen one or two friends at most it will easily be seen when we remember the number o
f millions who inhabit this world that probably since the earth was created the right man has never yet
met the right woman <end>
Doubtless there exists in this world precisely the right woman for any given man to marry and vice vers
a; but when you consider that a human being has the opportunity of being acquainted with only a few hun
dred people, and out of the few hundred that there are but a dozen or less whom he knows intimately, an
d out of the dozen, one or two friends at most, it will easily be seen, when we remember the number of
millions who inhabit this world, that probably, since the earth was created, the right man has never ye
t met the right woman.
<start> senza dubbio esiste in questo mondo proprio la donna giusta per ogni uomo da sposare e vicevers
a ma se si considera che un essere umano ha lopportunita di conoscere solo poche centinaia di persone e
fra le poche centinaia che ce ne sono solo una dozzina o meno che conosce intimamente e fra la dozzina
uno o due amici al massimo si vedra facilmente quando ricorderemo il numero di milioni che abitano ques
to mondo che probabilmente da quando e stata creata la terra luomo giusto non ha mai incontrato la donn
a giusta <end>
Senza dubbio esiste in questo mondo proprio la donna giusta per ogni uomo da sposare e viceversa; ma se
si considera che un essere umano ha l'opportunità di conoscere solo poche centinaia di persone, e fra l
e poche centinaia che ce ne sono solo una dozzina o meno che conosce intimamente e fra la dozzina, uno
o due amici al massimo, si vedrà facilmente, quando ricorderemo il numero di milioni che abitano questo
mondo, che probabilmente, da quando è stata creata la terra, l'uomo giusto non ha mai incontrato la don
na giusta.
```

Preparation of Data

Tokenize function

In [11]:

```
def tokenize(lang):
  lang_tokenizer = tf.keras.preprocessing.text.Tokenizer(
      filters='')
  lang_tokenizer.fit_on_texts(lang)

  tensor = lang_tokenizer.texts_to_sequences(lang)

  tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor,
                                                         padding='post')

  return tensor, lang_tokenizer
```

Tokenzing data into tensors and splitting into train and test

In [12]:

```
#Tokenizing data into tensor format
input_tensor, inp_lang = tokenize(italian_data[:100000])
target_tensor, targ_lang = tokenize(english_data[:100000])

max_length_targ, max_length_inp = target_tensor.shape[1], input_tensor.shape[1]

# Creating training and validation sets using an 80-20 split
input_tensor_train, input_tensor_val, target_tensor_train, target_tensor_val = train_test_split(input_t
ensor, target_tensor, test_size=0.2)

# Show length
print(len(input_tensor_train), len(target_tensor_train), len(input_tensor_val), len(target_tensor_val))
```

```
80000 80000 20000 20000
```

In [13]:

```
def convert(lang, tensor):
  for t in tensor:
    if t!=0:
      print ("%d ----> %s" % (t, lang.index_word[t]))

print ("Input Language; index to word mapping")
convert(inp_lang, input_tensor_train[0])
```

```
print ()
print ("Target Language; index to word mapping")
convert(targ_lang, target_tensor_train[0])
```

```
Input Language; index to word mapping
1 ----> <start>
1192 ----> tornero
2 ----> <end>

Target Language; index to word mapping
1 ----> <start>
3 ----> i
17 ----> will
28 ----> be
89 ----> back
2 ----> <end>
```

In [14]:

```
BATCH_SIZE = 128
embedding_dim = 256
units = 1024
```

Data Preparation

In [15]:

```
BUFFER_SIZE = len(input_tensor_train)

steps_per_epoch = len(input_tensor_train)//BATCH_SIZE

vocab_inp_size = len(inp_lang.word_index)+1
vocab_tar_size = len(targ_lang.word_index)+1

#getting slices of data in form of an array
dataset = tf.data.Dataset.from_tensor_slices((input_tensor_train, target_tensor_train)).shuffle(BUFFER_
SIZE)
#getting data batchwise
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)
```

In [16]:

```
example_input_batch, example_target_batch = next(iter(dataset))
example_input_batch.shape, example_target_batch.shape
```

Out[16]:

```
(TensorShape([128, 19]), TensorShape([128, 9]))
```

Implement custom encoder decoder and attention layers

Encoder

In [17]:

```
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.embedding_size = embedding_size
        self.lstm_size = lstm_size
        self.input_length = input_length
        #self.enc_units= 64
```

```
        #Initialize Embedding layer
        self.embedding = Embedding(self.inp_vocab_size, self.embedding_size)
        #Intialize Encoder LSTM layer
        self.lstm = CuDNNLSTM(self.lstm_size, return_state=True, return_sequences=True,
                        recurrent_initializer = 'glorot_uniform',time_major=False)


    def call(self,input_sequence,state_h,state_c):
      '''
          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 encod
er_lstm
          returns -- All encoder_outputs, last time steps hidden and cell state
      '''
      #Implementing Embedding layer
      input_embedd = self.embedding(input_sequence)
      #Implementing LSTM layer with o/p from Embedding layer
      self.lstm_output, self.lstm_state_h,self.lstm_state_c = self.lstm(input_embedd,initial_state = [s
tate_h,state_c])
      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 shape is [32,lstm_units], cell state shape is [32,lstm_units]
      '''
      self.batch_size = batch_size
      #print(tf.zeros((batch_size,self.lstm_size)).shape)
      return tf.zeros((batch_size,self.lstm_size)), tf.zeros((batch_size,self.lstm_size))
```

Grader function - 1

In [18]:

```
def grader_check_encoder():
    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_size,minval=0,dtype=t
f.int32)
    state_h,state_c=encoder.initialize_states(batch_size)
    encoder_output,state_h,state_c=encoder(input_sequence,state_h,state_c)

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

```
True
```

Attention

In [19]:

```
from tensorflow.python.keras import backend as k
from tensorflow.keras.layers import Dot

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

    if self.scoring_function=='dot':
      # No requirement of variables for this score function
```

```
      pass
    if scoring_function == 'general':
      # Intializing Weight dense layer with att_units
      self.W = tf.keras.layers.Dense(att_units)
      pass
    elif scoring_function == 'concat':
      # Intializing two Weight dense layer with att_units and one value Dense layer
      self.W1 = tf.keras.layers.Dense(att_units)
      self.W2 = tf.keras.layers.Dense(att_units)
      self.V = tf.keras.layers.Dense(1)
      pass


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

    if self.scoring_function == 'dot':
        # Implement Dot score function here
        # Dot-score = h_t*transpose(h_s)
        score = tf.matmul(encoder_output, tf.expand_dims(decoder_hidden_state, 1), transpose_b=True)
        #Applying softmax layer inorder to get Attention_weights
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector is product of attention weights and output from encoder
        context_vector = attention_weights * encoder_output
        context_vector = tf.reduce_sum(context_vector, axis=1)

        pass
    elif self.scoring_function == 'general':
        # Implement General score function here
        # Dot-score = (W*h_t)*transpose(h_s), applying weight matrix on encoder_output
        score = tf.matmul(self.W(encoder_output), tf.expand_dims(decoder_hidden_state, 1), transpose_b=
True)
        #Applying softmax layer inorder to get Attention_weights
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector is product of attention weights and output from encoder
        context_vector = attention_weights * encoder_output
        context_vector = tf.reduce_sum(context_vector, axis=1)

        pass
    elif self.scoring_function == 'concat':
        # Implement General score function here
        query_with_time_axis = tf.expand_dims(decoder_hidden_state, 1)
    # score shape == (batch_size, max_length, 1)
    # we get 1 at the last axis because we are applying score to self.V
    # the shape of the tensor before applying self.V is (batch_size, max_length, units)
        score = self.V(tf.nn.tanh(
        self.W1(query_with_time_axis) + self.W2(encoder_output)))

    # attention_weights shape == (batch_size, max_length, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

    # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * encoder_output
        context_vector = tf.reduce_sum(context_vector, axis=1)
        pass

    return context_vector, attention_weights
```

Grader function - 2

In [20]:

```
def grader_check_attention(scoring_fun):
    input_length=10
    vocab_size=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.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

In [21]:

```
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.tar_vocab_size = tar_vocab_size
      self.embedding_dim = embedding_dim
      self.input_length = input_length
      self.dec_units = dec_units
      self.score_fun = score_fun
      self.att_units = att_units
      #Initialsing Embedding layer
      self.embedding = tf.keras.layers.Embedding(tar_vocab_size, embedding_dim)
      #Initialising Lstm layer
      self.LSTM = CuDNNLSTM(self.dec_units,
                             return_sequences=True,
                             return_state=True,
                             recurrent_initializer='glorot_uniform',time_major=False)
      self.fc = tf.keras.layers.Dense(tar_vocab_size)

    # used for attention
      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(1,1,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
    '''

    x = self.embedding(input_to_decoder)

    context_vector, attention_weights = self.attention(state_h, encoder_output)

    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)
    output, state_h,state_c = self.LSTM(x, initial_state=[state_h,state_c])
    output1 = tf.reshape(output, (-1, output.shape[2]))

    x = self.fc(output1)

    return x, state_h, state_c, attention_weights, context_vector
```

Grader function - 3

In [22]:

```
def grader_onestepdecoder(score_fun):
    vocab_size=13
    embedding_dim=12
    input_length=10
    dec_units=16
    att_units=16
    batch_size=32
    onestepdecoder=One_Step_Decoder(vocab_size, embedding_dim, input_length, dec_units ,score_fun ,att_
units)
    input_to_decoder=tf.random.uniform(shape=(batch_size,1),maxval=10,minval=0,dtype=tf.int32)
    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_to_decoder,encoder_out
put,state_h,state_c)
    assert(output.shape==(batch_size,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

In [23]:

```
class Decoder(tf.keras.Model):
    def __init__(self,out_vocab_size, embedding_dim, output_length, dec_units ,score_fun ,att_units):
      #Intialize necessary variables and create an object from the class onestepdecoder
      super(Decoder, self).__init__()
      self.out_vocab_size = out_vocab_size
      self.embedding_dim = embedding_dim
      self.output_length = output_length
      self.dec_units = dec_units
      self.score_fun = score_fun
      self.att_units = att_units
      self.onestepdecoder = One_Step_Decoder(self.out_vocab_size, self.embedding_dim, self.output_lengt
h,
                                             self.dec_units ,self.score_fun ,self.att_units)


    def call(self, input_to_decoder,encoder_output,decoder_hidden_state,decoder_cell_state ):

        #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 = input_to_decoder.shape[1], name = 'output_array
s')

        #Iterate till the length of the decoder input
        for timestep in range(input_to_decoder.shape[1]):
            # Call onestepdecoder for each token in decoder_input
            output,state_h,state_c,attention_weights,context_vector=self.onestepdecoder(input_to_decode
r[:,timestep:timestep+1],
                                                                                          encoder_outpu
t,
                                                                                          decoder_hidde
n_state,
                                                                                        decoder_cell_st
ate)
            # Store the output in tensorarray
            all_outputs = all_outputs.write(timestep,output)
            self.decoder_hidden_state = state_h
            self.decoder_cell_state = state_c
```

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

        return all_outputs
```

Grader function - 4

In [24]:

```
def grader_decoder(score_fun):
    out_vocab_size=13
    embedding_dim=12
    input_length=10
    output_length=11
    dec_units=16
    att_units=16
    batch_size=32

    target_sentences=tf.random.uniform(shape=(batch_size,output_length),maxval=10,minval=0,dtype=tf.int
32)
    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, output_length, dec_units ,score_fun ,att_units)
    output=decoder(target_sentences,encoder_output, state_h, state_c)
    assert(output.shape==(batch_size,output_length,out_vocab_size))
    return True
print(grader_decoder('dot'))
print(grader_decoder('general'))
print(grader_decoder('concat'))
```

```
True
True
True
```

Encoder Decoder model

In [25]:

```
class encoder_decoder(tf.keras.Model):
  def __init__(self,vocab_size_enc,vocab_size_dec,embedding_dim_enc,embedding_dim_dec,lstm_size,
               input_length,output_length,dec_units,score_fun,att_units,batch_size,enc_input,dec_input)
:
    #Intialize objects from encoder decoder
    super(encoder_decoder, self).__init__()

    self.vocab_size_enc = vocab_size_enc
    self.vocab_size_dec = vocab_size_dec
    self.embedding_dim_enc = embedding_dim_enc
    self.embedding_dim_dec = embedding_dim_dec
    self.lstm_size = lstm_size
    self.input_length = input_length
    self.output_length = output_length
    self.dec_units = dec_units
    self.score_fun = score_fun
    self.att_units = att_units
    self.batch_size = batch_size
    self.enc_input = enc_input
    self.dec_input = dec_input

    self.encoder=Encoder(vocab_size_enc,embedding_dim_enc,lstm_size,input_length)

    self.decoder=Decoder(vocab_size_dec, embedding_dim_dec, output_length, dec_units ,score_fun ,att_un
its)

  def call(self,data):
    #Intialize encoder states, Pass the encoder_sequence to the embedding layer
```

```
    initial_state=encoder.initialize_states(batch_size)
    encoder_output,state_h,state_c=encoder(enc_input,initial_state)

    # Decoder initial states are encoder final states, Initialize it accordingly
    # Pass the decoder sequence,encoder_output,decoder states to Decoder
    output=decoder(dec_input,encoder_output, state_h, state_c)

    # return the decoder output
    return output
```

Custom loss function

In [26]:

```
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')

def custom_lossfunction(targets,logits):
  # Custom loss function that will not consider the loss for padded zeros.
  # Refer https://www.tensorflow.org/tutorials/text/nmt_with_attention#define_the_optimizer_and_the_los
s_function

  mask = tf.math.logical_not(tf.math.equal(targets, 0))
  loss_ = loss_object(targets, logits)

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

  return tf.reduce_mean(loss_)
```

Training

In [50]:

```
# 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.

'''Implementation of Teacher-Forcing Method'''
def train_step(inp, targ, state_h,state_c, score_fun):
  loss = 0

  with tf.GradientTape() as tape:

    encoder_output,state_h,state_c=encoder(inp,state_h,state_c)

    dec_state_h = state_h
    dec_state_c = state_c

    dec_input = tf.expand_dims([targ_lang.word_index['<start>']] * BATCH_SIZE, 1)

    # Teacher forcing - feeding the target as the next input
    for t in range(1, targ.shape[1]):
      # passing enc_output to the decoder

      predictions,dec_state_h,dec_state_c,_,_ = onestepdecoder(dec_input, encoder_output, dec_state_h,
dec_state_c)

      loss += custom_lossfunction(targ[:, t], predictions)

      # using teacher forcing
```

```
      dec_input = tf.expand_dims(targ[:, t], 1)


  batch_loss = (loss / int(targ.shape[1]))

  variables = encoder.trainable_variables + onestepdecoder.trainable_variables

  gradients = tape.gradient(loss, variables)

  optimizer.apply_gradients(zip(gradients, variables))

  return batch_loss
```

In [72]:

```
import time

def dot_func(score,EPOCHS):

  tf.config.experimental_run_functions_eagerly(True)

  for epoch in range(EPOCHS):
    start = time.time()
    #initializing states for encoder
    state_h,state_c=encoder.initialize_states(BATCH_SIZE)
    total_loss = 0

    for (batch, (inp, targ)) in enumerate(dataset.take(steps_per_epoch)):
      batch_loss = train_step(inp, targ, state_h,state_c, 'concat')

      total_loss += batch_loss

      if batch % 100 == 0:
        print('Epoch {} Batch {} Loss {:.4f}'.format(epoch + 1,
                                                   batch,
                                                   batch_loss.numpy()))
  # saving (checkpoint) the model every 2 epochs
    if (epoch + 1) % 2 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print('Epoch {} Loss {:.4f}'.format(epoch + 1,
                                      total_loss / steps_per_epoch))
    print('Time taken for {} epoch {} sec\n'.format(epoch+1,time.time() - start))
```

Inference

Plot attention weights

In [53]:

```
from matplotlib import ticker

def plot_attention(attention, sentence, predicted_sentence):
  #Refer: https://www.tensorflow.org/tutorials/text/nmt_with_attention#translate
  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

Predict the sentence translation

In [54]:

```
def evaluate(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 onest
epdecoder.
  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
  '''

  attention_plot = np.zeros((max_length_targ, max_length_inp))

  sentence = sentence.strip()

  #getting word indexes from tensors
  inputs = [inp_lang.word_index[i] for i in sentence.split(' ')]
  #padding data
  inputs = tf.keras.preprocessing.sequence.pad_sequences([inputs],
                                                         maxlen=max_length_inp,
                                                         padding='post')
  #conversion data to tensors
  inputs = tf.convert_to_tensor(inputs)

  result = ''

  #Creating hidden layers for encoder
  hidden1 = tf.zeros((1, units))
  hidden2 = tf.zeros((1, units))

  #Calling encoder function
  enc_out, state_h,state_c = encoder(inputs, hidden1, hidden2)

  #Making encoder final states as decoder initial states
  dec_state_h = state_h
  dec_state_c = state_c

  #Getting decoder input for first time step
  dec_input = tf.expand_dims([targ_lang.word_index['<start>']], 1)

  for t in range(max_length_targ):
    predictions,dec_state_h,dec_state_c,attention_weights,_ = onestepdecoder(dec_input, enc_out, dec_st
ate_h,
                                                                            dec_state_c)

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

    #getting predicted word from training using argmax function
    predicted_id = tf.argmax(predictions[0]).numpy()

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

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

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

  return result, sentence, attention_plot
```

In [55]:

```
def predict(input_sentence):

  result, sentence, attention_plot = evaluate(input_sentence)
```

```
    return result, sentence, attention_plot
```

In [57]:

```
def convert_tensor(lang, tensor):
    str=""
    for t in tensor:
        if t!=0:
            str+=" "+lang.index_word[t]
            #print ("%d ----> %s" % (t, lang.index_word[t]))
    return str
```

Calculate BLEU score

In [62]:

```
#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 = ['it is ship'.split(),] # the original
translation = 'it is ship'.split() # trasilated using model
print(reference, translation)
print('BLEU score: {}'.format(bleu.sentence_bleu(reference, translation)))
```

```
[['it', 'is', 'ship']] ['it', 'is', 'ship']
BLEU score: 1.0
```

Implement concat function here.

In [ ]:

```
score_fun='concat'
encoder=Encoder(vocab_inp_size,embedding_dim,units,10)
onestepdecoder=One_Step_Decoder(vocab_tar_size, embedding_dim, 10, units ,score_fun ,units)
```

In [ ]:

```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(optimizer=optimizer,
                                 encoder=encoder,
                                 decoder=onestepdecoder)
```

In [ ]:

```
dot_func(score_fun,epoch)
```

```
Epoch 1 Batch 0 Loss 4.8225
Epoch 1 Batch 100 Loss 2.0534
Epoch 1 Batch 200 Loss 1.8305
Epoch 1 Batch 300 Loss 1.6883
Epoch 1 Batch 400 Loss 1.5518
Epoch 1 Batch 500 Loss 1.3990
Epoch 1 Batch 600 Loss 1.1998
Epoch 1 Loss 1.7263
Time taken for 1 epoch 149.20565676689148 sec

Epoch 2 Batch 0 Loss 1.2393
Epoch 2 Batch 100 Loss 1.1061
```

```
Epoch 2 Batch 200 Loss 1.0771
Epoch 2 Batch 300 Loss 0.9988
Epoch 2 Batch 400 Loss 0.9385
Epoch 2 Batch 500 Loss 0.9077
Epoch 2 Batch 600 Loss 0.8667
Epoch 2 Loss 0.9926
Time taken for 2 epoch 150.26364302635193 sec

Epoch 3 Batch 0 Loss 0.7706
Epoch 3 Batch 100 Loss 0.7096
Epoch 3 Batch 200 Loss 0.6651
Epoch 3 Batch 300 Loss 0.6031
Epoch 3 Batch 400 Loss 0.6181
Epoch 3 Batch 500 Loss 0.6101
Epoch 3 Batch 600 Loss 0.5254
Epoch 3 Loss 0.6351
Time taken for 3 epoch 149.6530704498291 sec

Epoch 4 Batch 0 Loss 0.4824
Epoch 4 Batch 100 Loss 0.4643
Epoch 4 Batch 200 Loss 0.4314
Epoch 4 Batch 300 Loss 0.4238
Epoch 4 Batch 400 Loss 0.4116
Epoch 4 Batch 500 Loss 0.4365
Epoch 4 Batch 600 Loss 0.3971
Epoch 4 Loss 0.4195
Time taken for 4 epoch 149.61131644248962 sec

Epoch 5 Batch 0 Loss 0.2901
Epoch 5 Batch 100 Loss 0.2527
Epoch 5 Batch 200 Loss 0.2560
Epoch 5 Batch 300 Loss 0.2692
Epoch 5 Batch 400 Loss 0.2936
Epoch 5 Batch 500 Loss 0.2736
Epoch 5 Batch 600 Loss 0.2697
Epoch 5 Loss 0.2836
Time taken for 5 epoch 151.90541648864746 sec

Epoch 6 Batch 0 Loss 0.2244
Epoch 6 Batch 100 Loss 0.2046
Epoch 6 Batch 200 Loss 0.1897
Epoch 6 Batch 300 Loss 0.2364
Epoch 6 Batch 400 Loss 0.2081
Epoch 6 Batch 500 Loss 0.1617
Epoch 6 Batch 600 Loss 0.2085
Epoch 6 Loss 0.1991
Time taken for 6 epoch 149.54982113838196 sec

Epoch 7 Batch 0 Loss 0.1335
Epoch 7 Batch 100 Loss 0.1559
Epoch 7 Batch 200 Loss 0.1325
Epoch 7 Batch 300 Loss 0.1666
Epoch 7 Batch 400 Loss 0.1714
Epoch 7 Batch 500 Loss 0.1962
Epoch 7 Batch 600 Loss 0.1475
Epoch 7 Loss 0.1463
Time taken for 7 epoch 148.700261592865 sec

Epoch 8 Batch 0 Loss 0.1120
Epoch 8 Batch 100 Loss 0.1081
Epoch 8 Batch 200 Loss 0.1064
Epoch 8 Batch 300 Loss 0.1020
Epoch 8 Batch 400 Loss 0.1090
Epoch 8 Batch 500 Loss 0.1178
Epoch 8 Batch 600 Loss 0.1152
Epoch 8 Loss 0.1132
Time taken for 8 epoch 149.15125131607056 sec

Epoch 9 Batch 0 Loss 0.0816
Epoch 9 Batch 100 Loss 0.0904
Epoch 9 Batch 200 Loss 0.1005
Epoch 9 Batch 300 Loss 0.1036
Epoch 9 Batch 400 Loss 0.0946
Epoch 9 Batch 500 Loss 0.0948
Epoch 9 Batch 600 Loss 0.1137
Epoch 9 Loss 0.0924
Time taken for 9 epoch 150.0213484764099 sec
```

```
Epoch 10 Batch 0 Loss 0.0781
Epoch 10 Batch 100 Loss 0.0771
Epoch 10 Batch 200 Loss 0.0850
Epoch 10 Batch 300 Loss 0.0627
Epoch 10 Batch 400 Loss 0.0737
Epoch 10 Batch 500 Loss 0.0686
Epoch 10 Batch 600 Loss 0.0881
Epoch 10 Loss 0.0781
Time taken for 10 epoch 150.69288563728333 sec

Epoch 11 Batch 0 Loss 0.0625
Epoch 11 Batch 100 Loss 0.0594
Epoch 11 Batch 200 Loss 0.0467
Epoch 11 Batch 300 Loss 0.0622
Epoch 11 Batch 400 Loss 0.0865
Epoch 11 Batch 500 Loss 0.0722
Epoch 11 Batch 600 Loss 0.0717
Epoch 11 Loss 0.0698
Time taken for 11 epoch 150.25997757911682 sec

Epoch 12 Batch 0 Loss 0.0708
Epoch 12 Batch 100 Loss 0.0472
Epoch 12 Batch 200 Loss 0.0716
Epoch 12 Batch 300 Loss 0.0727
Epoch 12 Batch 400 Loss 0.0575
Epoch 12 Batch 500 Loss 0.0613
Epoch 12 Batch 600 Loss 0.0530
Epoch 12 Loss 0.0623
Time taken for 12 epoch 151.09101271629333 sec

Epoch 13 Batch 0 Loss 0.0723
Epoch 13 Batch 100 Loss 0.0578
Epoch 13 Batch 200 Loss 0.0761
Epoch 13 Batch 300 Loss 0.0490
Epoch 13 Batch 400 Loss 0.0789
Epoch 13 Batch 500 Loss 0.0550
Epoch 13 Batch 600 Loss 0.0889
Epoch 13 Loss 0.0585
Time taken for 13 epoch 150.03408217430115 sec

Epoch 14 Batch 0 Loss 0.0500
Epoch 14 Batch 100 Loss 0.0410
Epoch 14 Batch 200 Loss 0.0332
Epoch 14 Batch 300 Loss 0.0527
Epoch 14 Batch 400 Loss 0.0596
Epoch 14 Batch 500 Loss 0.0554
Epoch 14 Batch 600 Loss 0.0663
Epoch 14 Loss 0.0546
Time taken for 14 epoch 151.00791144371033 sec

Epoch 15 Batch 0 Loss 0.0351
Epoch 15 Batch 100 Loss 0.0313
Epoch 15 Batch 200 Loss 0.0557
Epoch 15 Batch 300 Loss 0.0469
Epoch 15 Batch 400 Loss 0.0727
Epoch 15 Batch 500 Loss 0.0925
Epoch 15 Batch 600 Loss 0.0559
Epoch 15 Loss 0.0520
Time taken for 15 epoch 150.58883213996887 sec
```

In [56]:

```
# restoring the latest checkpoint in checkpoint_dir for Concat Function
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

Out[56]:

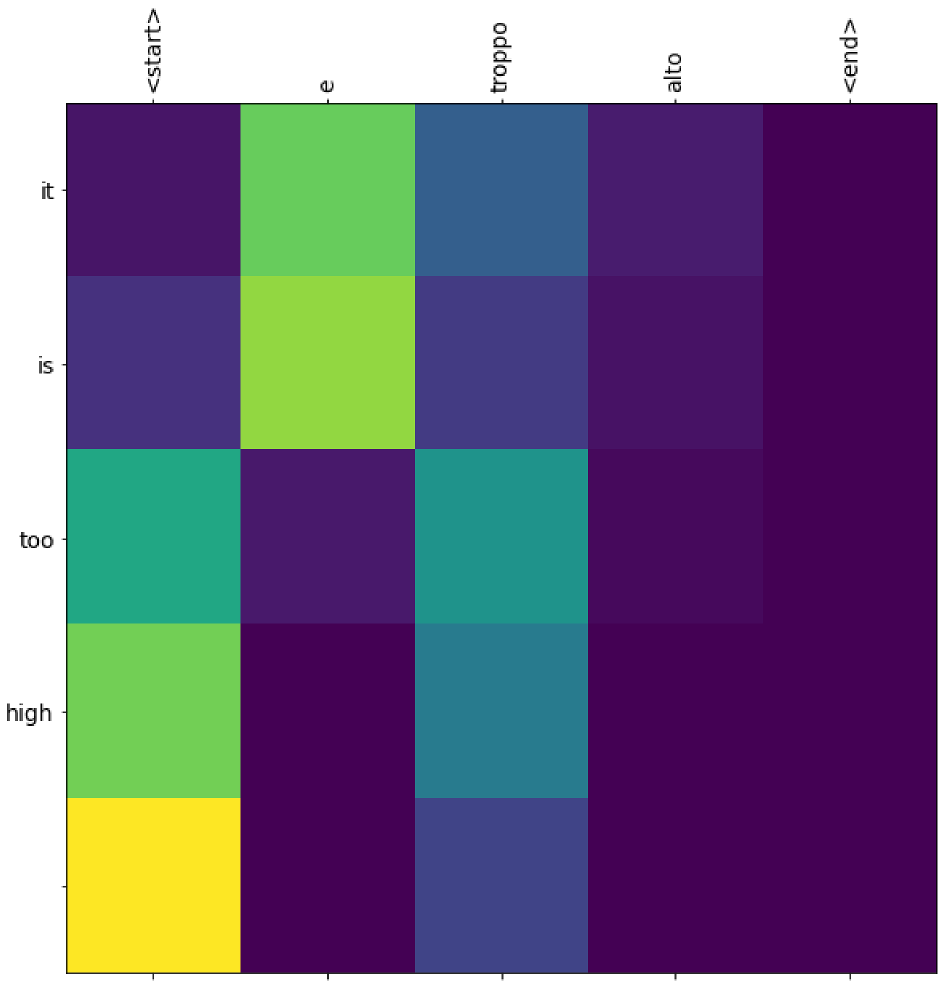
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f5c700f6da0>
```

Testing of concat function

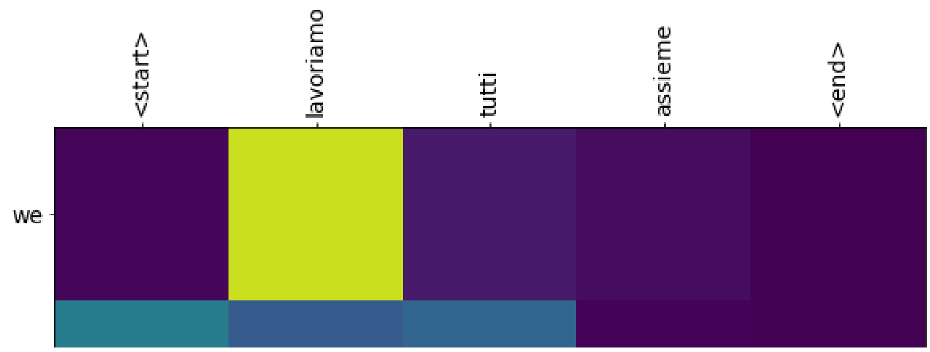
In [122]:

```
#testing of Concat function
import random
randomlist = random.sample(range(20000), 5)
for i in randomlist:
  print('Actual sentence: {}'.format(convert_tensor(targ_lang,target_tensor_val[i])))
  result,sentence,attention_plot = predict(convert_tensor(inp_lang,input_tensor_val[i]))
  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(' '))
```

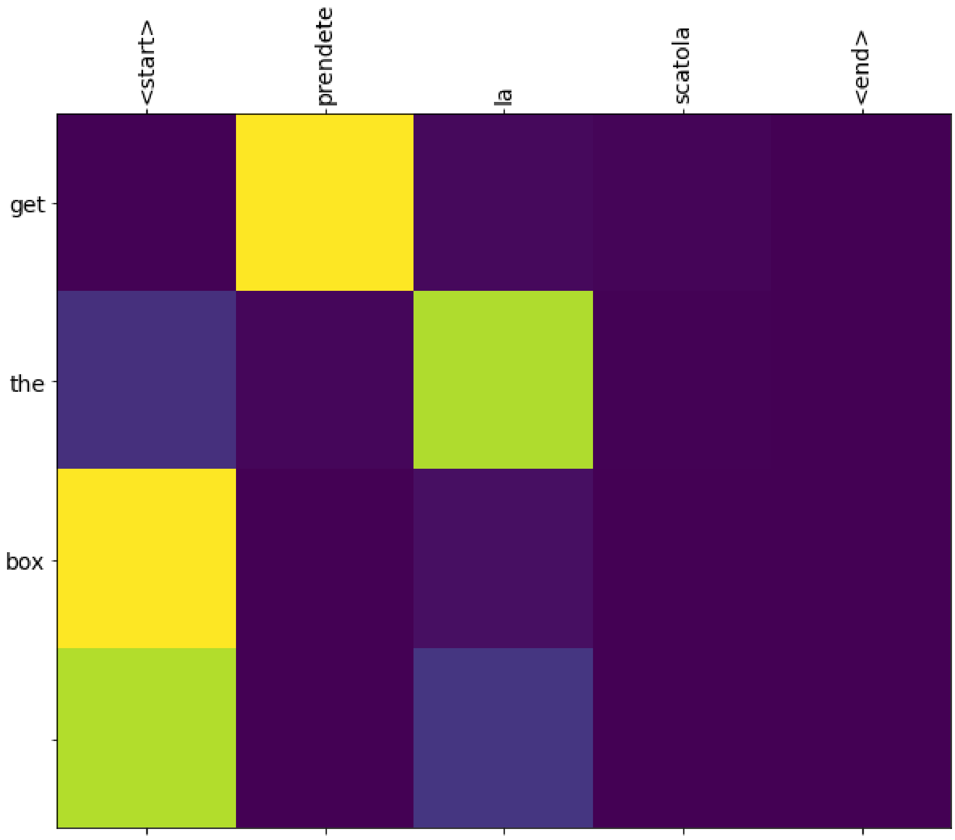
```
Actual sentence:  <start> it is too loud <end>
Input: <start> e troppo alto <end>
Predicted translation: it is too high
```



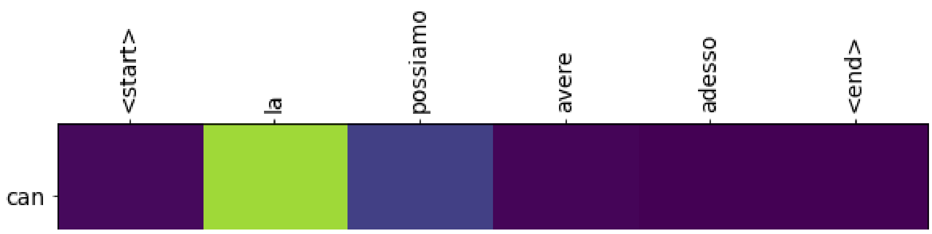

```
Actual sentence:  <start> we all work together <end>
Input: <start> lavoriamo tutti assieme <end>
Predicted translation: we all work together
```



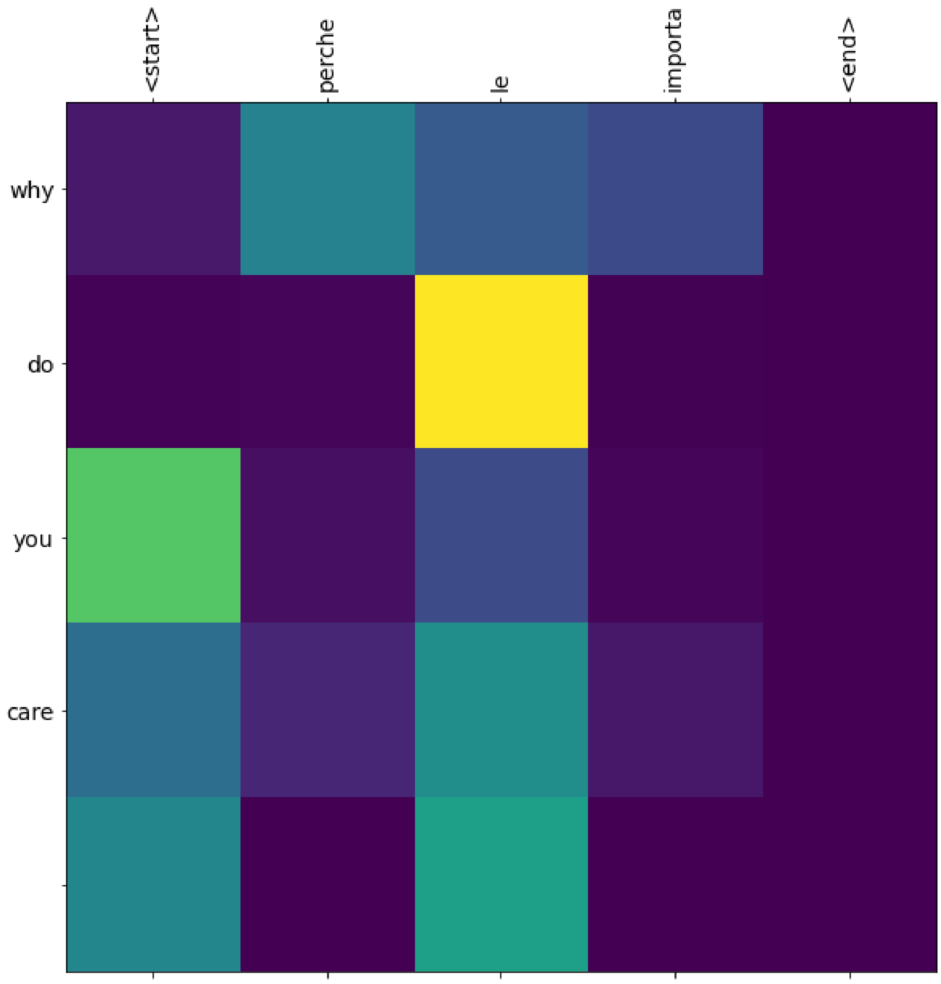

```
Actual sentence:  <start> get the box <end>
Input: <start> prendete la scatola <end>
Predicted translation: get the box
```




```
Actual sentence:  <start> can we have it now <end>
Input: <start> la possiamo avere adesso <end>
Predicted translation: can we have it now
```




```
Actual sentence:  <start> why do you care <end>
Input: <start> perche le importa <end>
Predicted translation: why do you care
```



Bleu score calculation for concat function

In [64]:

```
#Bleu score calculation for Concat function
randomlist = random.sample(range(20000), 1000)
bleu_avg=[]
for i in tqdm(randomlist):
  str=""
  for t in target_tensor_val[i]:
    if t>2:
      str+=" "+targ_lang.index_word[t]
  #print(str)
  result,sentence,attention_plot = predict(convert_tensor(inp_lang,input_tensor_val[i]))
  bleu_avg.append(bleu.sentence_bleu([str.strip().split()],result.split()))
print()
print(sum(bleu_avg)/len(bleu_avg))
```

```
100%|██████████| 1000/1000 [00:49<00:00, 20.26it/s]
```

```
0.8634698688282089
```

Repeat the same steps for Dot scoring function

In [ ]:

```
score_fun='dot'
encoder=Encoder(vocab_inp_size,embedding_dim,units,10)
onestepdecoder=One_Step_Decoder(vocab_tar_size, embedding_dim, 10, units ,score_fun ,units)
```

In [ ]:

```
checkpoint_dir2 = './training_checkpoints2'
checkpoint_prefix2 = os.path.join(checkpoint_dir2, "ckpt")
checkpoint2 = tf.train.Checkpoint(optimizer=optimizer,
                                 encoder=encoder,
                                 decoder=onestepdecoder)
```

In [ ]:

```
dot_func(score_fun,10)
```

```
Epoch 1 Batch 0 Loss 4.7692
Epoch 1 Batch 100 Loss 2.2148
Epoch 1 Batch 200 Loss 2.0480
Epoch 1 Batch 300 Loss 2.0794
Epoch 1 Batch 400 Loss 1.9128
Epoch 1 Batch 500 Loss 1.8664
Epoch 1 Batch 600 Loss 1.7544
Epoch 1 Loss 2.0583
Time taken for 1 epoch 125.67225241661072 sec

Epoch 2 Batch 0 Loss 1.6140
Epoch 2 Batch 100 Loss 1.5356
Epoch 2 Batch 200 Loss 1.4132
Epoch 2 Batch 300 Loss 1.3439
Epoch 2 Batch 400 Loss 1.2252
Epoch 2 Batch 500 Loss 1.2136
Epoch 2 Batch 600 Loss 1.1267
Epoch 2 Loss 1.3539
Time taken for 2 epoch 125.74435210227966 sec

Epoch 3 Batch 0 Loss 1.0620
Epoch 3 Batch 100 Loss 1.0104
Epoch 3 Batch 200 Loss 0.9783
Epoch 3 Batch 300 Loss 0.9392
Epoch 3 Batch 400 Loss 0.8473
Epoch 3 Batch 500 Loss 0.8310
Epoch 3 Batch 600 Loss 0.7718
Epoch 3 Loss 0.8998
Time taken for 3 epoch 125.65568852424622 sec
```

```
Epoch 4 Batch 0 Loss 0.7178
Epoch 4 Batch 100 Loss 0.6094
Epoch 4 Batch 200 Loss 0.6290
Epoch 4 Batch 300 Loss 0.5736
Epoch 4 Batch 400 Loss 0.5551
Epoch 4 Batch 500 Loss 0.4855
Epoch 4 Batch 600 Loss 0.5338
Epoch 4 Loss 0.5863
Time taken for 4 epoch 126.31546354293823 sec

Epoch 5 Batch 0 Loss 0.4735
Epoch 5 Batch 100 Loss 0.4073
Epoch 5 Batch 200 Loss 0.3889
Epoch 5 Batch 300 Loss 0.4331
Epoch 5 Batch 400 Loss 0.4027
Epoch 5 Batch 500 Loss 0.4138
Epoch 5 Batch 600 Loss 0.3603
Epoch 5 Loss 0.3990
Time taken for 5 epoch 126.31635022163391 sec

Epoch 6 Batch 0 Loss 0.2801
Epoch 6 Batch 100 Loss 0.2814
Epoch 6 Batch 200 Loss 0.3157
Epoch 6 Batch 300 Loss 0.3223
Epoch 6 Batch 400 Loss 0.2534
Epoch 6 Batch 500 Loss 0.3016
Epoch 6 Batch 600 Loss 0.3254
Epoch 6 Loss 0.2917
Time taken for 6 epoch 126.3590497970581 sec

Epoch 7 Batch 0 Loss 0.2395
Epoch 7 Batch 100 Loss 0.2117
Epoch 7 Batch 200 Loss 0.2428
Epoch 7 Batch 300 Loss 0.1897
Epoch 7 Batch 400 Loss 0.1855
Epoch 7 Batch 500 Loss 0.2354
Epoch 7 Batch 600 Loss 0.1926
Epoch 7 Loss 0.2252
Time taken for 7 epoch 126.6375789642334 sec

Epoch 8 Batch 0 Loss 0.1786
Epoch 8 Batch 100 Loss 0.1716
Epoch 8 Batch 200 Loss 0.2045
Epoch 8 Batch 300 Loss 0.1587
Epoch 8 Batch 400 Loss 0.1354
Epoch 8 Batch 500 Loss 0.1488
Epoch 8 Batch 600 Loss 0.2220
Epoch 8 Loss 0.1806
Time taken for 8 epoch 127.95059108734131 sec

Epoch 9 Batch 0 Loss 0.1596
Epoch 9 Batch 100 Loss 0.1526
Epoch 9 Batch 200 Loss 0.1843
Epoch 9 Batch 300 Loss 0.1250
Epoch 9 Batch 400 Loss 0.1342
Epoch 9 Batch 500 Loss 0.1802
Epoch 9 Batch 600 Loss 0.1276
Epoch 9 Loss 0.1495
Time taken for 9 epoch 126.89562463760376 sec

Epoch 10 Batch 0 Loss 0.1131
Epoch 10 Batch 100 Loss 0.1052
Epoch 10 Batch 200 Loss 0.1416
Epoch 10 Batch 300 Loss 0.1309
Epoch 10 Batch 400 Loss 0.1442
Epoch 10 Batch 500 Loss 0.1324
Epoch 10 Batch 600 Loss 0.1437
Epoch 10 Loss 0.1273
Time taken for 10 epoch 126.6572208404541 sec
```

In [ ]:

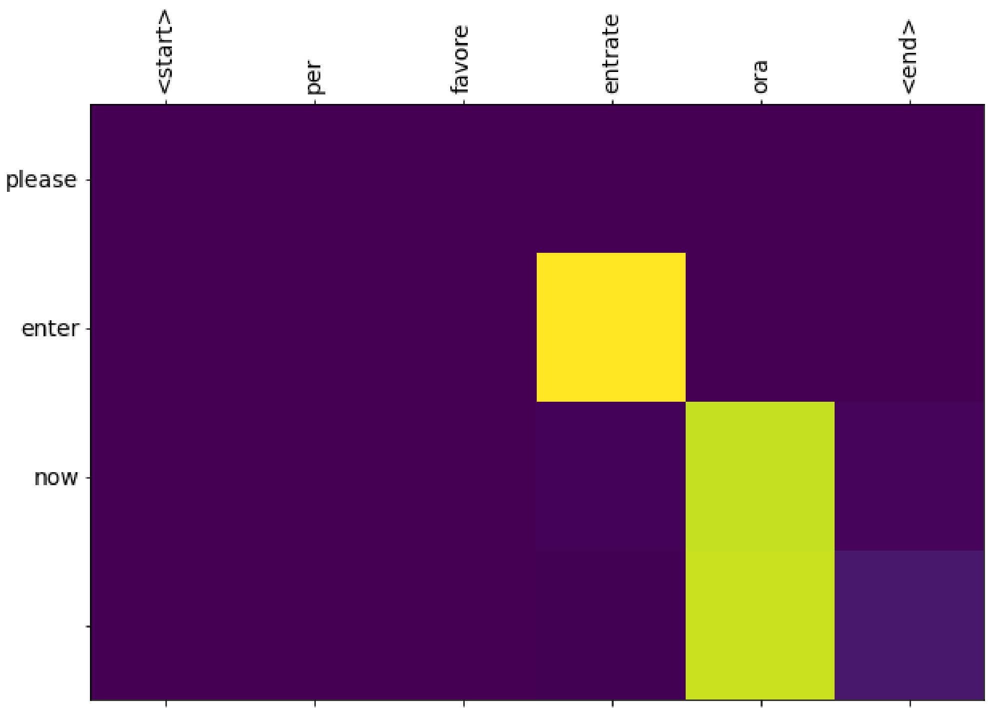
```
# restoring the latest checkpoint in checkpoint_dir for Dot Function
checkpoint2.restore(tf.train.latest_checkpoint(checkpoint_dir2))
```

Testing of Dot function

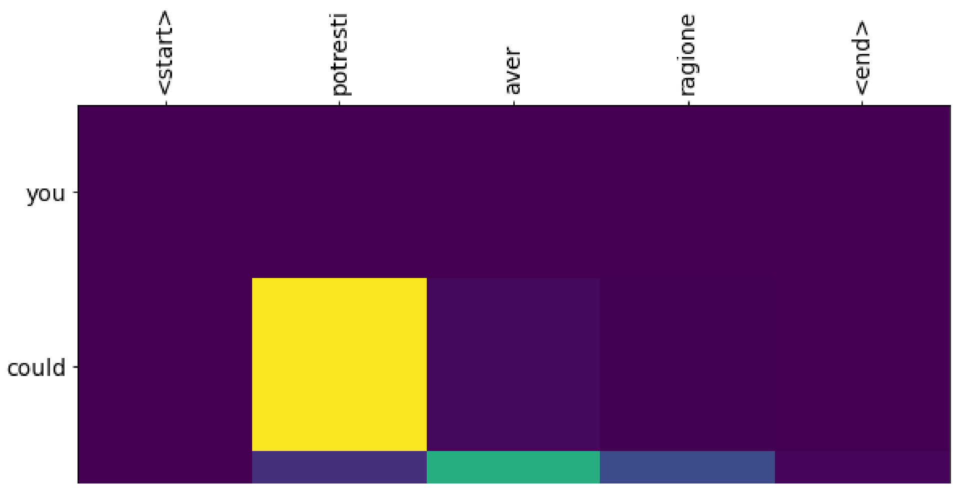
In [83]:

```
import random
randomlist = random.sample(range(20000), 5)
for i in randomlist:
  print('Actual sentence: {}'.format(convert_tensor(targ_lang,target_tensor_val[i])))
  result,sentence,attention_plot = predict(convert_tensor(inp_lang,input_tensor_val[i]))
  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(' '))
```

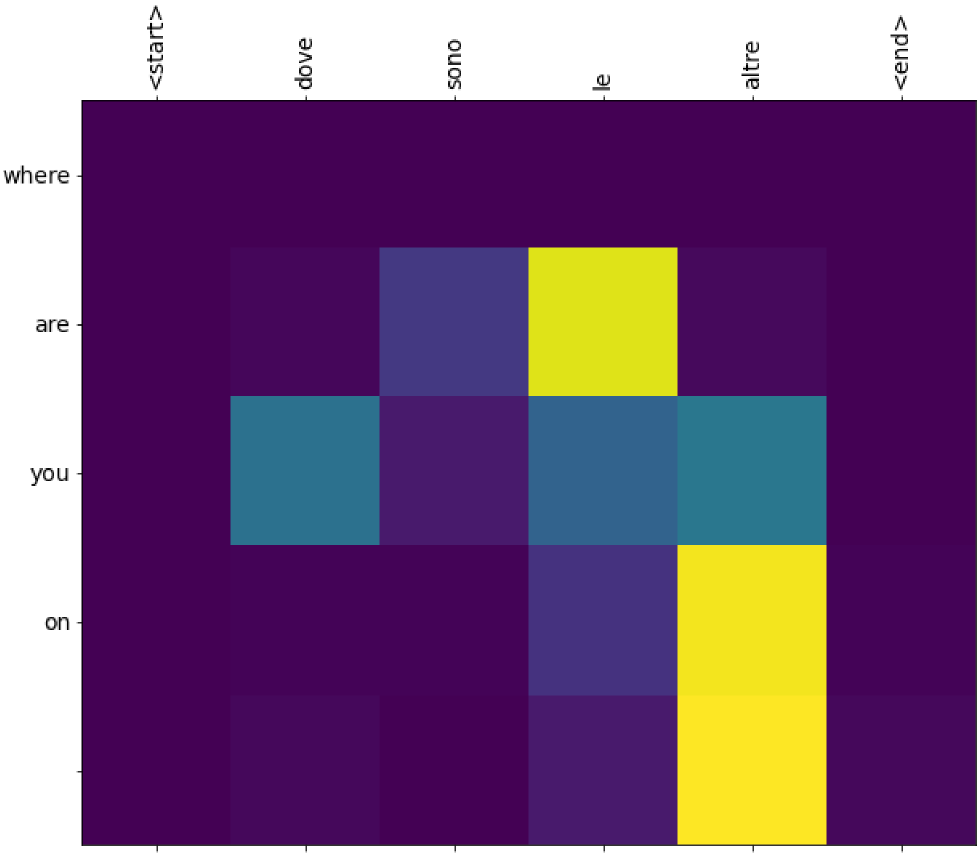
```
Actual sentence:  <start> please enter now <end>
Input: <start> per favore entrate ora <end>
Predicted translation: please enter now
```



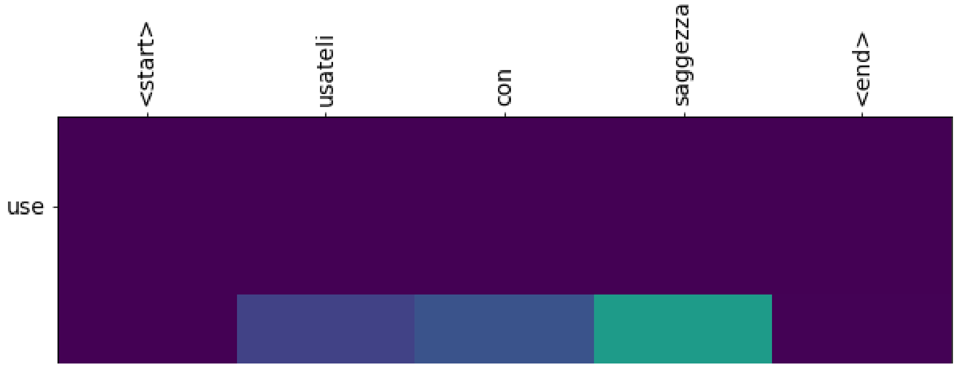

```
Actual sentence:  <start> you might be right <end>
Input: <start> potresti aver ragione <end>
Predicted translation: you could be right
```

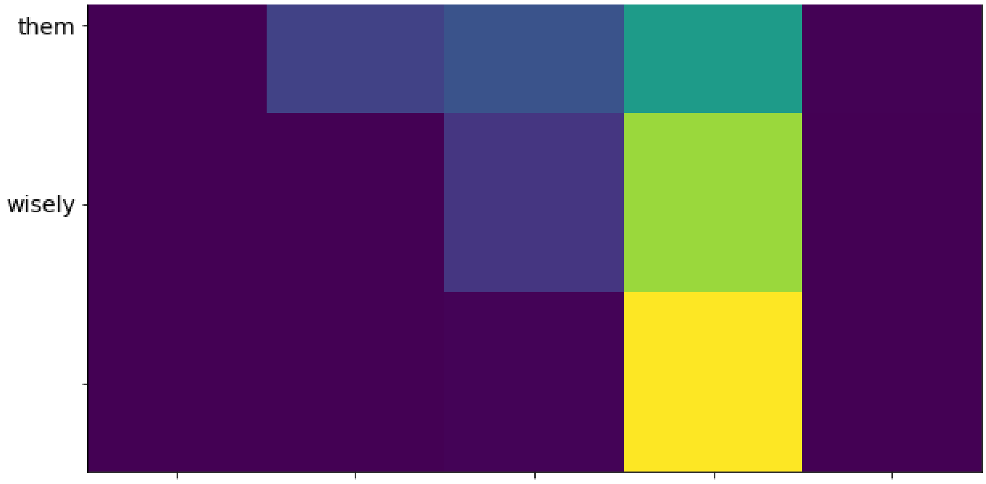



```
Actual sentence:  <start> where are the others <end>
Input: <start> dove sono le altre <end>
Predicted translation: where are you on
```

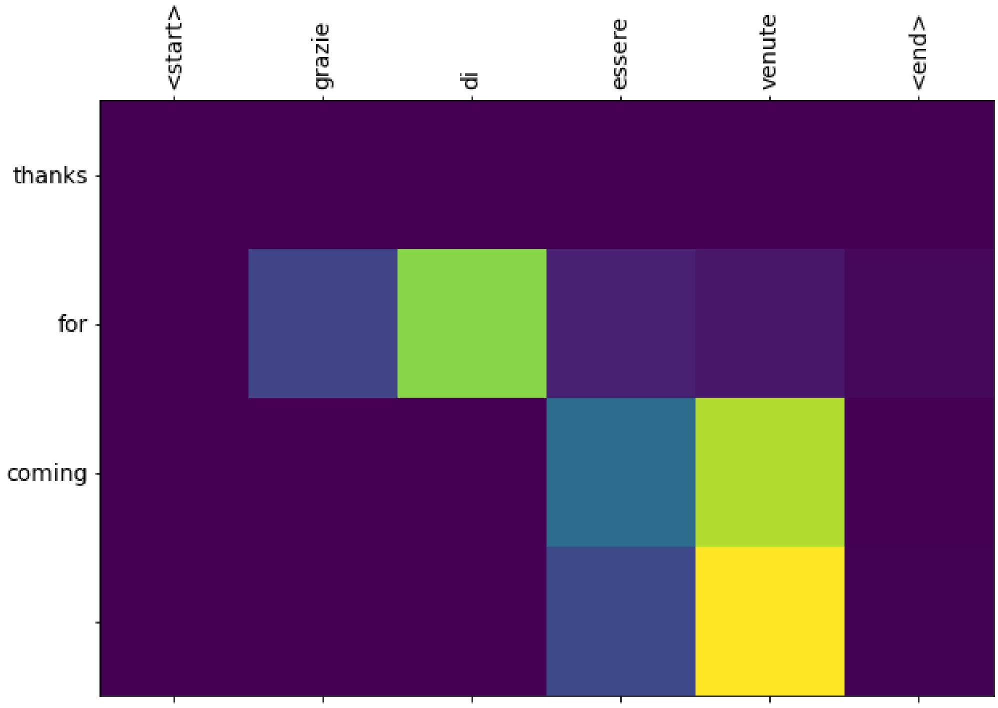



```
Actual sentence:  <start> use them wisely <end>
Input: <start> usateli con saggezza <end>
Predicted translation: use them wisely
```






```
Actual sentence:  <start> thanks for coming <end>
Input: <start> grazie di essere venute <end>
Predicted translation: thanks for coming
```



Bleu score calculation for Dot function

In [84]:

```
#calculation of bleu score for Dot function
randomlist = random.sample(range(20000), 1000)
bleu_avg=[]
for i in tqdm(randomlist):
  str=""
  for t in target_tensor_val[i]:
    if t>2:
      str+=" "+targ_lang.index_word[t]
  #print(str)
  result,sentence,attention_plot = predict(convert_tensor(inp_lang,input_tensor_val[i]))
  bleu_avg.append(bleu.sentence_bleu([str.strip().split()],result.split()))
print()
print(sum(bleu_avg)/len(bleu_avg))
```

```
100%|██████████| 1000/1000 [00:38<00:00, 26.09it/s]
```

```
0.8601504025677253
```

Repeat the same steps for General scoring function

In [ ]:

```
score_fun='general'
encoder=Encoder(vocab_inp_size,embedding_dim,units,10)
onestepdecoder=One_Step_Decoder(vocab_tar_size, embedding_dim, 10, units ,score_fun ,units)
```

In [ ]:

```
!rm -rf 'training_checkpoints3'
```

In [ ]:

```
checkpoint_dir3 = './training_checkpoints3'
checkpoint_prefix3 = os.path.join(checkpoint_dir3, "ckpt")
checkpoint3 = tf.train.Checkpoint(optimizer=optimizer,
                                 encoder=encoder,
                                 decoder=onestepdecoder)
```

In [ ]:

```
dot_func(score_fun,10)
```

```
Epoch 1 Batch 0 Loss 4.7996
Epoch 1 Batch 100 Loss 2.0088
Epoch 1 Batch 200 Loss 1.7828
Epoch 1 Batch 300 Loss 1.7123
Epoch 1 Batch 400 Loss 1.5727
Epoch 1 Batch 500 Loss 1.3512
Epoch 1 Batch 600 Loss 1.3529
Epoch 1 Loss 1.7684
Time taken for 1 epoch 139.67698764801025 sec

Epoch 2 Batch 0 Loss 1.2647
Epoch 2 Batch 100 Loss 1.1500
Epoch 2 Batch 200 Loss 1.1012
Epoch 2 Batch 300 Loss 1.0935
Epoch 2 Batch 400 Loss 1.0605
Epoch 2 Batch 500 Loss 0.9768
Epoch 2 Batch 600 Loss 0.9450
Epoch 2 Loss 1.0921
Time taken for 2 epoch 139.4440314769745 sec

Epoch 3 Batch 0 Loss 0.7987
Epoch 3 Batch 100 Loss 0.7325
Epoch 3 Batch 200 Loss 0.6988
Epoch 3 Batch 300 Loss 0.7123
Epoch 3 Batch 400 Loss 0.6810
Epoch 3 Batch 500 Loss 0.7144
Epoch 3 Batch 600 Loss 0.6102
Epoch 3 Loss 0.7171
Time taken for 3 epoch 139.09080529212952 sec

Epoch 4 Batch 0 Loss 0.4884
Epoch 4 Batch 100 Loss 0.4939
Epoch 4 Batch 200 Loss 0.4652
Epoch 4 Batch 300 Loss 0.4398
Epoch 4 Batch 400 Loss 0.4371
Epoch 4 Batch 500 Loss 0.4478
Epoch 4 Batch 600 Loss 0.4244
Epoch 4 Loss 0.4656
Time taken for 4 epoch 137.75526404380798 sec

Epoch 5 Batch 0 Loss 0.3464
Epoch 5 Batch 100 Loss 0.3097
Epoch 5 Batch 200 Loss 0.2445
Epoch 5 Batch 300 Loss 0.2848
Epoch 5 Batch 400 Loss 0.3092
```

```
Epoch 5 Batch 500 Loss 0.2709
Epoch 5 Batch 600 Loss 0.2659
Epoch 5 Loss 0.3060
Time taken for 5 epoch 136.8633894920349 sec

Epoch 6 Batch 0 Loss 0.2166
Epoch 6 Batch 100 Loss 0.2223
Epoch 6 Batch 200 Loss 0.2278
Epoch 6 Batch 300 Loss 0.2171
Epoch 6 Batch 400 Loss 0.2201
Epoch 6 Batch 500 Loss 0.2360
Epoch 6 Batch 600 Loss 0.2123
Epoch 6 Loss 0.2085
Time taken for 6 epoch 137.0866982936859 sec

Epoch 7 Batch 0 Loss 0.1523
Epoch 7 Batch 100 Loss 0.1486
Epoch 7 Batch 200 Loss 0.1599
Epoch 7 Batch 300 Loss 0.1506
Epoch 7 Batch 400 Loss 0.1634
Epoch 7 Batch 500 Loss 0.1463
Epoch 7 Batch 600 Loss 0.1705
Epoch 7 Loss 0.1499
Time taken for 7 epoch 136.63018131256104 sec

Epoch 8 Batch 0 Loss 0.1113
Epoch 8 Batch 100 Loss 0.1126
Epoch 8 Batch 200 Loss 0.1177
Epoch 8 Batch 300 Loss 0.1071
Epoch 8 Batch 400 Loss 0.0889
Epoch 8 Batch 500 Loss 0.1298
Epoch 8 Batch 600 Loss 0.1207
Epoch 8 Loss 0.1140
Time taken for 8 epoch 137.87583231925964 sec

Epoch 9 Batch 0 Loss 0.0765
Epoch 9 Batch 100 Loss 0.0763
Epoch 9 Batch 200 Loss 0.0885
Epoch 9 Batch 300 Loss 0.1020
Epoch 9 Batch 400 Loss 0.0971
Epoch 9 Batch 500 Loss 0.0933
Epoch 9 Batch 600 Loss 0.1042
Epoch 9 Loss 0.0922
Time taken for 9 epoch 137.4320731163025 sec

Epoch 10 Batch 0 Loss 0.0567
Epoch 10 Batch 100 Loss 0.0771
Epoch 10 Batch 200 Loss 0.0758
Epoch 10 Batch 300 Loss 0.0832
Epoch 10 Batch 400 Loss 0.0762
Epoch 10 Batch 500 Loss 0.0974
Epoch 10 Batch 600 Loss 0.0779
Epoch 10 Loss 0.0784
Time taken for 10 epoch 138.09355425834656 sec
```

In [93]:

```
# restoring the latest checkpoint in checkpoint_dir for General Function
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

Out[93]:

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f5c7fc2e160>
```

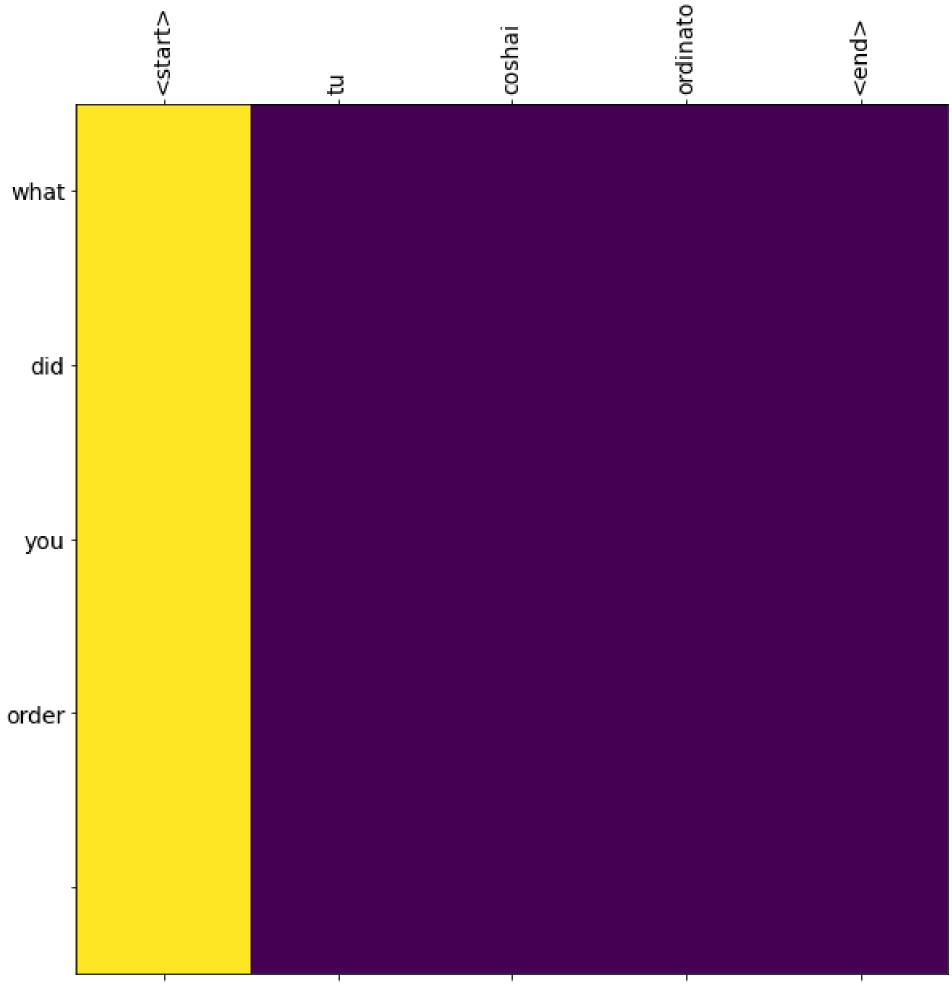
Testing of General function

In [94]:

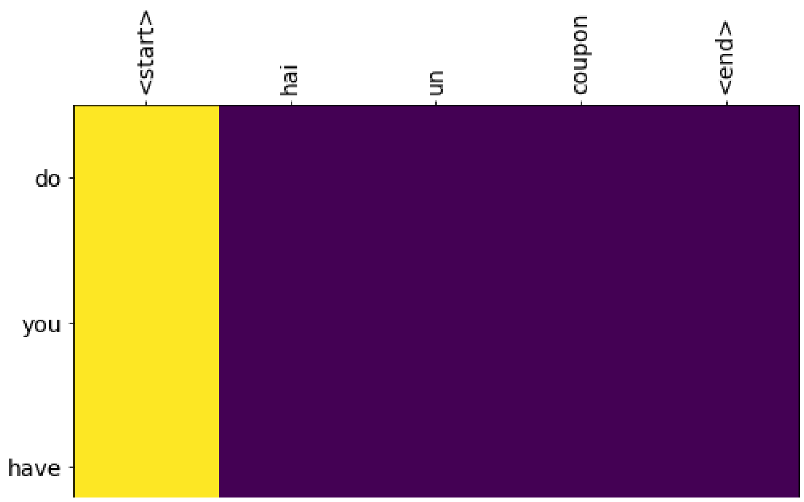
```
#testing of general function
import random
randomlist = random.sample(range(20000), 5)
for i in randomlist:
```

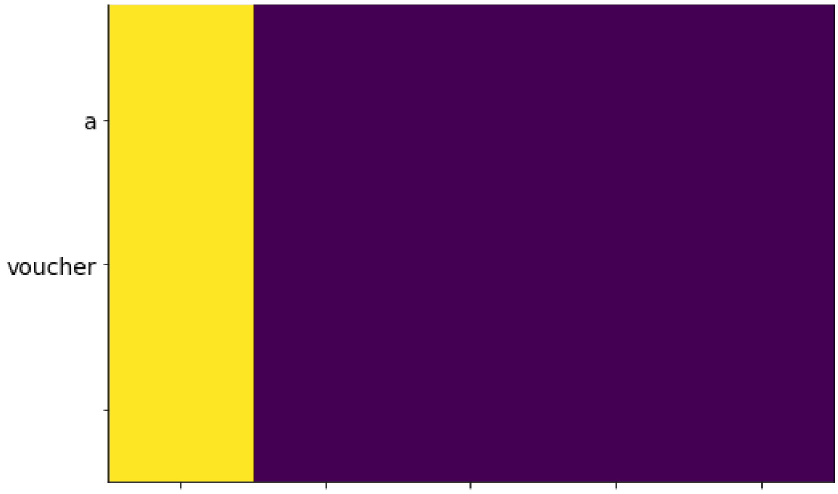
```
print('Actual sentence: {}'.format(convert_tensor(targ_lang,target_tensor_val[i])))
result,sentence,attention_plot = predict(convert_tensor(inp_lang,input_tensor_val[i]))
print('Input: %s' % (sentence))
print('Predicted translation: {}'.format(result))


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

```
Actual sentence:  <start> what did you order <end>
Input: <start> tu coshai ordinato <end>
Predicted translation: what did you order
```

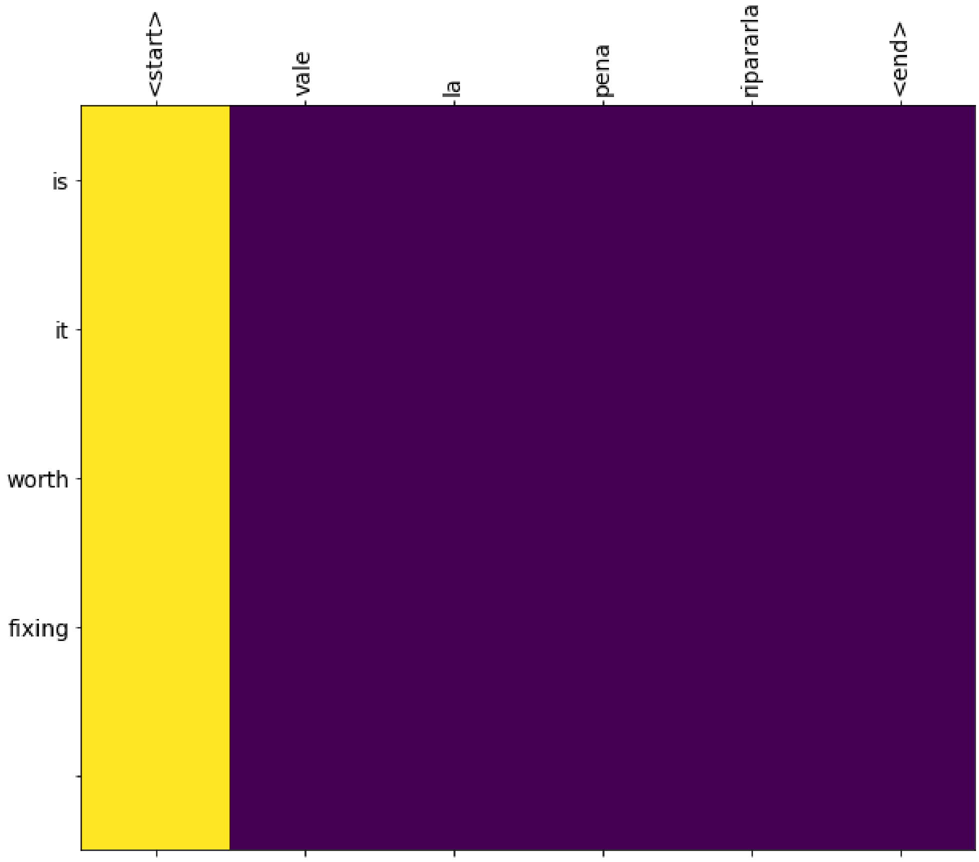



```
Actual sentence:  <start> do you have a voucher <end>
Input: <start> hai un coupon <end>
Predicted translation: do you have a voucher
```

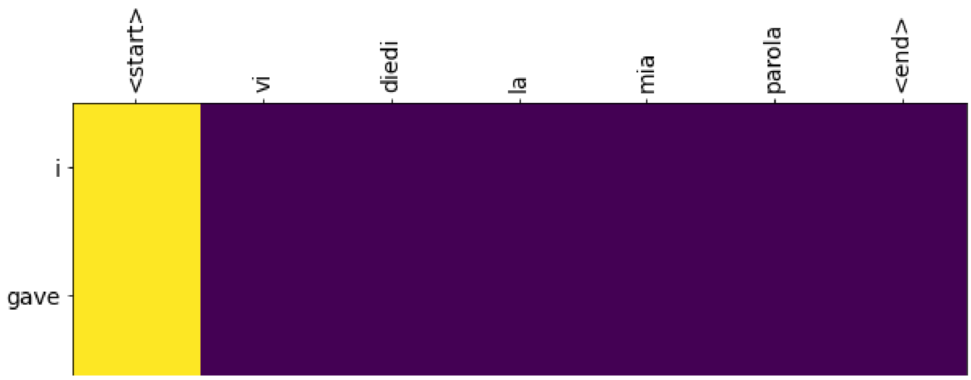




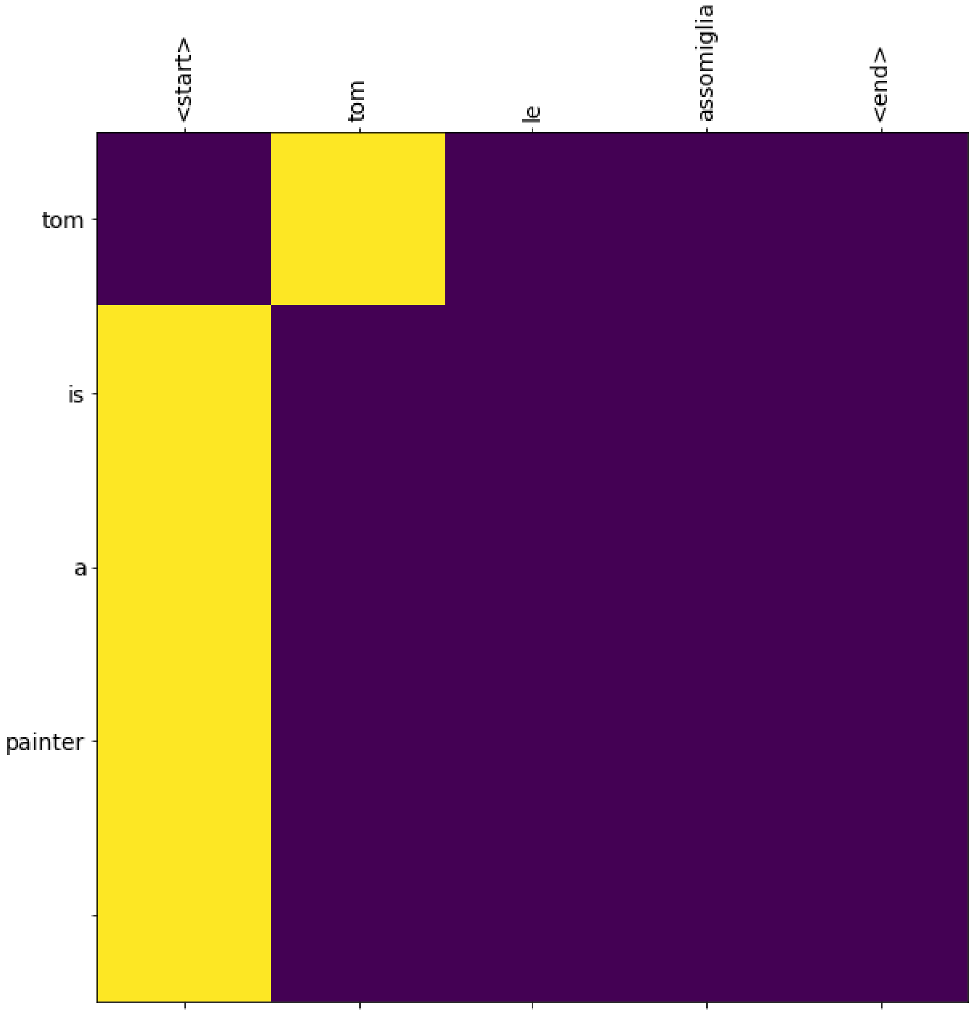

```
Actual sentence:  <start> is it worth fixing <end>
Input: <start> vale la pena ripararla <end>
Predicted translation: is it worth fixing
```




```
Actual sentence:  <start> i gave you my word <end>
Input: <start> vi diedi la mia parola <end>
Predicted translation: i gave you my word
```




```
Actual sentence:  <start> tom looks like you <end>
Input: <start> tom le assomiglia <end>
Predicted translation: tom is a painter
```



Bleu score calculation for General function

In [95]:

```
#Bleu score calculation for General Function
randomlist = random.sample(range(20000), 1000)
bleu_avg=[]
for i in tqdm(randomlist):
  str=""
  for t in target_tensor_val[i]:
    if t>2:
```

```
    if t>2:
      str+=" "+targ_lang.index_word[t]
  #print(str)
  result,sentence,attention_plot = predict(convert_tensor(inp_lang,input_tensor_val[i]))
  bleu_avg.append(bleu.sentence_bleu([str.strip().split()],result.split()))
print()
print(sum(bleu_avg)/len(bleu_avg))
```

```
100%|██████████| 1000/1000 [00:42<00:00, 23.77it/s]
```

```
0.8604354801810152
```

Observations

- For every observation I got Bleu score around 86.01 - 86.34.
- Which seems that for my data and training parameters I have used, Convergence is similar in all cases.
- Even when we see epoch Loss for all the functions, they are converging fast and similarly.
- In this calculation I have tried 10 epochs for General and 15 epochs each for Concat and Dot functions
- Even for 10 epochs also Loss has been decreased drastically for General function.
- While testing some random validation test and plotting attention plots, there our predicted sentence is semantically equals to target sentence.

Procedure

Note:I have used the References which you have mentioned at the top of this notebook for this assignment.

- At first I have downloaded data and stored in dataframe.
- Next, I have preprocessed data and convert into tensors as part of Data Preparation.
- In Encoder function, I have initialized states upon which I built one encoder using data from embedding layer.
- In Attention Mechanism Layer, I have implemented all the Score functions as per the given formula at the top of this Notebook.
- In OneStepDecoder I have used Decoder input with embedding layer data along with Encoder output and Encoder States.
- In Decoder Function, I have used the process to give the entire input to the decoder.
- I have used Adam as an Optimizer.
- In training part I have implemented Teacher Forcing Methodology batchwise and calculated batchwise loss for each epoch.
- In Predict the sentence Translation part, I have used Validation Tensor data.
- Sample testing phase and Bleu score calculation phase also I have used some Random Validation data.