In [54]:

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

```
Downloading...
From: https://drive.google.com/uc?id=1OurDQUtbWQacvT32HMqFL7vIUrSMllOp
To: /content/preprocessed_data.csv
100% 300k/300k [00:00<00:00, 43.2MB/s]
```

In [55]:

```
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 [56]:

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

In [57]:

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

Out[57]:

	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 [58]:

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

In [59]:

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

In [60]:

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

Out[60]:

	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 [61]:

```
df.shape
```

Out[61]:

```
(2000, 2)
```

In [62]:

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

In [63]:

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

In [64]:

```
df.shape
```

Out[64]:

```
(1990, 2)
```

In [65]:

```
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[65]:

	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 [66]:

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

In [67]:

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

Out[67]:

	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 [68]:

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

In [69]:

```
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 [70]:

```
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 [71]:

```
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)
```

```
3029
3698
```

In [72]:

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

Out[72]:

```
(1, 1439)
```

In [73]:

```
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 [74]:

```
#https://github.com/UdiBhaskar/TfKeras-Custom-Layers/blob/master/Seq2Seq/clayers.py
def _attention_score(dec_ht,
                     enc_hs,
                     attention_type,
                     weightwa=None,
                     weightua=None,
                     weightva=None):
    if attention_type == 'bahdanau':
        score = weightva(tf.nn.tanh(weightwa(dec_ht) + weightua(enc_hs)))
        score = tf.squeeze(score, [2])
    elif attention_type == 'dot':
        score = tf.matmul(dec_ht, enc_hs, transpose_b=True)
        score = tf.squeeze(score, 1)
    elif attention_type == 'general':
        score = weightwa(enc_hs)
        score = tf.matmul(dec_ht, score, transpose_b=True)
        score = tf.squeeze(score, 1)
    elif attention_type == 'concat':
        dec_ht = tf.tile(dec_ht, [1, enc_hs.shape[1], 1])
        score = weightva(tf.nn.tanh(weightwa(tf.concat((dec_ht, enc_hs), axis=-1))))
        score = tf.squeeze(score, 2)

    return score
```

In [75]:

```
def _monotonic_attetion(probabilities, attention_prev, mode):

    """Compute monotonic attention distribution from choosing probabilities.
    Implemented Based on -
    https://colinraffel.com/blog/online-and-linear-time-attention-by-enforcing-monotonic-alignments.html
    https://arxiv.org/pdf/1704.00784.pdf
    Mainly implemented by referring
    https://github.com/craffel/mad/blob/b3687a70615044359c8acc440e43a5e23dc58309/example_decoder.py#L22
    # Arguments:
        probabilities: Probability of choosing input sequence..
                       Should be of shape (batch_size, max_length),
                       and should all be in the range [0, 1].
        attention_prev: The attention distribution from the previous output timestep.
                            Should be of shape (batch_size, max_length).
                            For the first output timestep,
                            should be [1, 0, 0, ...,0] for all n in [0, ... batch_size - 1].
        mode: How to compute the attention distribution.
              Must be one of 'recursive', 'parallel', or 'hard'.
```

```
            - 'recursive' uses tf.scan to recursively compute the distribution.
            This is slowest but is exact, general, and does not suffer from
            numerical instabilities.
            - 'parallel' uses parallelized cumulative-sum and cumulative-product
            operations to compute a closed-form solution to the recurrence relation
            defining the attention distribution.  This makes it more efficient than 'recursive',
            but it requires numerical checks which make the distribution non-exact.
            This can be a problem in particular when max_length is long and/or
            probabilities has entries very close to 0 or 1.
            - 'hard' requires that  the probabilities in p_choose_i are all either 0 or 1,
            and subsequently uses a more efficient and exact solution.
    # Returns: A tensor of shape (batch_size, max_length) representing the attention distributions
               for each sequence in the batch.
    # Raises:
             ValueError: if mode is not one of 'recursive', 'parallel', 'hard'."""
    if mode == 'hard':
        #Remove any probabilities before the index chosen last time step
        probabilities = probabilities*tf.cumsum(attention_prev, axis=1)
        attention = probabilities*tf.math.cumprod(1-probabilities, axis=1, exclusive=True)
    elif mode == 'recursive':
        batch_size = tf.shape(input=probabilities)[0]
        shifted_1mp_probabilities = tf.concat([tf.ones((batch_size, 1)),\
            1 - probabilities[:, :-1]], 1)
        attention = probabilities*tf.transpose(a=tf.scan(lambda x, yz: tf.reshape(yz[0]*x + yz[1],\
            (batch_size,)), [tf.transpose(a=shifted_1mp_probabilities),\
                tf.transpose(a=attention_prev)], tf.zeros((batch_size,))))
    elif mode == 'parallel':
        cumprod_1mp_probabilities = tf.exp(tf.cumsum(tf.math.log(tf.clip_by_value(1-probabilities,\
            1e-10, 1)), axis=1, exclusive=True))
        attention = probabilities*cumprod_1mp_probabilities*tf.cumsum(attention_prev/\
            tf.clip_by_value(cumprod_1mp_probabilities, 1e-10, 1.), axis=1)
    else:
        raise ValueError("Mode must be 'hard', 'parallel' or 'recursive' ")

    return attention
```

In [76]:

```
class MonotonicBahdanauAttention(tf.keras.layers.Layer):
    '''
    MonotonicBahdanauAttention
    Implemented based on below paper
        https://arxiv.org/pdf/1704.00784.pdf
    # Arguments
        units = number of hidden units to use.
        mode = How to compute the attention distribution.
              Must be one of 'recursive', 'parallel', or 'hard'.
              - 'recursive' uses tf.scan to recursively compute the distribution.
              This is slowest but is exact, general, and does not suffer from
              numerical instabilities.
              - 'parallel' uses parallelized cumulative-sum and cumulative-product
              operations to compute a closed-form solution to the recurrence relation
              defining the attention distribution.  This makes it more efficient than 'recursive',
              but it requires numerical checks which make the distribution non-exact.
              This can be a problem in particular when max_length is long and/or
              probabilities has entries very close to 0 or 1.
              - 'hard' requires that  the probabilities in p_choose_i are all either 0 or 1,
              and subsequently uses a more efficient and exact solution.
        return_aweights = Bool, whether to return attention weights or not.
        scaling_factor = int/float to scale the score vector. default None=1
        noise_std = standard deviation of noise which will be added before
                    applying sigmoid function.(pre-sigmoid noise). If it is 0 or
                    mode="hard", we won't add any noise.
        weights_initializer = initializer for weight matrix
        weights_regularizer = Regularize the weights
        weights_constraint = Constraint function applied to the weights
    # Returns
        context_vector = context vector after applying attention.
        attention_weights = attention weights only if `return_aweights=True`.
    # Inputs to the layer
        inputs = dictionary with keys "enocderHs", "decoderHt", "prevAttention".
                enocderHs = all the encoder hidden states,
                            shape - (Batchsize, encoder_seq_len, enc_hidden_size)
                decoderHt = hidden state of decoder at that timestep,
                            shape - (Batchsize, dec_hidden_size)
                prevAttention = Previous probability distribution of attention
                                (previous attention weights)
        mask = You can apply mask for padded values or any custom values
```

```
            while calculating attention.
            if you are giving mask for encoder and deocoder then you have
            to give a dict similar to inputs. (keys: enocderHs, decoderHt)
            else you can give only for enocoder normally.(one tensor)
            mask shape should be (Batchsize, encoder_seq_len)
'''

def __init__(self, units,
             mode='parallel',
             return_aweights=False,
             scaling_factor=None,
             noise_std=0,
             weights_initializer='he_normal',
             bias_initializer='zeros',
             weights_regularizer=None,
             bias_regularizer=None,
             weights_constraint=None,
             bias_constraint=None,
             **kwargs):
    if 'name' not in kwargs:
        kwargs['name'] = ""
    super(MonotonicBahdanauAttention, self).__init__(**kwargs)
    self.units = units
    self.mode = mode
    self.return_aweights = return_aweights
    self.scaling_factor = scaling_factor
    self.noise_std = noise_std
    self.weights_initializer = tf.keras.initializers.get(weights_initializer)
    self.bias_initializer = tf.keras.initializers.get(bias_initializer)
    self.weights_regularizer = tf.keras.regularizers.get(weights_regularizer)
    self.bias_regularizer = tf.keras.regularizers.get(bias_regularizer)
    self.weights_constraint = tf.keras.constraints.get(weights_constraint)
    self.bias_constraint = tf.keras.constraints.get(bias_constraint)
    self._wa = tf.keras.layers.Dense(self.units, use_bias=False,\
        kernel_initializer=weights_initializer, bias_initializer=bias_initializer,\
            kernel_regularizer=weights_regularizer, bias_regularizer=bias_regularizer,\
                kernel_constraint=weights_constraint, bias_constraint=bias_constraint,\
                    name=self.name+"Wa")
    self._ua = tf.keras.layers.Dense(self.units,\
        kernel_initializer=weights_initializer, bias_initializer=bias_initializer,\
            kernel_regularizer=weights_regularizer, bias_regularizer=bias_regularizer,\
                kernel_constraint=weights_constraint, bias_constraint=bias_constraint,\
                    name=self.name+"Ua")
    self._va = tf.keras.layers.Dense(1, use_bias=False, kernel_initializer=weights_initializer,\
        kernel_regularizer=weights_regularizer, bias_regularizer=bias_regularizer,\
            bias_initializer=bias_initializer, kernel_constraint=weights_constraint,\
                bias_constraint=bias_constraint, name=self.name+"Va")
    self.supports_masking = True


def call(self, inputs, training=True):
    '''call'''
    #assert isinstance(inputs, dict)


    if ('enocderHs' not in inputs.keys())or ('decoderHt' not in inputs.keys()\
        or 'prevAttention' not in inputs.keys()):
        raise ValueError("Input to the layer must be a dict with \
        keys=['enocderHs','decoderHt', 'prevAttention']")

    #if isinstance(mask, dict):
    #   mask_enc = mask.get('enocderHs', None)
    #   mask_dec = mask.get('decoderHt', None)
    #else:
    #    mask_enc = mask
    #    mask_dec = None
    enc_out, dec_prev_hs = tf.cast(inputs['enocderHs'], tf.float32), \
        tf.cast(inputs['decoderHt'], tf.float32)

    prev_attention = inputs['prevAttention']


    #if mask_dec is not None:
       # dec_prev_hs = dec_prev_hs * tf.cast(mask_dec, dec_prev_hs.dtype)
    #if mask_enc is not None:
       # enc_out = enc_out * tf.cast(tf.expand_dims(mask_enc, 2), enc_out.dtype)
```

```
        # decprev_hs - Decoder hidden shape == (batch_size, hidden size)
        # hidden_with_time_axis shape == (batch_size, 1, hidden size)
        dec_hidden_with_time_axis = tf.expand_dims(dec_prev_hs, 1)


        # score shape == (batch_size, max_length)
        score = _attention_score(dec_ht=dec_hidden_with_time_axis, enc_hs=enc_out,\
                    attention_type='concat', weightwa=self._wa,\
                        weightua=self._ua, weightva=self._va)

        if self.scaling_factor is not None:
            score = score/tf.sqrt(self.scaling_factor)

        if training:
            if self.noise_std > 0:
                random_noise = tf.random.normal(shape=tf.shape(input=score), mean=0,\
                    stddev=self.noise_std, dtype=score.dtype, seed=self.seed)
                score = score + random_noise

        #if mask_enc is not None:
        #    score = score + (tf.cast(tf.math.equal(mask_enc, False), score.dtype)*-1e9)

        if self.mode == 'hard':
            probabilities = tf.cast(score > 0, score.dtype)
        else:
            probabilities = tf.sigmoid(score)

        attention_weights = _monotonic_attetion(probabilities, prev_attention, self.mode)
        attention_weights = tf.expand_dims(attention_weights, 1)

        #context_vector shape (batch_size, hidden_size)
        context_vector = tf.matmul(attention_weights, enc_out)
        context_vector = tf.squeeze(context_vector, 1, name="context_vector")

        if self.return_aweights:

            return context_vector, tf.squeeze(attention_weights, 1, name='attention_weights')
        return context_vector
```

In [77]:

```
class One_Step_Decoder(tf.keras.Model):
  def __init__(self,tar_vocab_size, embedding_dim, input_length, dec_units ,mode ,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.mode = mode
      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")
      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 = MonotonicBahdanauAttention(self.att_units,mode=self.mode,return_aweights=True)

  def call(self,input_to_decoder, encoder_output, state_h,state_c,attention_weights):


    inputs= dict()
    inputs['enocderHs']=encoder_output
    inputs['decoderHt']=state_h
    inputs['prevAttention']=attention_weights

    output = self.embedding(input_to_decoder)
    context_vector,attention_weights = self.attention(inputs)
    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 [78]:

```
class Decoder(tf.keras.Model):
    def __init__(self,out_vocab_size, embedding_dim, input_length, dec_units ,mode ,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.mode=mode
      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,attention_weig

        #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 [79]:

```
class encoder_decoder(tf.keras.Model):
  def __init__(self,encoder_inputs_length,decoder_inputs_length, output_vocab_size,batch_size,mode):
    #Intialize objects from encoder decoder
    super().__init__() # https://stackoverflow.com/a/27134600/4084039
    self.batch_size=batch_size
    self.encoder_inputs_length=encoder_inputs_length
    self.decoder_inputs_length=decoder_inputs_length
    self.encoder = Encoder(vocab_size_source+1,300,128,self.encoder_inputs_length)
    self.decoder = Decoder(vocab_size_target+1,300,self.decoder_inputs_length,128,mode,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]
    attention_weights = np.zeros((self.batch_size,self.encoder_inputs_length),dtype='float32')
    attention_weights[:,0]=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,attention_weights)
    return decoder_output
```

In [80]:

```
#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 l
    for the padded zeros. i.e when the input is zero then we donot need to worry what the output is. Thi
    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 [81]:

```
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 [82]:

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

train_dataloader = Dataloder(train_dataset, batch_size=20)
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, 39) (512, 39)
```

In [83]:

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

```
WARNING:tensorflow:From <ipython-input-83-bdb3352f611a>:1: experimental_run_functions_eagerly (from tenso
rflow.python.eager.def_function) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.config.run_functions_eagerly` instead of the experimental version.
```

In [84]:

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

In [91]:

```
#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=39,output_vocab_size=vocab_size_t
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=260)
```

```
/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 1/260
3/3 [==============================] - 2s 743ms/step - loss: 3.0825
Epoch 2/260
3/3 [==============================] - 2s 765ms/step - loss: 2.5597
Epoch 3/260
3/3 [==============================] - 2s 792ms/step - loss: 2.4022
Epoch 4/260
3/3 [==============================] - 2s 729ms/step - loss: 2.3590
Epoch 5/260
3/3 [==============================] - 2s 732ms/step - loss: 2.3331
Epoch 6/260
3/3 [==============================] - 2s 721ms/step - loss: 2.3062
Epoch 7/260
3/3 [==============================] - 2s 722ms/step - loss: 2.2742
Epoch 8/260
3/3 [==============================] - 2s 744ms/step - loss: 2.2484
Epoch 9/260
3/3 [==============================] - 2s 718ms/step - loss: 2.2178
Epoch 10/260
3/3 [==============================] - 2s 726ms/step - loss: 2.1843
Epoch 11/260
3/3 [==============================] - 2s 749ms/step - loss: 2.1481
Epoch 12/260
3/3 [==============================] - 2s 796ms/step - loss: 2.1150
Epoch 13/260
3/3 [==============================] - 2s 721ms/step - loss: 2.0810
Epoch 14/260
3/3 [==============================] - 2s 732ms/step - loss: 2.0482
Epoch 15/260
3/3 [==============================] - 2s 734ms/step - loss: 2.0164
Epoch 16/260
3/3 [==============================] - 2s 744ms/step - loss: 1.9865
Epoch 17/260
3/3 [==============================] - 2s 738ms/step - loss: 1.9572
Epoch 18/260
3/3 [==============================] - 2s 738ms/step - loss: 1.9336
Epoch 19/260
3/3 [==============================] - 2s 739ms/step - loss: 1.9052
Epoch 20/260
3/3 [==============================] - 2s 739ms/step - loss: 1.8785
Epoch 21/260
3/3 [==============================] - 2s 746ms/step - loss: 1.8536
Epoch 22/260
3/3 [==============================] - 2s 732ms/step - loss: 1.8336
Epoch 23/260
3/3 [==============================] - 2s 735ms/step - loss: 1.8096
Epoch 24/260
3/3 [==============================] - 2s 723ms/step - loss: 1.7856
Epoch 25/260
3/3 [==============================] - 2s 742ms/step - loss: 1.7616
Epoch 26/260
3/3 [==============================] - 2s 723ms/step - loss: 1.7424
Epoch 27/260
```

```
Epoch 27/260
3/3 [==============================] - 2s 779ms/step - loss: 1.7244
Epoch 28/260
3/3 [==============================] - 2s 719ms/step - loss: 1.7009
Epoch 29/260
3/3 [==============================] - 2s 737ms/step - loss: 1.6814
Epoch 30/260
3/3 [==============================] - 2s 753ms/step - loss: 1.6587
Epoch 31/260
3/3 [==============================] - 2s 751ms/step - loss: 1.6414
Epoch 32/260
3/3 [==============================] - 2s 725ms/step - loss: 1.6219
Epoch 33/260
3/3 [==============================] - 2s 753ms/step - loss: 1.6019
Epoch 34/260
3/3 [==============================] - 2s 745ms/step - loss: 1.5812
Epoch 35/260
3/3 [==============================] - 2s 741ms/step - loss: 1.5608
Epoch 36/260
3/3 [==============================] - 2s 741ms/step - loss: 1.5435
Epoch 37/260
3/3 [==============================] - 2s 735ms/step - loss: 1.5222
Epoch 38/260
3/3 [==============================] - 2s 741ms/step - loss: 1.5047
Epoch 39/260
3/3 [==============================] - 2s 810ms/step - loss: 1.4891
Epoch 40/260
3/3 [==============================] - 2s 732ms/step - loss: 1.4701
Epoch 41/260
3/3 [==============================] - 2s 749ms/step - loss: 1.4510
Epoch 42/260
3/3 [==============================] - 2s 753ms/step - loss: 1.4390
Epoch 43/260
3/3 [==============================] - 2s 743ms/step - loss: 1.4195
Epoch 44/260
3/3 [==============================] - 2s 734ms/step - loss: 1.4016
Epoch 45/260
3/3 [==============================] - 2s 747ms/step - loss: 1.3860
Epoch 46/260
3/3 [==============================] - 2s 811ms/step - loss: 1.3689
Epoch 47/260
3/3 [==============================] - 2s 738ms/step - loss: 1.3540
Epoch 48/260
3/3 [==============================] - 2s 744ms/step - loss: 1.3412
Epoch 49/260
3/3 [==============================] - 2s 738ms/step - loss: 1.3250
Epoch 50/260
3/3 [==============================] - 2s 741ms/step - loss: 1.3122
Epoch 51/260
3/3 [==============================] - 2s 752ms/step - loss: 1.2980
Epoch 52/260
3/3 [==============================] - 2s 732ms/step - loss: 1.2844
Epoch 53/260
3/3 [==============================] - 2s 733ms/step - loss: 1.2699
Epoch 54/260
3/3 [==============================] - 2s 762ms/step - loss: 1.2544
Epoch 55/260
3/3 [==============================] - 2s 800ms/step - loss: 1.2422
Epoch 56/260
3/3 [==============================] - 2s 750ms/step - loss: 1.2350
Epoch 57/260
3/3 [==============================] - 2s 734ms/step - loss: 1.2259
Epoch 58/260
3/3 [==============================] - 2s 755ms/step - loss: 1.2133
Epoch 59/260
3/3 [==============================] - 2s 762ms/step - loss: 1.1994
Epoch 60/260
3/3 [==============================] - 2s 743ms/step - loss: 1.1835
Epoch 61/260
3/3 [==============================] - 2s 731ms/step - loss: 1.1678
Epoch 62/260
3/3 [==============================] - 2s 732ms/step - loss: 1.1542
Epoch 63/260
3/3 [==============================] - 2s 742ms/step - loss: 1.1408
Epoch 64/260
3/3 [==============================] - 2s 727ms/step - loss: 1.1303
Epoch 65/260
```

```
3/3 [==============================] - 2s 742ms/step - loss: 1.1190
Epoch 66/260
3/3 [==============================] - 2s 720ms/step - loss: 1.1053
Epoch 67/260
3/3 [==============================] - 2s 732ms/step - loss: 1.0932
Epoch 68/260
3/3 [==============================] - 2s 800ms/step - loss: 1.0818
Epoch 69/260
3/3 [==============================] - 2s 728ms/step - loss: 1.0692
Epoch 70/260
3/3 [==============================] - 2s 737ms/step - loss: 1.0593
Epoch 71/260
3/3 [==============================] - 2s 753ms/step - loss: 1.0474
Epoch 72/260
3/3 [==============================] - 2s 751ms/step - loss: 1.0372
Epoch 73/260
3/3 [==============================] - 2s 747ms/step - loss: 1.0254
Epoch 74/260
3/3 [==============================] - 2s 758ms/step - loss: 1.0177
Epoch 75/260
3/3 [==============================] - 2s 742ms/step - loss: 1.0073
Epoch 76/260
3/3 [==============================] - 2s 738ms/step - loss: 0.9988
Epoch 77/260
3/3 [==============================] - 2s 746ms/step - loss: 0.9863
Epoch 78/260
3/3 [==============================] - 2s 737ms/step - loss: 0.9771
Epoch 79/260
3/3 [==============================] - 2s 735ms/step - loss: 0.9656
Epoch 80/260
3/3 [==============================] - 2s 729ms/step - loss: 0.9548
Epoch 81/260
3/3 [==============================] - 2s 729ms/step - loss: 0.9427
Epoch 82/260
3/3 [==============================] - 2s 736ms/step - loss: 0.9296
Epoch 83/260
3/3 [==============================] - 2s 795ms/step - loss: 0.9243
Epoch 84/260
3/3 [==============================] - 2s 749ms/step - loss: 0.9167
Epoch 85/260
3/3 [==============================] - 2s 741ms/step - loss: 0.9074
Epoch 86/260
3/3 [==============================] - 2s 756ms/step - loss: 0.9134
Epoch 87/260
3/3 [==============================] - 2s 757ms/step - loss: 0.8936
Epoch 88/260
3/3 [==============================] - 2s 746ms/step - loss: 0.8800
Epoch 89/260
3/3 [==============================] - 2s 734ms/step - loss: 0.8719
Epoch 90/260
3/3 [==============================] - 2s 750ms/step - loss: 0.8589
Epoch 91/260
3/3 [==============================] - 3s 759ms/step - loss: 0.8473
Epoch 92/260
3/3 [==============================] - 2s 751ms/step - loss: 0.8349
Epoch 93/260
3/3 [==============================] - 2s 762ms/step - loss: 0.8262
Epoch 94/260
3/3 [==============================] - 3s 843ms/step - loss: 0.8144
Epoch 95/260
3/3 [==============================] - 2s 739ms/step - loss: 0.8019
Epoch 96/260
3/3 [==============================] - 2s 746ms/step - loss: 0.7936
Epoch 97/260
3/3 [==============================] - 2s 741ms/step - loss: 0.7820
Epoch 98/260
3/3 [==============================] - 2s 742ms/step - loss: 0.7730
Epoch 99/260
3/3 [==============================] - 2s 741ms/step - loss: 0.7635
Epoch 100/260
3/3 [==============================] - 2s 736ms/step - loss: 0.7517
Epoch 101/260
3/3 [==============================] - 2s 733ms/step - loss: 0.7427
Epoch 102/260
3/3 [==============================] - 2s 735ms/step - loss: 0.7307
Epoch 103/260
3/3 [==============================] - 2s 737ms/step - loss: 0.7200
```

```
Epoch 104/260
3/3 [==============================] - 2s 734ms/step - loss: 0.7092
Epoch 105/260
3/3 [==============================] - 2s 743ms/step - loss: 0.7016
Epoch 106/260
3/3 [==============================] - 2s 744ms/step - loss: 0.6892
Epoch 107/260
3/3 [==============================] - 2s 739ms/step - loss: 0.6790
Epoch 108/260
3/3 [==============================] - 2s 744ms/step - loss: 0.6677
Epoch 109/260
3/3 [==============================] - 2s 731ms/step - loss: 0.6590
Epoch 110/260
3/3 [==============================] - 2s 726ms/step - loss: 0.6492
Epoch 111/260
3/3 [==============================] - 2s 788ms/step - loss: 0.6381
Epoch 112/260
3/3 [==============================] - 2s 717ms/step - loss: 0.6309
Epoch 113/260
3/3 [==============================] - 2s 720ms/step - loss: 0.6218
Epoch 114/260
3/3 [==============================] - 2s 737ms/step - loss: 0.6116
Epoch 115/260
3/3 [==============================] - 2s 738ms/step - loss: 0.6065
Epoch 116/260
3/3 [==============================] - 2s 731ms/step - loss: 0.5983
Epoch 117/260
3/3 [==============================] - 2s 730ms/step - loss: 0.5883
Epoch 118/260
3/3 [==============================] - 2s 731ms/step - loss: 0.5786
Epoch 119/260
3/3 [==============================] - 2s 727ms/step - loss: 0.5712
Epoch 120/260
3/3 [==============================] - 2s 787ms/step - loss: 0.5622
Epoch 121/260
3/3 [==============================] - 2s 724ms/step - loss: 0.5538
Epoch 122/260
3/3 [==============================] - 2s 724ms/step - loss: 0.5455
Epoch 123/260
3/3 [==============================] - 2s 724ms/step - loss: 0.5339
Epoch 124/260
3/3 [==============================] - 2s 724ms/step - loss: 0.5258
Epoch 125/260
3/3 [==============================] - 2s 799ms/step - loss: 0.5153
Epoch 126/260
3/3 [==============================] - 2s 742ms/step - loss: 0.5045
Epoch 127/260
3/3 [==============================] - 2s 744ms/step - loss: 0.4963
Epoch 128/260
3/3 [==============================] - 2s 744ms/step - loss: 0.4880
Epoch 129/260
3/3 [==============================] - 2s 732ms/step - loss: 0.4813
Epoch 130/260
3/3 [==============================] - 2s 745ms/step - loss: 0.4716
Epoch 131/260
3/3 [==============================] - 2s 720ms/step - loss: 0.4632
Epoch 132/260
3/3 [==============================] - 2s 778ms/step - loss: 0.4572
Epoch 133/260
3/3 [==============================] - 2s 746ms/step - loss: 0.4472
Epoch 134/260
3/3 [==============================] - 2s 739ms/step - loss: 0.4380
Epoch 135/260
3/3 [==============================] - 2s 729ms/step - loss: 0.4292
Epoch 136/260
3/3 [==============================] - 2s 729ms/step - loss: 0.4223
Epoch 137/260
3/3 [==============================] - 2s 727ms/step - loss: 0.4145
Epoch 138/260
3/3 [==============================] - 2s 737ms/step - loss: 0.4066
Epoch 139/260
3/3 [==============================] - 2s 745ms/step - loss: 0.3993
Epoch 140/260
3/3 [==============================] - 2s 730ms/step - loss: 0.3921
Epoch 141/260
3/3 [==============================] - 2s 732ms/step - loss: 0.3874
Epoch 142/260
```

```
3/3 [==============================] - 2s 747ms/step - loss: 0.3793
Epoch 143/260
3/3 [==============================] - 2s 746ms/step - loss: 0.3721
Epoch 144/260
3/3 [==============================] - 2s 748ms/step - loss: 0.3646
Epoch 145/260
3/3 [==============================] - 2s 738ms/step - loss: 0.3578
Epoch 146/260
3/3 [==============================] - 2s 746ms/step - loss: 0.3510
Epoch 147/260
3/3 [==============================] - 2s 730ms/step - loss: 0.3473
Epoch 148/260
3/3 [==============================] - 2s 808ms/step - loss: 0.3403
Epoch 149/260
3/3 [==============================] - 2s 732ms/step - loss: 0.3338
Epoch 150/260
3/3 [==============================] - 2s 743ms/step - loss: 0.3298
Epoch 151/260
3/3 [==============================] - 2s 740ms/step - loss: 0.3219
Epoch 152/260
3/3 [==============================] - 2s 726ms/step - loss: 0.3180
Epoch 153/260
3/3 [==============================] - 2s 731ms/step - loss: 0.3127
Epoch 154/260
3/3 [==============================] - 2s 778ms/step - loss: 0.3059
Epoch 155/260
3/3 [==============================] - 2s 728ms/step - loss: 0.2986
Epoch 156/260
3/3 [==============================] - 2s 737ms/step - loss: 0.2926
Epoch 157/260
3/3 [==============================] - 2s 731ms/step - loss: 0.2884
Epoch 158/260
3/3 [==============================] - 2s 729ms/step - loss: 0.2821
Epoch 159/260
3/3 [==============================] - 2s 722ms/step - loss: 0.2776
Epoch 160/260
3/3 [==============================] - 2s 731ms/step - loss: 0.2717
Epoch 161/260
3/3 [==============================] - 2s 732ms/step - loss: 0.2672
Epoch 162/260
3/3 [==============================] - 2s 716ms/step - loss: 0.2624
Epoch 163/260
3/3 [==============================] - 2s 721ms/step - loss: 0.2560
Epoch 164/260
3/3 [==============================] - 2s 784ms/step - loss: 0.2526
Epoch 165/260
3/3 [==============================] - 2s 729ms/step - loss: 0.2472
Epoch 166/260
3/3 [==============================] - 2s 732ms/step - loss: 0.2415
Epoch 167/260
3/3 [==============================] - 2s 738ms/step - loss: 0.2373
Epoch 168/260
3/3 [==============================] - 2s 721ms/step - loss: 0.2327
Epoch 169/260
3/3 [==============================] - 2s 729ms/step - loss: 0.2277
Epoch 170/260
3/3 [==============================] - 2s 740ms/step - loss: 0.2241
Epoch 171/260
3/3 [==============================] - 2s 753ms/step - loss: 0.2204
Epoch 172/260
3/3 [==============================] - 2s 746ms/step - loss: 0.2157
Epoch 173/260
3/3 [==============================] - 2s 746ms/step - loss: 0.2119
Epoch 174/260
3/3 [==============================] - 2s 738ms/step - loss: 0.2073
Epoch 175/260
3/3 [==============================] - 2s 782ms/step - loss: 0.2034
Epoch 176/260
3/3 [==============================] - 2s 736ms/step - loss: 0.2001
Epoch 177/260
3/3 [==============================] - 2s 715ms/step - loss: 0.1964
Epoch 178/260
3/3 [==============================] - 2s 723ms/step - loss: 0.1916
Epoch 179/260
3/3 [==============================] - 2s 785ms/step - loss: 0.1877
Epoch 180/260
3/3 [==============================] - 2s 729ms/step - loss: 0.1838
```

```
Epoch 181/260
3/3 [==============================] - 2s 725ms/step - loss: 0.1802
Epoch 182/260
3/3 [==============================] - 2s 726ms/step - loss: 0.1768
Epoch 183/260
3/3 [==============================] - 2s 725ms/step - loss: 0.1731
Epoch 184/260
3/3 [==============================] - 2s 760ms/step - loss: 0.1696
Epoch 185/260
3/3 [==============================] - 2s 741ms/step - loss: 0.1665
Epoch 186/260
3/3 [==============================] - 2s 747ms/step - loss: 0.1645
Epoch 187/260
3/3 [==============================] - 2s 717ms/step - loss: 0.1617
Epoch 188/260
3/3 [==============================] - 2s 721ms/step - loss: 0.1599
Epoch 189/260
3/3 [==============================] - 2s 732ms/step - loss: 0.1577
Epoch 190/260
3/3 [==============================] - 2s 729ms/step - loss: 0.1550
Epoch 191/260
3/3 [==============================] - 2s 726ms/step - loss: 0.1521
Epoch 192/260
3/3 [==============================] - 2s 721ms/step - loss: 0.1490
Epoch 193/260
3/3 [==============================] - 2s 721ms/step - loss: 0.1461
Epoch 194/260
3/3 [==============================] - 2s 723ms/step - loss: 0.1445
Epoch 195/260
3/3 [==============================] - 2s 719ms/step - loss: 0.1421
Epoch 196/260
3/3 [==============================] - 2s 722ms/step - loss: 0.1396
Epoch 197/260
3/3 [==============================] - 2s 715ms/step - loss: 0.1367
Epoch 198/260
3/3 [==============================] - 2s 718ms/step - loss: 0.1342
Epoch 199/260
3/3 [==============================] - 2s 725ms/step - loss: 0.1328
Epoch 200/260
3/3 [==============================] - 2s 723ms/step - loss: 0.1308
Epoch 201/260
3/3 [==============================] - 2s 776ms/step - loss: 0.1289
Epoch 202/260
3/3 [==============================] - 2s 715ms/step - loss: 0.1270
Epoch 203/260
3/3 [==============================] - 2s 730ms/step - loss: 0.1250
Epoch 204/260
3/3 [==============================] - 2s 730ms/step - loss: 0.1240
Epoch 205/260
3/3 [==============================] - 2s 780ms/step - loss: 0.1220
Epoch 206/260
3/3 [==============================] - 2s 719ms/step - loss: 0.1202
Epoch 207/260
3/3 [==============================] - 2s 723ms/step - loss: 0.1186
Epoch 208/260
3/3 [==============================] - 2s 722ms/step - loss: 0.1166
Epoch 209/260
3/3 [==============================] - 2s 718ms/step - loss: 0.1144
Epoch 210/260
3/3 [==============================] - 2s 715ms/step - loss: 0.1126
Epoch 211/260
3/3 [==============================] - 2s 733ms/step - loss: 0.1111
Epoch 212/260
3/3 [==============================] - 2s 753ms/step - loss: 0.1092
Epoch 213/260
3/3 [==============================] - 2s 781ms/step - loss: 0.1071
Epoch 214/260
3/3 [==============================] - 2s 734ms/step - loss: 0.1051
Epoch 215/260
3/3 [==============================] - 2s 732ms/step - loss: 0.1036
Epoch 216/260
3/3 [==============================] - 2s 725ms/step - loss: 0.1020
Epoch 217/260
3/3 [==============================] - 2s 724ms/step - loss: 0.1009
Epoch 218/260
3/3 [==============================] - 2s 725ms/step - loss: 0.0995
Epoch 219/260
```

```
3/3 [==============================] - 2s 720ms/step - loss: 0.0981
Epoch 220/260
3/3 [==============================] - 2s 716ms/step - loss: 0.0974
Epoch 221/260
3/3 [==============================] - 2s 785ms/step - loss: 0.0962
Epoch 222/260
3/3 [==============================] - 2s 723ms/step - loss: 0.0949
Epoch 223/260
3/3 [==============================] - 2s 739ms/step - loss: 0.0935
Epoch 224/260
3/3 [==============================] - 2s 741ms/step - loss: 0.0929
Epoch 225/260
3/3 [==============================] - 2s 729ms/step - loss: 0.0916
Epoch 226/260
3/3 [==============================] - 2s 739ms/step - loss: 0.0904
Epoch 227/260
3/3 [==============================] - 2s 730ms/step - loss: 0.0890
Epoch 228/260
3/3 [==============================] - 2s 781ms/step - loss: 0.0883
Epoch 229/260
3/3 [==============================] - 2s 729ms/step - loss: 0.0874
Epoch 230/260
3/3 [==============================] - 2s 728ms/step - loss: 0.0873
Epoch 231/260
3/3 [==============================] - 2s 730ms/step - loss: 0.0862
Epoch 232/260
3/3 [==============================] - 2s 730ms/step - loss: 0.0851
Epoch 233/260
3/3 [==============================] - 2s 741ms/step - loss: 0.0842
Epoch 234/260
3/3 [==============================] - 2s 724ms/step - loss: 0.0829
Epoch 235/260
3/3 [==============================] - 2s 723ms/step - loss: 0.0822
Epoch 236/260
3/3 [==============================] - 2s 737ms/step - loss: 0.0811
Epoch 237/260
3/3 [==============================] - 2s 731ms/step - loss: 0.0801
Epoch 238/260
3/3 [==============================] - 2s 727ms/step - loss: 0.0794
Epoch 239/260
3/3 [==============================] - 2s 718ms/step - loss: 0.0784
Epoch 240/260
3/3 [==============================] - 2s 720ms/step - loss: 0.0778
Epoch 241/260
3/3 [==============================] - 2s 809ms/step - loss: 0.0771
Epoch 242/260
3/3 [==============================] - 2s 733ms/step - loss: 0.0763
Epoch 243/260
3/3 [==============================] - 2s 729ms/step - loss: 0.0756
Epoch 244/260
3/3 [==============================] - 2s 722ms/step - loss: 0.0749
Epoch 245/260
3/3 [==============================] - 2s 726ms/step - loss: 0.0744
Epoch 246/260
3/3 [==============================] - 2s 729ms/step - loss: 0.0743
Epoch 247/260
3/3 [==============================] - 2s 717ms/step - loss: 0.0735
Epoch 248/260
3/3 [==============================] - 2s 708ms/step - loss: 0.0730
Epoch 249/260
3/3 [==============================] - 2s 723ms/step - loss: 0.0728
Epoch 250/260
3/3 [==============================] - 2s 726ms/step - loss: 0.0725
Epoch 251/260
3/3 [==============================] - 2s 725ms/step - loss: 0.0719
Epoch 252/260
3/3 [==============================] - 2s 709ms/step - loss: 0.0714
Epoch 253/260
3/3 [==============================] - 2s 729ms/step - loss: 0.0706
Epoch 254/260
3/3 [==============================] - 2s 733ms/step - loss: 0.0700
Epoch 255/260
3/3 [==============================] - 2s 731ms/step - loss: 0.0698
Epoch 256/260
3/3 [==============================] - 2s 732ms/step - loss: 0.0697
Epoch 257/260
3/3 [==============================] - 2s 731ms/step - loss: 0.0687
```

```
Epoch 258/260
3/3 [==============================] - 2s 726ms/step - loss: 0.0679
Epoch 259/260
3/3 [==============================] - 2s 727ms/step - loss: 0.0673
Epoch 260/260
3/3 [==============================] - 2s 721ms/step - loss: 0.0666
```

Out[91]:

```
<tensorflow.python.keras.callbacks.History at 0x7f2fe20e4150>
```

In [92]:

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

In [93]:

```
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)
  attention_weights = np.zeros((1,39),dtype='float32')
  attention_weights[:,0]=1

  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(39):
    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 [94]:

```
for i in validation['source']:
  print("The Actual output is: ",i)
  predicted=predict(i)
  print("The predicted output is: ",predicted)
  print(">"*180)
```

```
The Actual output is:  Wow, haha, can go try b mistress 4 few days lor, get a taste of it... Aiya, in ar
my one lor, u wont noe one.... Den mei one is kua's fren, thgt u noe liao...
The predicted output is:  i just got time do you that i don't know what time then where are quite well h
ope you are you mum there maybe it's okay all depends on tuesday i've got like to see die we go and
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual output is:  Same as u 1245... I oso wan shop, but cant leh, parents dun let me go out liao...
The predicted output is:  ok i'm sorry to watch in camp too so how about so bored or smu for your give
my like there and then end of her outside this show is there then there will tell you get it then
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual output is:  Aiya... Lk tt den no nd go oredi... So pissed... Dunno wat's wrong w it...
The predicted output is:  hmm nevermind
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual output is:  Lea my day fine. How's yrs? Lea yr real name?
The predicted output is:  hmm sorry i think can meet you reach on the girl it will reach home from home
now you get it by this from town anyway thanks for me your parents then maybe it's okay all depends on m
y
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual output is:  hahaha..hey, MERINA is my name. ok, female. where r u now JORDAN? malay?
The predicted output is:  hey i just got time do work to go to watch in my work to meet you are in her b
y it by this is your 6 yiyun money rush friends may buy first for us singapore building all
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

The Actual output is: Should be well rested. u free nxt wk
The predicted output is: on time i still got did you reach
>>>
>>>
The Actual output is: Eh. Take from where? Amk? How to go from yck?
The predicted output is: no lecture then stay in school
>>
>>>
The Actual output is: cannot la.. long hair not nice.. short hair nicer.. reali... haha.. no la.. still
fren..btw.. y dun wan to cut?haha..
The predicted output is: haha so what time how is at home from home and wait for your first right
>>
>>>
The Actual output is: ANGEL u fall slp liao izit y no response frm u.i still waitin for u ok.
The predicted output is: you finish with you later you tomorrow i have already i've got like to lose li
ke your first right then
>>
>>>
The Actual output is: Okie okie.. I'll update u bout e situation another day then.. Hee
The predicted output is: ok then i got time then you been meaningful you get seats this from you there
will tell you get it by this is at home this from town at home first who are your right please tomorrow
>>
>>>
The Actual output is: But i'm gettin fat sittin ard... I juz wan a change of environment lor... Try dif
f jobs...
The predicted output is: i can see you are you are you cut yet can give me my lesson sorry about me you
r headache now you the 12 then tell you girl you get me your parents then did you ok then where
>>
>>>
The Actual output is: Hey... So qiao... U oso learn drivin here...
The predicted output is: oh no dancing or do you cut you cut yet by the way can meet in the restaurant
really good and will save far thanks before buying our medical checkup that before before me before me r
ight before me
>>
>>>
The Actual output is: Can i confirm w ü e time tmr?
The predicted output is: you all where to orchard is at home now i just got time my questions free time
my questions got watch that one later thanks study hard to watch there by the way about at orchard is
here right
>>
>>
The Actual output is: Eh not meetg le...Other days den meet k. Jun goin home le...
The predicted output is: i can sit properly on her hehe because in her hehe in the university in this i
s it is it it with must have must have got must be if anything well but it's not sure got watch at
>>
>>
The Actual output is: Hello.Wakey wakey.>poke poke< at home ah?Watchin tv?
The predicted output is: hello how are your user very lonely
>>
>>
The Actual output is: Tireness draws across the mind making the body fade flexibility n soon windows of
soul begin 2 close N enter the dreamland! OYASUMINASAI! Sweet Dreams!
The predicted output is: same time then where did you i'm quite ok i'm like to see so later you
tomorrow the but ask you after 4 then later sorry to orchard ok the wait for mich tomorrow how are your
password
>>
>>
The Actual output is: Wat nus des? Ok lor, msg me again...
The predicted output is: you know what time will get me when you hope this the girl thanks so it
depends on the girl in this semester so how's your user id and find me before me right who else ok but c
an
>>
>>
The Actual output is: Crazy its v normal wat. He will do e same 4 audrey they all also.
The predicted output is: i'm not sure how to orchard is also the busy one is also sick so what time wil
l get the but after school then so sweet sure i'll still already can meet by the whole groceries and she
can't
>>
>>
The Actual output is: Okay... Ü call us when ü reach... My drivin is at 240 tmr... Ü have 2 lessons? Or
only one?
The predicted output is: hi baby it's not my questions so it from your user id and jun to orchard is
going my house then so you get by the restaurant really but call it's raining but tomorrow but tomorrow
later call me
>>
>>
The Actual output is: Yup... I dont mind... Why?

```
The Actual output is:  Yup... I dont mind... why.
The predicted output is:  haha okay i have already
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

In [95]:

```python
# 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 [96]:

```python
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:  0.19536971776642903
```

In [97]:

```python
bleu_scores_lst
```

Out[97]:

```
[0.4001601601922499,
 0,
 0,
 0.4001601601922499,
 0.47587330964125224,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0.47587330964125224,
 0.4001601601922499,
 0,
 0.47897362544357464,
 0.4001601601922499,
 0.4001601601922499,
 0.47587330964125224,
 0]
```