In [1]:

```
!gdown --id 1OurDQUtbWQacvT32HMqFL7vIUrSMllOp
```

```
Downloading...
From: https://drive.google.com/uc?id=1OurDQUtbWQacvT32HMqFL7vIUrSMllOp
To: /content/preprocessed_data.csv
100% 300k/300k [00:00<00:00, 9.72MB/s]
```

In [69]:

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [155]:

```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages (1.5.12)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packages (from kaggle) (5.
0.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from kaggle) (4.41.1)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from kaggle) (2.23.0)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from kaggle) (1.24.3)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages (from kaggle) (1.15.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packages (from kaggle) (2
.8.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from kaggle) (2021.5.30
)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-packages (from python
-slugify->kaggle) (1.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests
->kaggle) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->kag
gle) (2.10)
```

In [156]:

```
"""
To use the Kaggle API, sign up for a Kaggle account at https://www.kaggle.com.
Then go to the 'Account' tab of your user profile (https://www.kaggle.com/<username>/account) and select
This will trigger the download of kaggle.json, a file containing your API credentials.
Upload that file to google colab/google cloud platform
"""
api_token = {"username":"manojkumar83000","key":"a6c354dd1bc5460d07ffb4844b923064"}
```

In [157]:

```
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 /root/.kaggle/kaggle.json
!kaggle datasets download -d yekenot/fasttext-crawl-300d-2m
```

```
mkdir: cannot create directory ‘/root/.kaggle’: File exists
fasttext-crawl-300d-2m.zip: Skipping, found more recently modified local copy (use --force to force
download)
```

In [ ]:

```
!7z e fasttext-crawl-300d-2m.zip -o/content -r
```

In [ ]:

```
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def fasttextModel(gloveFile):
    print ("Loading Fasttext Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}#for storing word and the corresponding embedding vector for that word
    for line in f:
        splitLine = line.split()#splitting the line and storing it in a list
        word = splitLine[0]#getting the first element and storing it in word
        embedding = np.array([float(val) for val in splitLine[1:]])#obtaining corresponding vector for th
        model[word] = embedding#storing word as key and embedding vector for that word as value
    print ("Done.",len(model)," words loaded!")
    return model
model = fasttextModel('/content/crawl-300d-2M.vec')
```

In [158]:

```
df=pd.read_csv('preprocessed_data.csv')
```

In [159]:

```
df.head(4)
```

Out[159]:

	Unnamed: 0	source	target
0	0	U wan me to "chop" seat 4 u nt?\n	Do you want me to reserve seat for you or not?\n
1	1	Yup. U reaching. We order some durian pastry a...	Yeap. You reaching? We ordered some Durian pas...
2	2	They become more ex oredi... Mine is like 25.....	They become more expensive already. Mine is li...
3	3	I'm thai. what do u do?\n	I'm Thai. What do you do?\n

In [160]:

```
def preprocess(x):
  x=x[:-1]
  return x
```

In [161]:

```
df['source']=df['source'].apply(preprocess)
df['target']=df['target'].apply(preprocess)
```

In [162]:

```
df=df[['source','target']]
df.head()
```

Out[162]:

	source	target
0	U wan me to "chop" seat 4 u nt?	Do you want me to reserve seat for you or not?
1	Yup. U reaching. We order some durian pastry a...	Yeap. You reaching? We ordered some Durian pas...
2	They become more ex oredi... Mine is like 25.....	They become more expensive already. Mine is li...
3	I'm thai. what do u do?	I'm Thai. What do you do?
4	Hi! How did your week go? Haven heard from you...	Hi! How did your week go? Haven't heard from y...

In [163]:

```
df.shape
```

Out[163]:

```
(2000, 2)
```

In [169]:

```
def length(text):#for calculating the length of the sentence
    return len(str(text))
```

In [170]:

```
df=df[df['source'].apply(length)<170]
df=df[df['target'].apply(length)<200]
```

In [171]:

```
df.shape
```

Out[171]:

```
(1990, 2)
```

In [172]:

```
df['target_in'] = '<start> ' + df['target'].astype(str)
df['target_out'] = df['target'].astype(str) + ' <end>'
# only for the first sentance add a toke <end> so that we will have <end> in tokenizer
df.head()
```

Out[172]:

	source	target	target_in	target_out
0	U wan me to "chop" seat 4 u nt?	Do you want me to reserve seat for you or not?	<start> Do you want me to reserve seat for you...	Do you want me to reserve seat for you or not?...
1	Yup. U reaching. We order some durian pastry a...	Yeap. You reaching? We ordered some Durian pas...	<start> Yeap. You reaching? We ordered some Du...	Yeap. You reaching? We ordered some Durian pas...
2	They become more ex oredi... Mine is like 25.....	They become more expensive already. Mine is li...	<start> They become more expensive already. Mi...	They become more expensive already. Mine is li...
3	I'm thai. what do u do?	I'm Thai. What do you do?	<start> I'm Thai. What do you do?	I'm Thai. What do you do? <end>
4	Hi! How did your week go? Haven heard from you...	Hi! How did your week go? Haven't heard from y...	<start> Hi! How did your week go? Haven't hear...	Hi! How did your week go? Haven't heard from y...

In [173]:

```
df=df.drop('target',axis=1)
```

In [174]:

```
df.head(4)
```

Out[174]:

	source	target_in	target_out
0	U wan me to "chop" seat 4 u nt?	<start> Do you want me to reserve seat for you...	Do you want me to reserve seat for you or not?...
1	Yup. U reaching. We order some durian pastry a...	<start> Yeap. You reaching? We ordered some Du...	Yeap. You reaching? We ordered some Durian pas...
2	They become more ex oredi... Mine is like 25.....	<start> They become more expensive already. Mi...	They become more expensive already. Mine is li...
3	I'm thai. what do u do?	<start> I'm Thai. What do you do?	I'm Thai. What do you do? <end>

In [175]:

```
from sklearn.model_selection import train_test_split
train, validation = train_test_split(df, test_size=0.01)
```

In [176]:

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

```
(1970, 3) (20, 3)
```

In [177]:

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

In [178]:

```
vocab_size_target=len(tknizer_target.word_index.keys())
print(vocab_size_target)
vocab_size_source=len(tknizer_source.word_index.keys())
print(vocab_size_source)
```

```
3038
3706
```

In [180]:

```
decoder_embedding_matrix = np.zeros((vocab_size_target+1, 300))
for word, i in tknizer_target.word_index.items():
    embedding_vector = model.get(word)
    if embedding_vector is not None:
        decoder_embedding_matrix[i] = embedding_vector
```

In [182]:

```
encoder_embedding_matrix = np.zeros((vocab_size_source+1, 300))
for word, i in tknizer_source.word_index.items():
    embedding_vector = model.get(word)
    if embedding_vector is not None:
        encoder_embedding_matrix[i] = embedding_vector
```

In [185]:

```
print(decoder_embedding_matrix.shape)
```

```
(3039, 300)
```

In [186]:

```
print(encoder_embedding_matrix.shape)
```

```
(3707, 300)
```

In [187]:

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

Out[187]:

```
(1, 1445)
```

In [188]:

```
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):

        #Initialize Embedding layer
        #Intialize Encoder LSTM layer
```

```
        super().__init__()
        self.vocab_size = inp_vocab_size
        self.embedding_size = embedding_size
        self.input_length = input_length
        self.lstm_size= lstm_size
        self.lstm_output=0
        self.embedding = tf.keras.layers.Embedding(input_dim=self.vocab_size, output_dim=self.embedding_s
                           mask_zero=True,weights=[encoder_embedding_matrix],name="embedding_layer_encod
        self.lstm = tf.keras.layers.LSTM(self.lstm_size, return_state=True, return_sequences=True, name="


    def call(self,input_sequence,states):

      '''
          This function takes a sequence input and the initial states of the encoder.
          Pass the input_sequence input to the Embedding layer, Pass the embedding layer oupulstm_state_h
          returns -- All encoder_outputs, last time steps hidden and cell state
      '''

      input_embedd                              = self.embedding(input_sequence)
      lstm_state_h,lstm_state_c= states[0],states[1]
      self.lstm_output,lstm_state_h,lstm_state_c=self.lstm(input_embedd)
      return self.lstm_output,lstm_state_h,lstm_state_c



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

In [189]:

```
class Attention(tf.keras.layers.Layer):
  '''
    Class the calculates score based on the scoring_function using Bahdanu attention mechanism.
  '''
  def __init__(self,scoring_function, att_units):
    super().__init__()
    self.scoring_function=scoring_function
    # Please go through the reference notebook and research paper to complete the scoring functions

    if self.scoring_function=='dot':
      # Intialize variables needed for Dot score function here
      pass
    if self.scoring_function == 'general':
      # Intialize variables needed for General score function here
      self.weight=tf.keras.layers.Dense(att_units)
    elif self.scoring_function == 'concat':
      # Intialize variables needed for Concat score function here
      self.weight1=tf.keras.layers.Dense(att_units)
      self.weight2=tf.keras.layers.Dense(att_units)
      self.v=tf.keras.layers.Dense(1)


  def call(self,decoder_hidden_state,encoder_output):
    '''
      Attention mechanism takes two inputs current step -- decoder_hidden_state and all the encoder_outpu
      * Based on the scoring function we will find the score or similarity between decoder_hidden_state a
        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
        decoder_hidden_state=tf.expand_dims(decoder_hidden_state,axis=2)
        value=tf.matmul(encoder_output,decoder_hidden_state)
    elif self.scoring_function == 'general':
        # Implement General score function here
        decoder_hidden_state=tf.expand_dims(decoder_hidden_state,axis=2)
        value=tf.matmul(self.weight(encoder_output),decoder_hidden_state)
    elif self.scoring_function == 'concat':
        # Implement General score function here
        decoder_hidden_state=tf.expand_dims(decoder_hidden_state,axis=1)
        value=self.v(tf.nn.tanh(self.weight1(decoder_hidden_state)+self.weight2(encoder_output)))
```

```
    attention_weights=tf.nn.softmax(value,axis=1)
    context_vector=attention_weights*encoder_output
    return tf.reduce_sum(context_vector,axis=1),attention_weights
```

In [190]:

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

      # Initialize decoder embedding layer, LSTM and any other objects needed
      super().__init__()
      self.tar_vocab_size = tar_vocab_size
      self.embedding_dim = embedding_dim
      self.input_length = input_length
      self.dec_units = dec_units
      self.score_fun = score_fun
      self.att_units = att_units
      # we are using embedding_matrix and not training the embedding layer
      self.embedding = tf.keras.layers.Embedding(input_dim=self.tar_vocab_size, output_dim=self.embedding
                           mask_zero=True,weights=[decoder_embedding_matrix],name="embedding_layer_decod
      self.lstm = tf.keras.layers.LSTM(self.dec_units, return_sequences=True, return_state=True)
      self.dense = tf.keras.layers.Dense(self.tar_vocab_size)
      self.attention = Attention(self.score_fun,self.att_units)

  def call(self,input_to_decoder, encoder_output, state_h,state_c):
    '''
        One step decoder mechanisim step by step:
      A. Pass the input_to_decoder to the embedding layer and then get the output(batch_size,1,embedding_
      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
    '''
    output = self.embedding(input_to_decoder)
    context_vector,attention_weights = self.attention(state_h,encoder_output)
    context_vector1 = tf.expand_dims(context_vector,1)
    concat = tf.concat([output,context_vector1],axis=-1)
    decoder_output,state_h,state_c = self.lstm(concat,initial_state=[state_h,state_c])
    final_output = self.dense(decoder_output)
    final_output = tf.reshape(final_output,(-1,final_output.shape[2]))
    return final_output,state_h,state_c,attention_weights,context_vector
```

In [191]:

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

    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
        all_outputs=tf.TensorArray(tf.float32,size=input_to_decoder.shape[1])
        #Create a tensor array as shown in the reference notebook
        #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_decoder[

            # Store the output in tensorarray
            all_outputs=all_outputs.write(timestep,output)
        all_outputs=tf.transpose(all_outputs.stack(),[1,0,2])
        # Return the tensor array
        return all_outputs
```

In [198]:

```
class encoder_decoder(tf.keras.Model):
  def __init__(self,encoder_inputs_length,decoder_inputs_length, output_vocab_size,batch_size,score_fun):
    #Intialize objects from encoder decoder
    super().__init__() # https://stackoverflow.com/a/27134600/4084039
    self.batch_size=batch_size
    self.encoder = Encoder(vocab_size_source+1,300,100,encoder_inputs_length)
    self.decoder = Decoder(vocab_size_target+1,300,decoder_inputs_length,100,score_fun,100)

  def call(self,data):
    #Intialize encoder states, Pass the encoder_sequence to the embedding layer
    # Decoder initial states are encoder final states, Initialize it accordingly
    # Pass the decoder sequence,encoder_output,decoder states to Decoder
    # return the decoder output
    input,output = data[0], data[1]
    initial_state=self.encoder.initialize_states(self.batch_size)
    encoder_output, encoder_h, encoder_c = self.encoder(input,initial_state)
    decoder_output= self.decoder(output, encoder_output, encoder_h, encoder_c)
    return decoder_output
```

In [199]:

```
#https://www.tensorflow.org/tutorials/text/image_captioning#model
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')
def loss_function(real, pred):
    """ Custom loss function that will not consider the loss for padded zeros.
    why are we using this, can't we use simple sparse categorical crossentropy?
    Yes, you can use simple sparse categorical crossentropy as loss like we did in task-1. But in this lo
    for the padded zeros. i.e when the input is zero then we donot need to worry what the output is. This
    during preprocessing to make equal length for all the sentences.

    """


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

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

    return tf.reduce_mean(loss_)
```

In [200]:

```
class Dataset:
    def __init__(self, df, tknizer_source, tknizer_target, source_len,target_len):
        self.encoder_inps = df['source'].values
        self.decoder_inps = df['target_in'].values
        self.decoder_outs = df['target_out'].values
        self.tknizer_target = tknizer_target
        self.tknizer_source = tknizer_source
        self.source_len = source_len
        self.target_len = target_len


    def __getitem__(self, i):
        self.encoder_seq = self.tknizer_source.texts_to_sequences([self.encoder_inps[i]]) # need to pass
        self.decoder_inp_seq = self.tknizer_target.texts_to_sequences([self.decoder_inps[i]])
        self.decoder_out_seq = self.tknizer_target.texts_to_sequences([self.decoder_outs[i]])

        self.encoder_seq = pad_sequences(self.encoder_seq, maxlen=self.source_len, dtype='int32', padding
        self.decoder_inp_seq = pad_sequences(self.decoder_inp_seq, maxlen=self.target_len, dtype='int32',
        self.decoder_out_seq = pad_sequences(self.decoder_out_seq, maxlen=self.target_len, dtype='int32',
        return self.encoder_seq, self.decoder_inp_seq, self.decoder_out_seq

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


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


    def __getitem__(self, i):
        start = i * self.batch_size
        stop = (i + 1) * self.batch_size
```

```
        data = []
        for j in range(start, stop):
            data.append(self.dataset[j])

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

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

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

In [201]:

```
train_dataset = Dataset(train, tknizer_source, tknizer_target,39,43)
test_dataset  = Dataset(validation, tknizer_source, tknizer_target,39,43)

train_dataloader = Dataloder(train_dataset, batch_size=512)
test_dataloader = Dataloder(test_dataset, batch_size=20)


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

```
(512, 39) (512, 43) (512, 43)
```

In [202]:

```
tf.config.experimental_run_functions_eagerly(True)
```

In [203]:

```
tf.config.run_functions_eagerly(True)
```

In [205]:

```
#Create an object of encoder_decoder Model class,
# Compile the model and fit the model
# 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.
model  = encoder_decoder(encoder_inputs_length=39,decoder_inputs_length=43,output_vocab_size=vocab_size_t
optimizer = tf.keras.optimizers.Adam(0.1)
model.compile(optimizer=optimizer,loss=loss_function)
train_steps=train.shape[0]//512
valid_steps=validation.shape[0]//20
model.fit_generator(train_dataloader, steps_per_epoch=train_steps, epochs=100, validation_data=test_datal
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1940: UserWarning:
`Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`,
which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/dataset_ops.py:3704: UserWarning: Even
though the `tf.config.experimental_run_functions_eagerly` option is set, this option does not apply to
tf.data functions. To force eager execution of tf.data functions, please use
`tf.data.experimental.enable.debug_mode()`.
  "Even though the `tf.config.experimental_run_functions_eagerly` "
Epoch 1/100
3/3 [==============================] - 2s 727ms/step - loss: 2.6743 - val_loss: 2.8001
Epoch 2/100
3/3 [==============================] - 2s 718ms/step - loss: 3.1049 - val_loss: 2.6459
Epoch 3/100
3/3 [==============================] - 2s 701ms/step - loss: 2.8063 - val_loss: 2.4101
Epoch 4/100
3/3 [==============================] - 2s 719ms/step - loss: 2.6555 - val_loss: 2.3383
Epoch 5/100
3/3 [==============================] - 2s 712ms/step - loss: 2.5396 - val_loss: 2.2771
Epoch 6/100
3/3 [==============================] - 2s 777ms/step - loss: 2.4468 - val_loss: 2.1897
Epoch 7/100
3/3 [==============================] - 2s 715ms/step - loss: 2.3849 - val_loss: 2.0806
Epoch 8/100
3/3 [==============================] - 2s 709ms/step - loss: 2.2844 - val_loss: 2.0817
Epoch 9/100
```

```
Epoch 9/100
3/3 [==============================] - 2s 744ms/step - loss: 2.2685 - val_loss: 2.0791
Epoch 10/100
3/3 [==============================] - 2s 716ms/step - loss: 2.2113 - val_loss: 2.0348
Epoch 11/100
3/3 [==============================] - 2s 718ms/step - loss: 2.2045 - val_loss: 2.0618
Epoch 12/100
3/3 [==============================] - 2s 717ms/step - loss: 2.1675 - val_loss: 2.0392
Epoch 13/100
3/3 [==============================] - 2s 710ms/step - loss: 2.1733 - val_loss: 2.0068
Epoch 14/100
3/3 [==============================] - 2s 714ms/step - loss: 2.1584 - val_loss: 2.0042
Epoch 15/100
3/3 [==============================] - 2s 763ms/step - loss: 2.1371 - val_loss: 1.9930
Epoch 16/100
3/3 [==============================] - 2s 718ms/step - loss: 2.1129 - val_loss: 2.0101
Epoch 17/100
3/3 [==============================] - 2s 710ms/step - loss: 2.1117 - val_loss: 2.0135
Epoch 18/100
3/3 [==============================] - 2s 703ms/step - loss: 2.1219 - val_loss: 1.9791
Epoch 19/100
3/3 [==============================] - 2s 719ms/step - loss: 2.1001 - val_loss: 1.9775
Epoch 20/100
3/3 [==============================] - 2s 709ms/step - loss: 2.1127 - val_loss: 1.9801
Epoch 21/100
3/3 [==============================] - 2s 721ms/step - loss: 2.1343 - val_loss: 2.0149
Epoch 22/100
3/3 [==============================] - 2s 707ms/step - loss: 2.1217 - val_loss: 1.9798
Epoch 23/100
3/3 [==============================] - 2s 720ms/step - loss: 2.1119 - val_loss: 1.9931
Epoch 24/100
3/3 [==============================] - 2s 726ms/step - loss: 2.1105 - val_loss: 1.9835
Epoch 25/100
3/3 [==============================] - 2s 706ms/step - loss: 2.1120 - val_loss: 1.9812
Epoch 26/100
3/3 [==============================] - 2s 707ms/step - loss: 2.1272 - val_loss: 2.0150
Epoch 27/100
3/3 [==============================] - 2s 709ms/step - loss: 2.1338 - val_loss: 1.9878
Epoch 28/100
3/3 [==============================] - 2s 708ms/step - loss: 2.1208 - val_loss: 2.0014
Epoch 29/100
3/3 [==============================] - 2s 708ms/step - loss: 2.1082 - val_loss: 1.9929
Epoch 30/100
3/3 [==============================] - 2s 702ms/step - loss: 2.1146 - val_loss: 1.9973
Epoch 31/100
3/3 [==============================] - 2s 703ms/step - loss: 2.1049 - val_loss: 2.0026
Epoch 32/100
3/3 [==============================] - 2s 703ms/step - loss: 2.1250 - val_loss: 1.9729
Epoch 33/100
3/3 [==============================] - 2s 715ms/step - loss: 2.1311 - val_loss: 1.9957
Epoch 34/100
3/3 [==============================] - 2s 706ms/step - loss: 2.1171 - val_loss: 1.9856
Epoch 35/100
3/3 [==============================] - 2s 705ms/step - loss: 2.1256 - val_loss: 2.0031
Epoch 36/100
3/3 [==============================] - 2s 706ms/step - loss: 2.1054 - val_loss: 1.9840
Epoch 37/100
3/3 [==============================] - 2s 712ms/step - loss: 2.0978 - val_loss: 1.9751
Epoch 38/100
3/3 [==============================] - 2s 772ms/step - loss: 2.0975 - val_loss: 1.9841
Epoch 39/100
3/3 [==============================] - 2s 731ms/step - loss: 2.0863 - val_loss: 1.9830
Epoch 40/100
3/3 [==============================] - 2s 745ms/step - loss: 2.1015 - val_loss: 1.9927
Epoch 41/100
3/3 [==============================] - 2s 709ms/step - loss: 2.1103 - val_loss: 1.9815
Epoch 42/100
3/3 [==============================] - 2s 716ms/step - loss: 2.1009 - val_loss: 1.9851
Epoch 43/100
3/3 [==============================] - 2s 707ms/step - loss: 2.0853 - val_loss: 1.9844
Epoch 44/100
3/3 [==============================] - 2s 712ms/step - loss: 2.1094 - val_loss: 1.9984
Epoch 45/100
3/3 [==============================] - 2s 703ms/step - loss: 2.1050 - val_loss: 1.9882
Epoch 46/100
3/3 [==============================] - 2s 702ms/step - loss: 2.1141 - val_loss: 1.9824
Epoch 47/100
3/3 [==============================] - 2s 708ms/step - loss: 2.1094 - val_loss: 2.0025
```

3/3 [==============================] - 2s 708ms/step - loss: 2.1094 - val_loss: 2.0025
Epoch 48/100
3/3 [==============================] - 2s 715ms/step - loss: 2.1121 - val_loss: 1.9829
Epoch 49/100
3/3 [==============================] - 2s 791ms/step - loss: 2.0847 - val_loss: 1.9925
Epoch 50/100
3/3 [==============================] - 2s 711ms/step - loss: 2.0908 - val_loss: 1.9837
Epoch 51/100
3/3 [==============================] - 2s 718ms/step - loss: 2.0827 - val_loss: 1.9892
Epoch 52/100
3/3 [==============================] - 2s 706ms/step - loss: 2.0754 - val_loss: 1.9712
Epoch 53/100
3/3 [==============================] - 2s 721ms/step - loss: 2.1039 - val_loss: 1.9608
Epoch 54/100
3/3 [==============================] - 2s 715ms/step - loss: 2.0708 - val_loss: 2.0018
Epoch 55/100
3/3 [==============================] - 2s 709ms/step - loss: 2.1111 - val_loss: 2.0088
Epoch 56/100
3/3 [==============================] - 2s 715ms/step - loss: 2.0874 - val_loss: 1.9793
Epoch 57/100
3/3 [==============================] - 2s 705ms/step - loss: 2.1122 - val_loss: 1.9807
Epoch 58/100
3/3 [==============================] - 2s 715ms/step - loss: 2.1115 - val_loss: 2.0028
Epoch 59/100
3/3 [==============================] - 2s 706ms/step - loss: 2.1093 - val_loss: 1.9878
Epoch 60/100
3/3 [==============================] - 2s 785ms/step - loss: 2.1056 - val_loss: 1.9987
Epoch 61/100
3/3 [==============================] - 2s 705ms/step - loss: 2.1165 - val_loss: 1.9693
Epoch 62/100
3/3 [==============================] - 2s 716ms/step - loss: 2.1020 - val_loss: 1.9992
Epoch 63/100
3/3 [==============================] - 2s 715ms/step - loss: 2.0814 - val_loss: 1.9966
Epoch 64/100
3/3 [==============================] - 2s 715ms/step - loss: 2.1038 - val_loss: 1.9744
Epoch 65/100
3/3 [==============================] - 2s 702ms/step - loss: 2.1183 - val_loss: 2.0225
Epoch 66/100
3/3 [==============================] - 2s 717ms/step - loss: 2.1104 - val_loss: 1.9974
Epoch 67/100
3/3 [==============================] - 2s 714ms/step - loss: 2.1058 - val_loss: 1.9886
Epoch 68/100
3/3 [==============================] - 2s 719ms/step - loss: 2.1088 - val_loss: 2.0191
Epoch 69/100
3/3 [==============================] - 2s 714ms/step - loss: 2.0988 - val_loss: 1.9854
Epoch 70/100
3/3 [==============================] - 2s 782ms/step - loss: 2.1329 - val_loss: 2.0273
Epoch 71/100
3/3 [==============================] - 2s 743ms/step - loss: 2.1320 - val_loss: 1.9902
Epoch 72/100
3/3 [==============================] - 2s 722ms/step - loss: 2.0959 - val_loss: 2.0073
Epoch 73/100
3/3 [==============================] - 2s 718ms/step - loss: 2.0861 - val_loss: 2.0226
Epoch 74/100
3/3 [==============================] - 2s 721ms/step - loss: 2.0886 - val_loss: 1.9666
Epoch 75/100
3/3 [==============================] - 2s 734ms/step - loss: 2.0987 - val_loss: 1.9815
Epoch 76/100
3/3 [==============================] - 2s 708ms/step - loss: 2.1030 - val_loss: 1.9998
Epoch 77/100
3/3 [==============================] - 2s 708ms/step - loss: 2.1003 - val_loss: 1.9984
Epoch 78/100
3/3 [==============================] - 2s 709ms/step - loss: 2.0706 - val_loss: 1.9761
Epoch 79/100
3/3 [==============================] - 2s 719ms/step - loss: 2.1079 - val_loss: 1.9823
Epoch 80/100
3/3 [==============================] - 2s 712ms/step - loss: 2.0910 - val_loss: 1.9905
Epoch 81/100
3/3 [==============================] - 2s 790ms/step - loss: 2.0991 - val_loss: 2.0074
Epoch 82/100
3/3 [==============================] - 2s 725ms/step - loss: 2.0768 - val_loss: 1.9801
Epoch 83/100
3/3 [==============================] - 2s 711ms/step - loss: 2.1060 - val_loss: 1.9789
Epoch 84/100
3/3 [==============================] - 2s 710ms/step - loss: 2.0763 - val_loss: 1.9917
Epoch 85/100
3/3 [==============================] - 2s 708ms/step - loss: 2.0920 - val_loss: 1.9878
Epoch 86/100

```
Epoch 86/100
3/3 [==============================] - 2s 718ms/step - loss: 2.0883 - val_loss: 2.0033
Epoch 87/100
3/3 [==============================] - 2s 702ms/step - loss: 2.0801 - val_loss: 1.9853
Epoch 88/100
3/3 [==============================] - 2s 713ms/step - loss: 2.0882 - val_loss: 1.9630
Epoch 89/100
3/3 [==============================] - 2s 731ms/step - loss: 2.0831 - val_loss: 1.9978
Epoch 90/100
3/3 [==============================] - 2s 724ms/step - loss: 2.0773 - val_loss: 1.9906
Epoch 91/100
3/3 [==============================] - 2s 712ms/step - loss: 2.0675 - val_loss: 1.9766
Epoch 92/100
3/3 [==============================] - 2s 704ms/step - loss: 2.0673 - val_loss: 1.9759
Epoch 93/100
3/3 [==============================] - 2s 767ms/step - loss: 2.0663 - val_loss: 1.9672
Epoch 94/100
3/3 [==============================] - 2s 711ms/step - loss: 2.0774 - val_loss: 1.9735
Epoch 95/100
3/3 [==============================] - 2s 730ms/step - loss: 2.0841 - val_loss: 1.9904
Epoch 96/100
3/3 [==============================] - 2s 714ms/step - loss: 2.0921 - val_loss: 1.9782
Epoch 97/100
3/3 [==============================] - 2s 705ms/step - loss: 2.0842 - val_loss: 1.9968
Epoch 98/100
3/3 [==============================] - 2s 711ms/step - loss: 2.1122 - val_loss: 1.9860
Epoch 99/100
3/3 [==============================] - 2s 738ms/step - loss: 2.1077 - val_loss: 1.9917
Epoch 100/100
3/3 [==============================] - 2s 714ms/step - loss: 2.0866 - val_loss: 2.0040
```

Out[205]:

```
<tensorflow.python.keras.callbacks.History at 0x7ff74b6e5c50>
```

In [112]:

```
batch_size=512
units=100
```

In [113]:

```
def predict(input_sentence):

  '''
  A. Given input sentence, convert the sentence into integers using tokenizer used earlier
  B. Pass the input_sequence to encoder. we get encoder_outputs, last time step hidden and cell state
  C. Initialize index of <start> as input to decoder. and encoder final states as input_states to onestep
  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,
        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
  '''
  initial_state_enc=[np.zeros((batch_size,units)),np.zeros((batch_size,units))]
  inp_seq = tknizer_source.texts_to_sequences([input_sentence])
  inp_seq = pad_sequences(inp_seq,padding='post',maxlen=39)

  en_outputs,state_h , state_c = model.layers[0](tf.constant(inp_seq),initial_state_enc)
  cur_vec = tf.constant([[tknizer_target.word_index['<start>']]])
  pred = []
  #Here 43 is the max_length of the sequence
  for i in range(43):
    output,state_h,state_c,attention_weights,context_vector = model.layers[1].onestepdecoder(cur_vec,en_o
    cur_vec = np.reshape(np.argmax(output), (1, 1))
    pred.append(tknizer_target.index_word[cur_vec[0][0]])
    if(pred[-1]=='<end>'):
      break
    translated_sentence = ' '.join(pred)
  return translated_sentence
```

In [114]:

```
validation['target_in']
```

Out[114]:

```
1279    <start> At some coffee shop behind a building....
1372    <start> Shuhui in Ang Mo Kio, she asks if want...
796     <start> I'm not working. What time is Junmei a...
1230    <start> Told you to go to Bugis already. Very ...
1946    <start> Oh, that guy who is much taller than m...
661     <start> Joan never replied me. Call her but sh...
1546    <start> Hi, never worry about the truth becaus...
129                            <start> Saturday. Can?
1668                  <start> Huh? How come, too taxing?
325     <start> I'm going for lecture later. So pick m...
1441                          <start> Just left office.
1195    <start> Your chauffeur? Hahaha, who is it? Fro...
460     <start> You looking for June? Came back must P...
1804    <start> That pest's father's handphone. Then y...
771     <start> So sad. I bought the opera bar without...
1109    <start> Yay! I am taking ST and LSM this term....
534     <start> HKY, I remember I have to give you $30...
929        <start> Ah, I'm in exam period. Ah, I'm dying.
134           <start> Wu Jian Dao got sneaks? I anything.
1236    <start> Yes. By the way, I'll be buying the pr...
Name: target_in, dtype: object
```

In [116]:

```
for i in validation['source']:
  predicted=predict(i)
  print("The predicted output is: ",predicted)
```

```
The predicted output is:  hey i was accurate club a check mrt movie question is dividend
The predicted output is:  i see you know ok then i am on thursday mummy stuff which 225
The predicted output is:  i am accurate club getting mrt station
The predicted output is:  hey i need to go
The predicted output is:  hey i think they kill me to go project you know ok i know ok i know ok
The predicted output is:  i am not not not not not not not not not not not not not not not not not not not
not not not not not not not not not not not not not not not not not not not not not not not not not not not
The predicted output is:  i was nus 30pm
The predicted output is:  then i am accurate put you get here
The predicted output is:  what you know ok
The predicted output is:  i meet tomorrow
The predicted output is:  we are you know ok i need to go to go to go time off
The predicted output is:  hey i was now
The predicted output is:  hey i was happy number
The predicted output is:  i am on thursday on on thursday yet see you have check
The predicted output is:  weight work not having put done you know ok
The predicted output is:  you are you know ok i need to go
The predicted output is:  hey i was a exercise don't worry
The predicted output is:  hey you are you know how have you know how have you know how have you know mrt
whole days either mrt outing because it got study hard in 25 you come week tell you know ok i need to go
to go
The predicted output is:  hey you want to go
The predicted output is:  i got first to go for a bit 40 happy interview you colour mrt station
```

In [ ]:

```
# Predict on 1000 random sentences on test data and calculate the average BLEU score of these sentences.
import nltk.translate.bleu_score as bleu
bleu_scores_lst=[]
for i in validation[:]['source']:
  reference = [i.split(),] # the original
  predicted=predict([i])
  translation = predicted.split()
  values=bleu.sentence_bleu(reference, translation)
  bleu_scores_lst.append(values)

# https://www.nltk.org/_modules/nltk/translate/bleu_score.html
```

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

In [ ]:

```
average_bleu_scores=sum(bleu_scores_lst)/len(bleu_scores_lst)
print("Average BLEU score of these 1000 test data sentences is: ",average_bleu_scores)
```

```
Average BLEU score of these 1000 test data sentences is:  0.03251957333271974
```

In [ ]:

```
bleu_scores_lst
```

Out[ ]:

```
[0,
 0,
 0,
 0,
 0,
 0,
 0,
 0.46230595512422085,
 0,
 0,
 0,
 0,
 0.03589763288144409,
 0.15218787864872976,
 0,
 0,
 0,
 0,
 0,
 0]
```

Character_Level:

In [138]:

```
df=pd.read_csv('preprocessed_data.csv')
df.head()
```

Out[138]:

	Unnamed: 0	source	target
0	0	U wan me to "chop" seat 4 u nt?\n	Do you want me to reserve seat for you or not?\n
1	1	Yup. U reaching. We order some durian pastry a...	Yeap. You reaching? We ordered some Durian pas...
2	2	They become more ex oredi... Mine is like 25.....	They become more expensive already. Mine is li...
3	3	I'm thai. what do u do?\n	I'm Thai. What do you do?\n
4	4	Hi! How did your week go? Haven heard from you...	Hi! How did your week go? Haven't heard from y...

In [139]:

```
def preprocess(x):
  x=x[:-1]
  return x
```

In [140]:

```
df['source']=df['source'].apply(preprocess)
df['target']=df['target'].apply(preprocess)
```

In [141]:

```
df=df[['source','target']]
df.head()
```

Out[141]:

	source	target
0	U wan me to "chop" seat 4 u nt?	Do you want me to reserve seat for you or not?
1	Yup. U reaching. We order some durian pastry a...	Yeap. You reaching? We ordered some Durian pas...
2	They become more ex oredi... Mine is like 25.....	They become more expensive already. Mine is li...
3	I'm thai. what do u do?	I'm Thai. What do you do?
4	Hi! How did your week go? Haven heard from you...	Hi! How did your week go? Haven't heard from y...

In [142]:

```
df.shape
```

Out[142]:

```
(2000, 2)
```

In [143]:

```
def length(text):#for calculating the length of the sentence
    return len(str(text))
```

In [144]:

```
df=df[df['source'].apply(length)<=170]
```

```
df=df[df['target'].apply(length)<=200]
```

In [145]:

```
df.shape
```

Out[145]:

```
(1993, 2)
```

In [146]:

```
df['target_in'] = '\t' + df['target'].astype(str)
df['target_out'] = df['target'].astype(str) + '\n'
# only for the first sentance add a toke <end> so that we will have <end> in tokenizer
df.head()
```

Out[146]:

	source	target	target_in	target_out
0	U wan me to "chop" seat 4 u nt?	Do you want me to reserve seat for you or not?	\tDo you want me to reserve seat for you or not?	Do you want me to reserve seat for you or not?\n
1	Yup. U reaching. We order some durian pastry a...	Yeap. You reaching? We ordered some Durian pas...	\tYeap. You reaching? We ordered some Durian p...	Yeap. You reaching? We ordered some Durian pas...
2	They become more ex oredi... Mine is like 25.....	They become more expensive already. Mine is li...	\tThey become more expensive already. Mine is ...	They become more expensive already. Mine is li...
3	I'm thai. what do u do?	I'm Thai. What do you do?	\tI'm Thai. What do you do?	I'm Thai. What do you do?\n
4	Hi! How did your week go? Haven heard from you...	Hi! How did your week go? Haven't heard from y...	\tHi! How did your week go? Haven't heard from...	Hi! How did your week go? Haven't heard from y...

In [147]:

```
df=df.drop('target',axis=1)
```

In [148]:

```
df.head(4)
```

Out[148]:

	source	target_in	target_out
0	U wan me to "chop" seat 4 u nt?	\tDo you want me to reserve seat for you or not?	Do you want me to reserve seat for you or not?\n
1	Yup. U reaching. We order some durian pastry a...	\tYeap. You reaching? We ordered some Durian p...	Yeap. You reaching? We ordered some Durian pas...
2	They become more ex oredi... Mine is like 25.....	\tThey become more expensive already. Mine is ...	They become more expensive already. Mine is li...
3	I'm thai. what do u do?	\tI'm Thai. What do you do?	I'm Thai. What do you do?\n

In [149]:

```
from sklearn.model_selection import train_test_split
train, validation = train_test_split(df, test_size=0.01)
```

In [150]:

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

```
(1973, 3) (20, 3)
```

In [151]:

```
tknizer_source = Tokenizer(filters=None,char_level=True,lower=False)
tknizer_source.fit_on_texts(train['source'].values)
tknizer_target = Tokenizer(filters=None,char_level=True,lower=False)
tknizer_target.fit_on_texts(train['target_in'].values)
```

In [152]:

```
vocab_size_target=len(tknizer_target.word_index.keys())
print(vocab_size_target)
vocab_size_source=len(tknizer_source.word_index.keys())
print(vocab_size_source)
```

```
92
103
```

In [47]:

```
tknizer_target.word_index['\t'], tknizer_target.word_index['\n']
```

Out[47]:

```
(20, 85)
```

In [48]:

```
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):

        #Initialize Embedding layer
        #Intialize Encoder LSTM layer
        super().__init__()
        self.vocab_size = inp_vocab_size
        self.embedding_size = embedding_size
        self.input_length = input_length
        self.lstm_size= lstm_size
        self.lstm_output=0
        self.embedding = tf.keras.layers.Embedding(input_dim=self.vocab_size, output_dim=self.embedding_s
                           mask_zero=True, name="embedding_layer_encoder")
        self.lstm = tf.keras.layers.LSTM(self.lstm_size, return_state=True, return_sequences=True, name="


    def call(self,input_sequence,states):

      '''
          This function takes a sequence input and the initial states of the encoder.
          Pass the input_sequence input to the Embedding layer, Pass the embedding layer oupulstm_state_h
          returns -- All encoder_outputs, last time steps hidden and cell state
      '''

      input_embedd                              = self.embedding(input_sequence)
      lstm_state_h,lstm_state_c= states[0],states[1]
      self.lstm_output,lstm_state_h,lstm_state_c=self.lstm(input_embedd)
      return self.lstm_output,lstm_state_h,lstm_state_c



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

In [49]:

```
class Attention(tf.keras.layers.Layer):
  '''
    Class the calculates score based on the scoring_function using Bahdanu attention mechanism.
  '''
  def __init__(self,scoring_function, att_units):
    super().__init__()
    self.scoring_function=scoring_function
    # Please go through the reference notebook and research paper to complete the scoring functions

    if self.scoring_function=='dot':
      # Intialize variables needed for Dot score function here
      pass
    if self.scoring_function == 'general':
      # Intialize variables needed for General score function here
      self.weight=tf.keras.layers.Dense(att_units)
    elif self.scoring_function == 'concat':
      # Intialize variables needed for Concat score function here
      self.weight1=tf.keras.layers.Dense(att_units)
      self.weight2=tf.keras.layers.Dense(att_units)
      self.v=tf.keras.layers.Dense(1)


  def call(self,decoder_hidden_state,encoder_output):
    '''
      Attention mechanism takes two inputs current step -- decoder_hidden_state and all the encoder_outpu
      * Based on the scoring function we will find the score or similarity between decoder_hidden_state a
        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
        decoder_hidden_state=tf.expand_dims(decoder_hidden_state,axis=2)
        value=tf.matmul(encoder_output,decoder_hidden_state)
```

```
    elif self.scoring_function == 'general':
        # Implement General score function here
        decoder_hidden_state=tf.expand_dims(decoder_hidden_state,axis=2)
        value=tf.matmul(self.weight(encoder_output),decoder_hidden_state)
    elif self.scoring_function == 'concat':
        # Implement General score function here
        decoder_hidden_state=tf.expand_dims(decoder_hidden_state,axis=1)
        value=self.v(tf.nn.tanh(self.weight1(decoder_hidden_state)+self.weight2(encoder_output)))
    attention_weights=tf.nn.softmax(value,axis=1)
    context_vector=attention_weights*encoder_output
    return tf.reduce_sum(context_vector,axis=1),attention_weights
```

In [50]:

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

      # Initialize decoder embedding layer, LSTM and any other objects needed
      super().__init__()
      self.tar_vocab_size = tar_vocab_size
      self.embedding_dim = embedding_dim
      self.input_length = input_length
      self.dec_units = dec_units
      self.score_fun = score_fun
      self.att_units = att_units
      # we are using embedding_matrix and not training the embedding layer
      self.embedding = tf.keras.layers.Embedding(input_dim=self.tar_vocab_size, output_dim=self.embedding
                          mask_zero=True, name="embedding_layer_decoder", trainable=True)
      self.lstm = tf.keras.layers.LSTM(self.dec_units, return_sequences=True, return_state=True)
      self.dense = tf.keras.layers.Dense(self.tar_vocab_size)
      self.attention = Attention(self.score_fun,self.att_units)

  def call(self,input_to_decoder, encoder_output, state_h,state_c):
    '''
        One step decoder mechanisim step by step:
      A. Pass the input_to_decoder to the embedding layer and then get the output(batch_size,1,embedding_
      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
    '''
    output = self.embedding(input_to_decoder)
    context_vector,attention_weights = self.attention(state_h,encoder_output)
    context_vector1 = tf.expand_dims(context_vector,1)
    concat = tf.concat([output,context_vector1],axis=-1)
    decoder_output,state_h,state_c = self.lstm(concat,initial_state=[state_h,state_c])
    final_output = self.dense(decoder_output)
    final_output = tf.reshape(final_output,(-1,final_output.shape[2]))
    return final_output,state_h,state_c,attention_weights,context_vector
```

In [51]:

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

    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
        all_outputs=tf.TensorArray(tf.float32,size=tf.shape(input_to_decoder)[1])
        #Create a tensor array as shown in the reference notebook
        #Iterate till the length of the decoder input
        for timestep in range(tf.shape(input_to_decoder)[1]):
            # Call onestepdecoder for each token in decoder_input
            output,state_h,state_c,attention_weights,context_vector=self.onestepdecoder(input_to_decoder[

            # Store the output in tensorarray
```

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

In [54]:

```
class encoder_decoder(tf.keras.Model):
  def __init__(self,encoder_inputs_length,decoder_inputs_length, output_vocab_size,batch_size,score_fun):
    #Intialize objects from encoder decoder
    super().__init__() # https://stackoverflow.com/a/27134600/4084039
    self.batch_size=batch_size
    self.encoder = Encoder(vocab_size_source+1,100,128,encoder_inputs_length)
    self.decoder = Decoder(vocab_size_target+1,100,decoder_inputs_length,128,score_fun,128)

  def call(self,data):
    #Intialize encoder states, Pass the encoder_sequence to the embedding layer
    # Decoder initial states are encoder final states, Initialize it accordingly
    # Pass the decoder sequence,encoder_output,decoder states to Decoder
    # return the decoder output
    input,output = data[0], data[1]
    initial_state=self.encoder.initialize_states(self.batch_size)
    encoder_output, encoder_h, encoder_c = self.encoder(input,initial_state)
    decoder_output= self.decoder(output, encoder_output, encoder_h, encoder_c)
    return decoder_output
```

In [55]:

```
#https://www.tensorflow.org/tutorials/text/image_captioning#model
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')
def loss_function(real, pred):
    """ Custom loss function that will not consider the loss for padded zeros.
    why are we using this, can't we use simple sparse categorical crossentropy?
    Yes, you can use simple sparse categorical crossentropy as loss like we did in task-1. But in this lo
    for the padded zeros. i.e when the input is zero then we donot need to worry what the output is. This
    during preprocessing to make equal length for all the sentences.

    """


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

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

    return tf.reduce_mean(loss_)
```

In [56]:

```
class Dataset:
    def __init__(self, df, tknizer_source, tknizer_target, source_len,target_len):
        self.encoder_inps = df['source'].values
        self.decoder_inps = df['target_in'].values
        self.decoder_outs = df['target_out'].values
        self.tknizer_target = tknizer_target
        self.tknizer_source = tknizer_source
        self.source_len = source_len
        self.target_len = target_len


    def __getitem__(self, i):
        self.encoder_seq = self.tknizer_source.texts_to_sequences([self.encoder_inps[i]]) # need to pass
        self.decoder_inp_seq = self.tknizer_target.texts_to_sequences([self.decoder_inps[i]])
        self.decoder_out_seq = self.tknizer_target.texts_to_sequences([self.decoder_outs[i]])

        self.encoder_seq = pad_sequences(self.encoder_seq, maxlen=self.source_len, dtype='int32', padding
        self.decoder_inp_seq = pad_sequences(self.decoder_inp_seq, maxlen=self.target_len, dtype='int32',
        self.decoder_out_seq = pad_sequences(self.decoder_out_seq, maxlen=self.target_len, dtype='int32',
        return self.encoder_seq, self.decoder_inp_seq, self.decoder_out_seq

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


class Dataloder(tf.keras.utils.Sequence):
    def __init__(self, dataset, batch_size=1):
```

```
        self.dataset = dataset
        self.batch_size = batch_size
        self.indexes = np.arange(len(self.dataset.encoder_inps))


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

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

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

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

In [57]:

```
train_dataset = Dataset(train, tknizer_source, tknizer_target,170,200)
test_dataset  = Dataset(validation, tknizer_source, tknizer_target,170,200)

train_dataloader = Dataloder(train_dataset, batch_size=512)
test_dataloader = Dataloder(test_dataset, batch_size=20)


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

```
(512, 170) (512, 200) (512, 200)
```

In [58]:

```
#Create an object of encoder_decoder Model class,
# Compile the model and fit the model
# 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.
model  = encoder_decoder(encoder_inputs_length=170,decoder_inputs_length=200,output_vocab_size=vocab_size
optimizer = tf.keras.optimizers.Adam(0.01)
model.compile(optimizer=optimizer,loss=loss_function)
train_steps=train.shape[0]//512
valid_steps=validation.shape[0]//20
model.fit_generator(train_dataloader, steps_per_epoch=train_steps, epochs=100, validation_data=test_datal
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1940: UserWarning:
`Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`,
which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/100
3/3 [==============================] - 16s 3s/step - loss: 1.5136 - val_loss: 1.5831
Epoch 2/100
3/3 [==============================] - 8s 3s/step - loss: 1.2368 - val_loss: 1.4793
Epoch 3/100
3/3 [==============================] - 8s 3s/step - loss: 1.1685 - val_loss: 1.4271
Epoch 4/100
3/3 [==============================] - 8s 3s/step - loss: 1.1407 - val_loss: 1.3892
Epoch 5/100
3/3 [==============================] - 8s 3s/step - loss: 1.1088 - val_loss: 1.3515
Epoch 6/100
3/3 [==============================] - 8s 3s/step - loss: 1.0753 - val_loss: 1.3076
Epoch 7/100
3/3 [==============================] - 8s 3s/step - loss: 1.0427 - val_loss: 1.2692
Epoch 8/100
3/3 [==============================] - 8s 3s/step - loss: 1.0122 - val_loss: 1.2346
Epoch 9/100
3/3 [==============================] - 8s 3s/step - loss: 0.9857 - val_loss: 1.2067
Epoch 10/100
```

```
3/3 [==============================] - 8s 3s/step - loss: 0.9659 - val_loss: 1.1834
Epoch 11/100
3/3 [==============================] - 10s 3s/step - loss: 0.9485 - val_loss: 1.1672
Epoch 12/100
3/3 [==============================] - 8s 3s/step - loss: 0.9367 - val_loss: 1.1538
Epoch 13/100
3/3 [==============================] - 8s 3s/step - loss: 0.9264 - val_loss: 1.1410
Epoch 14/100
3/3 [==============================] - 8s 3s/step - loss: 0.9184 - val_loss: 1.1300
Epoch 15/100
3/3 [==============================] - 8s 3s/step - loss: 0.9109 - val_loss: 1.1238
Epoch 16/100
3/3 [==============================] - 8s 3s/step - loss: 0.9045 - val_loss: 1.1198
Epoch 17/100
3/3 [==============================] - 8s 3s/step - loss: 0.8997 - val_loss: 1.1138
Epoch 18/100
3/3 [==============================] - 8s 3s/step - loss: 0.8954 - val_loss: 1.1088
Epoch 19/100
3/3 [==============================] - 8s 3s/step - loss: 0.8921 - val_loss: 1.1055
Epoch 20/100
3/3 [==============================] - 8s 3s/step - loss: 0.8892 - val_loss: 1.1024
Epoch 21/100
3/3 [==============================] - 8s 3s/step - loss: 0.8868 - val_loss: 1.0994
Epoch 22/100
3/3 [==============================] - 8s 3s/step - loss: 0.8845 - val_loss: 1.0968
Epoch 23/100
3/3 [==============================] - 8s 3s/step - loss: 0.8825 - val_loss: 1.0947
Epoch 24/100
3/3 [==============================] - 8s 3s/step - loss: 0.8809 - val_loss: 1.0930
Epoch 25/100
3/3 [==============================] - 8s 3s/step - loss: 0.8795 - val_loss: 1.0911
Epoch 26/100
3/3 [==============================] - 8s 3s/step - loss: 0.8781 - val_loss: 1.0899
Epoch 27/100
3/3 [==============================] - 8s 3s/step - loss: 0.8770 - val_loss: 1.0895
Epoch 28/100
3/3 [==============================] - 8s 3s/step - loss: 0.8757 - val_loss: 1.0883
Epoch 29/100
3/3 [==============================] - 8s 3s/step - loss: 0.8747 - val_loss: 1.0868
Epoch 30/100
3/3 [==============================] - 8s 3s/step - loss: 0.8736 - val_loss: 1.0864
Epoch 31/100
3/3 [==============================] - 8s 3s/step - loss: 0.8726 - val_loss: 1.0852
Epoch 32/100
3/3 [==============================] - 8s 3s/step - loss: 0.8717 - val_loss: 1.0840
Epoch 33/100
3/3 [==============================] - 8s 3s/step - loss: 0.8709 - val_loss: 1.0836
Epoch 34/100
3/3 [==============================] - 8s 3s/step - loss: 0.8698 - val_loss: 1.0823
Epoch 35/100
3/3 [==============================] - 8s 3s/step - loss: 0.8691 - val_loss: 1.0828
Epoch 36/100
3/3 [==============================] - 8s 3s/step - loss: 0.8682 - val_loss: 1.0818
Epoch 37/100
3/3 [==============================] - 8s 3s/step - loss: 0.8674 - val_loss: 1.0811
Epoch 38/100
3/3 [==============================] - 8s 3s/step - loss: 0.8664 - val_loss: 1.0801
Epoch 39/100
3/3 [==============================] - 8s 3s/step - loss: 0.8656 - val_loss: 1.0800
Epoch 40/100
3/3 [==============================] - 8s 3s/step - loss: 0.8651 - val_loss: 1.0794
Epoch 41/100
3/3 [==============================] - 8s 3s/step - loss: 0.8656 - val_loss: 1.0777
Epoch 42/100
3/3 [==============================] - 8s 3s/step - loss: 0.8660 - val_loss: 1.0786
Epoch 43/100
3/3 [==============================] - 8s 3s/step - loss: 0.8641 - val_loss: 1.0774
Epoch 44/100
3/3 [==============================] - 8s 3s/step - loss: 0.8622 - val_loss: 1.0762
Epoch 45/100
3/3 [==============================] - 8s 3s/step - loss: 0.8605 - val_loss: 1.0769
Epoch 46/100
3/3 [==============================] - 8s 3s/step - loss: 0.8602 - val_loss: 1.0758
Epoch 47/100
3/3 [==============================] - 8s 3s/step - loss: 0.8595 - val_loss: 1.0756
Epoch 48/100
3/3 [==============================] - 8s 3s/step - loss: 0.8585 - val_loss: 1.0755
```

```
Epoch 49/100
3/3 [==============================] - 8s 3s/step - loss: 0.8583 - val_loss: 1.0746
Epoch 50/100
3/3 [==============================] - 8s 3s/step - loss: 0.8571 - val_loss: 1.0738
Epoch 51/100
3/3 [==============================] - 8s 3s/step - loss: 0.8560 - val_loss: 1.0742
Epoch 52/100
3/3 [==============================] - 8s 3s/step - loss: 0.8548 - val_loss: 1.0746
Epoch 53/100
3/3 [==============================] - 8s 3s/step - loss: 0.8540 - val_loss: 1.0726
Epoch 54/100
3/3 [==============================] - 8s 3s/step - loss: 0.8528 - val_loss: 1.0733
Epoch 55/100
3/3 [==============================] - 8s 3s/step - loss: 0.8522 - val_loss: 1.0739
Epoch 56/100
3/3 [==============================] - 8s 3s/step - loss: 0.8515 - val_loss: 1.0744
Epoch 57/100
3/3 [==============================] - 8s 3s/step - loss: 0.8505 - val_loss: 1.0754
Epoch 58/100
3/3 [==============================] - 8s 3s/step - loss: 0.8502 - val_loss: 1.0737
Epoch 59/100
3/3 [==============================] - 8s 3s/step - loss: 0.8486 - val_loss: 1.0742
Epoch 60/100
3/3 [==============================] - 8s 3s/step - loss: 0.8474 - val_loss: 1.0754
Epoch 61/100
3/3 [==============================] - 8s 3s/step - loss: 0.8461 - val_loss: 1.0720
Epoch 62/100
3/3 [==============================] - 8s 3s/step - loss: 0.8448 - val_loss: 1.0728
Epoch 63/100
3/3 [==============================] - 8s 3s/step - loss: 0.8435 - val_loss: 1.0744
Epoch 64/100
3/3 [==============================] - 8s 3s/step - loss: 0.8442 - val_loss: 1.0754
Epoch 65/100
3/3 [==============================] - 8s 3s/step - loss: 0.8426 - val_loss: 1.0732
Epoch 66/100
3/3 [==============================] - 8s 3s/step - loss: 0.8413 - val_loss: 1.0717
Epoch 67/100
3/3 [==============================] - 8s 3s/step - loss: 0.8397 - val_loss: 1.0706
Epoch 68/100
3/3 [==============================] - 8s 3s/step - loss: 0.8387 - val_loss: 1.0734
Epoch 69/100
3/3 [==============================] - 8s 3s/step - loss: 0.8371 - val_loss: 1.0734
Epoch 70/100
3/3 [==============================] - 8s 3s/step - loss: 0.8358 - val_loss: 1.0698
Epoch 71/100
3/3 [==============================] - 8s 3s/step - loss: 0.8349 - val_loss: 1.0705
Epoch 72/100
3/3 [==============================] - 8s 3s/step - loss: 0.8332 - val_loss: 1.0714
Epoch 73/100
3/3 [==============================] - 8s 3s/step - loss: 0.8314 - val_loss: 1.0683
Epoch 74/100
3/3 [==============================] - 8s 3s/step - loss: 0.8296 - val_loss: 1.0681
Epoch 75/100
3/3 [==============================] - 8s 3s/step - loss: 0.8276 - val_loss: 1.0679
Epoch 76/100
3/3 [==============================] - 8s 3s/step - loss: 0.8263 - val_loss: 1.0668
Epoch 77/100
3/3 [==============================] - 8s 3s/step - loss: 0.8245 - val_loss: 1.0675
Epoch 78/100
3/3 [==============================] - 8s 3s/step - loss: 0.8228 - val_loss: 1.0682
Epoch 79/100
3/3 [==============================] - 8s 3s/step - loss: 0.8209 - val_loss: 1.0658
Epoch 80/100
3/3 [==============================] - 8s 3s/step - loss: 0.8190 - val_loss: 1.0655
Epoch 81/100
3/3 [==============================] - 8s 3s/step - loss: 0.8178 - val_loss: 1.0660
Epoch 82/100
3/3 [==============================] - 8s 3s/step - loss: 0.8153 - val_loss: 1.0671
Epoch 83/100
3/3 [==============================] - 8s 3s/step - loss: 0.8160 - val_loss: 1.0688
Epoch 84/100
3/3 [==============================] - 8s 3s/step - loss: 0.8132 - val_loss: 1.0664
Epoch 85/100
3/3 [==============================] - 8s 3s/step - loss: 0.8102 - val_loss: 1.0654
Epoch 86/100
3/3 [==============================] - 8s 3s/step - loss: 0.8081 - val_loss: 1.0653
Epoch 87/100
```

```
Epoch 87/100
3/3 [==============================] - 8s 3s/step - loss: 0.8054 - val_loss: 1.0664
Epoch 88/100
1/3 [=========>....................] - ETA: 5s - loss: 0.8124
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-58-32a430cf4dce> in <module>()
     18 train_steps=train.shape[0]//512
     19 valid_steps=validation.shape[0]//20
---> 20 model.fit_generator(train_dataloader, steps_per_epoch=train_steps, epochs=100, validation_data=te
st_dataloader, validation_steps=valid_steps)

/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py in fit_generator(self,
generator, steps_per_epoch, epochs, verbose, callbacks, validation_data, validation_steps,
validation_freq, class_weight, max_queue_size, workers, use_multiprocessing, shuffle, initial_epoch)
   1955         use_multiprocessing=use_multiprocessing,
   1956         shuffle=shuffle,
-> 1957         initial_epoch=initial_epoch)
   1958
   1959   def evaluate_generator(self,

/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py in fit(self, x, y, batc
h_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, class_weight,
sample_weight, initial_epoch, steps_per_epoch, validation_steps, validation_batch_size, validation_freq,
max_queue_size, workers, use_multiprocessing)
   1181                 _r=1):
   1182               callbacks.on_train_batch_begin(step)
-> 1183               tmp_logs = self.train_function(iterator)
   1184               if data_handler.should_sync:
   1185                 context.async_wait()

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py in __call__(self, *args,
**kwds)
    887
    888       with OptionalXlaContext(self._jit_compile):
--> 889         result = self._call(*args, **kwds)
    890
    891       new_tracing_count = self.experimental_get_tracing_count()

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py in _call(self, *args,
**kwds)
    915       # In this case we have created variables on the first call, so we run the
    916       # defunned version which is guaranteed to never create variables.
--> 917       return self._stateless_fn(*args, **kwds)  # pylint: disable=not-callable
    918     elif self._stateful_fn is not None:
    919       # Release the lock early so that multiple threads can perform the call

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in __call__(self, *args, **kwa
rgs)
   3022        filtered_flat_args) = self._maybe_define_function(args, kwargs)
   3023     return graph_function._call_flat(
-> 3024         filtered_flat_args, captured_inputs=graph_function.captured_inputs)  # pylint: disable=pr
otected-access
   3025
   3026   @property

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in _call_flat(self, args, capt
ured_inputs, cancellation_manager)
   1959       # No tape is watching; skip to running the function.
   1960       return self._build_call_outputs(self._inference_function.call(
-> 1961           ctx, args, cancellation_manager=cancellation_manager))
   1962     forward_backward = self._select_forward_and_backward_functions(
   1963         args,

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in call(self, ctx, args, cance
llation_manager)
    594               inputs=args,
    595               attrs=attrs,
--> 596               ctx=ctx)
    597         else:
    598           outputs = execute.execute_with_cancellation(

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/execute.py in quick_execute(op_name,
num_outputs, inputs, attrs, ctx, name)
     58     ctx.ensure_initialized()
     59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
---> 60                                         inputs, attrs, num_outputs)
     61   except core._NotOkStatusException as e:
```

```
    62     if name is not None:

KeyboardInterrupt:
```

In [59]:

```
batch_size=512
units=128
```

In [62]:

```
def predict(input_sentence):

  '''
  A. Given input sentence, convert the sentence into integers using tokenizer used earlier
  B. Pass the input_sequence to encoder. we get encoder_outputs, last time step hidden and cell state
  C. Initialize index of <start> as input to decoder. and encoder final states as input_states to onestep
  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,
        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
  '''
  initial_state_enc=[np.zeros((batch_size,units)),np.zeros((batch_size,units))]
  inp_seq = tknizer_source.texts_to_sequences([input_sentence])
  inp_seq = pad_sequences(inp_seq,padding='post',maxlen=170)

  en_outputs,state_h , state_c = model.layers[0](tf.constant(inp_seq),initial_state_enc)
  cur_vec = tf.constant([[tknizer_target.word_index['\t']]])
  pred = []
  #Here 20 is the max_length of the sequence
  for i in range(200):
    output,state_h,state_c,attention_weights,context_vector = model.layers[1].onestepdecoder(cur_vec,en_o
    cur_vec = np.reshape(np.argmax(output), (1, 1))
    pred.append(tknizer_target.index_word[cur_vec[0][0]])
    if(pred[-1]=='\n'):
      break
    translated_sentence = ' '.join(pred)
  return translated_sentence
```

In [68]:

```
for i in validation['source']:
  pred=predict(i)
  print(i)
  print(pred)
```

```
Hi its me you are probably having too much fun to get this message but i thought id txt u cos im bored!
and james has been farting at me all night
H i m m m e e r e r   o r o u r   n o u r o n o u r e r   n o u r o n o u r e r   n o u r o n o u r e r
n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o
u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o
u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o
Yupz... Luv my trip...Weather is great too... Cant take e heat here now...
H i m m m e e r e r   o r o u r   n o u r o n o u r e r   n o u r o n o u r e r   n o u r o n o u r e r
n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o
u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o
u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o
Thanx 4 the time we've spent 2geva, its bin mint! Ur my Baby and all I want is u!
H i m m m e e r e r   o r o u r   n o u r o n o u r e r   n o u r o n o u r e r   n o u r o n o u r e r
n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o
u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o
u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o
Hey....I know its rude of me not to do something abt e fone.N i'm sorry it died on u.
H i m m m e e r e r   o r o u r   n o u r o n o u r e r   n o u r o n o u r e r   n o u r o n o u r e r
n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o
u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o
u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o
Haha... Okay... Sure thing must carry around sch ah....
H i m m m e e r e r   o r o u r   n o u r o n o u r e r   n o u r o n o u r e r   n o u r o n o u r e r
n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o
u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o
u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o
Yup... Wah. How come you choose comp sci? After discussion with parents and sis? What mod ü need to
take? Ü need to take stats? And ü doing gem this sem ?
H i m m m e e r e r   o r o u r   n o u r o n o u r e r   n o u r o n o u r e r   n o u r o n o u r e r
n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o
u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o
u r o n o u r e r   n o u r e r   n o u r o n o u r e r   n o u r e r   n o u r o n o
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-68-7433921a2b4d> in <module>()
      1 for i in validation['source']:
----> 2   pred=predict([i])
      3   print(i)
      4   print(pred)

<ipython-input-62-e2ceec33b5ae> in predict(input_sentence)
     21   #Here 20 is the max_length of the sequence
     22   for i in range(200):
---> 23     output,state_h,state_c,attention_weights,context_vector = model.layers[1].onestepdecoder(cur_v
ec,en_outputs,state_h,state_c)
     24     cur_vec = np.reshape(np.argmax(output), (1, 1))
     25     pred.append(tknizer_target.index_word[cur_vec[0][0]])

/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py in layers(self)
   2487   @property
   2488   def layers(self):
-> 2489     return list(self._flatten_layers(include_self=False, recursive=False))
   2490
   2491   def get_layer(self, name=None, index=None):

/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/base_layer.py in
_flatten_layers(self, recursive, include_self)
   2821   def _flatten_layers(self, recursive=True, include_self=True):
   2822     for m in self._flatten_modules(
-> 2823         recursive=recursive, include_self=include_self):
   2824       if isinstance(m, Layer):
   2825         yield m

/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/base_layer.py in
_flatten_modules(self, recursive, include_self)
   2852         # Metrics are not considered part of the Layer's topology.
   2853         if (isinstance(trackable_obj, module.Module) and
-> 2854             not isinstance(trackable_obj, metrics_mod.Metric)):
   2855           yield trackable_obj
   2856           # Introspect recursively through sublayers.

/usr/lib/python3.7/abc.py in __instancecheck__(cls, instance)
    137         def __instancecheck__(cls, instance):
    138             """Override for isinstance(instance, cls)."""
--> 139             return _abc_instancecheck(cls, instance)
    140
    141         def __subclasscheck__(cls, subclass):

KeyboardInterrupt:
```

In [66]:

```
# Predict on 1000 random sentences on test data and calculate the average BLEU score of these sentences.
import nltk.translate.bleu_score as bleu
bleu_scores_lst=[]
for i in validation[:]['source']:
  reference = [i.split(),] # the original
  predicted=predict(i)
  translation = predicted.split()
  values=bleu.sentence_bleu(reference, translation)
  bleu_scores_lst.append(values)

# https://www.nltk.org/_modules/nltk/translate/bleu_score.html
```

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

In [67]:

```
average_bleu_scores=sum(bleu_scores_lst)/len(bleu_scores_lst)
print("Average BLEU score of these 1000 test data sentences is: ",average_bleu_scores)
```

```
Average BLEU score of these 1000 test data sentences is:  0.08111379872068883
```

In [ ]:

```
bleu_scores_lst
```

Out[ ]:

```
[0.287190894500909,
 0,
 0.34152945510447685,
 0,
 0.287190894500909,
 0.34152945510447685,
 0,
 0,
 0,
 0,
 0,
 0.287190894500909,
 0,
 0.287190894500909,
 0,
 0.3779644730092272,
 0.287190894500909,
 0,
 0.287190894500909,
 0]
```

In [ ]: