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, 89.7MB/s]
```

In [2]:

```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages (1.5.12)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from kaggle) (2021.5.30
)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages (from kaggle) (1.15.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from kaggle) (2.23.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packages (from kaggle) (5.
0.2)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from kaggle) (1.24.3)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packages (from kaggle) (2
.8.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from kaggle) (4.41.1)
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)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-packages (from python
-slugify->kaggle) (1.3)
```

In [3]:

```
!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 [4]:

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

```
7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,2 CPUs Intel(R) Xeon(R) CPU @
2.20GHz (406F0),ASM,AES-NI)

Scanning the drive for archives:
  0M Scan                          1 file, 1545551987 bytes (1474 MiB)

Extracting archive: fasttext-crawl-300d-2m.zip
--
Path = fasttext-crawl-300d-2m.zip
Type = zip
Physical Size = 1545551987

  0%
Would you like to replace the existing file:
  Path:     /content/crawl-300d-2M.vec
  Size:     4516698366 bytes (4308 MiB)
  Modified: 2019-09-27 20:43:22
with the file from archive:
  Path:     crawl-300d-2M.vec
  Size:     4516698366 bytes (4308 MiB)
  Modified: 2019-09-27 20:43:22
? (Y)es / (N)o / (A)lways / (S)kip all / A(u)to rename all / (Q)uit?
```

In [7]:

```
#Importing necessary libraries
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 [8]:

```
# 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')
```

```
Loading Fasttext Model
Done. 2000000  words loaded!
```

In [9]:

```
df=pd.read_csv('preprocessed_data.csv')#creating DataFrame using preprocessed_data.csv
```

In [10]:

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

Out[10]:

	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 [11]:

```
def preprocess(x):#removing last character
  x=x[:-1]
  return x
```

In [12]:

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

In [13]:

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

Out[13]:

	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 [14]:

```
df.shape
```

Out[14]:

```
(2000, 2)
```

In [15]:

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

In [16]:

```
df=df[df['source'].apply(length)<170]#removing the datapoints where the source sentence length is greater
df=df[df['target'].apply(length)<200]#removing the datapoints where the source sentence length is greater
```

In [17]:

```
df.shape
```

Out[17]:

```
(1990, 2)
```

In [18]:

```
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[18]:

	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 [19]:

```
df=df.drop('target',axis=1)#removing the target column
```

In [20]:

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

Out[20]:

	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 [21]:

```
from sklearn.model_selection import train_test_split
train, validation = train_test_split(df, test_size=0.01)#splitting the data in ratio 99:1
```

In [22]:

```
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 [23]:

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

In [24]:

```
vocab_size_target=len(tknizer_target.word_index.keys())#target vocab size
print(vocab_size_target)
vocab_size_source=len(tknizer_source.word_index.keys())#source vocab size
print(vocab_size_source)
```

```
3037
3703
```

In [25]:

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

Out[25]:

```
(1, 1447)
```

In [26]:

```
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 [27]:

```
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 [28]:

```
class Encoder(tf.keras.Model):
    '''
    Encoder model -- That takes a input sequence and returns encoder-outputs,encoder_final_state_h,encode
    '''

    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.lstm_state_h=0
        self.lstm_state_c=0
        self.embedding = tf.keras.layers.Embedding(input_dim=self.vocab_size, output_dim=self.embedding_s
                           mask_zero=True,name="embedding_layer_encoder",weights=[encoder_embedding_matr
        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 ouput to encode
          returns -- encoder_output, last time step's 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 [3
      '''
      return [np.zeros((batch_size,self.lstm_size)),np.zeros((batch_size,self.lstm_size))]
```

In [29]:

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

    def __init__(self,out_vocab_size,embedding_size,lstm_size,input_length):

        #Initialize Embedding layer
        #Intialize Decoder LSTM layer
        super().__init__()
        self.out_vocab_size = out_vocab_size
        self.embedding_size = embedding_size
        self.lstm_size = lstm_size
        self.input_length = input_length
        # we are using embedding_matrix and not training the embedding layer
        self.embedding = tf.keras.layers.Embedding(input_dim=self.out_vocab_size, output_dim=self.embeddi
                           mask_zero=True, name="embedding_layer_decoder",weights=[decoder_embedding_mat
        self.lstm = tf.keras.layers.LSTM(self.lstm_size, return_sequences=True, return_state=True, name="

    def call(self,input_sequence,initial_states):
        '''
          This function takes a sequence input and the initial states of the encoder.
          Pass the input_sequence input to the Embedding layer, Pass the embedding layer ouput to decode

          returns -- decoder_output,decoder_final_state_h,decoder_final_state_c
        '''
        target_embedd = self.embedding(input_sequence)
        decoder_output,decoder_final_state_h,decoder_final_state_c = self.lstm(target_embedd, initial_sta
        return decoder_output,decoder_final_state_h,decoder_final_state_c
```

In [30]:

```
class Encoder_decoder(tf.keras.Model):

    def __init__(self,encoder_inputs_length,decoder_inputs_length, output_vocab_size,batch_size):

        #Create encoder object
        #Create decoder object
        #Intialize Dense layer(out_vocab_size) with activation='softmax'
        super().__init__() # https://stackoverflow.com/a/27134600/4084039
        self.batch_size=batch_size
        self.encoder = Encoder(vocab_size_source+1,300,128,encoder_inputs_length)
        self.decoder = Decoder(vocab_size_target+1,300,128,decoder_inputs_length)
        self.dense   = tf.keras.layers.Dense(output_vocab_size, activation='softmax')


    def call(self,data):
        '''
        A. Pass the input sequence to Encoder layer -- Return encoder_output,encoder_final_state_h,encode
        B. Pass the target sequence to Decoder layer with intial states as encoder_final_state_h,encoder_
        C. Pass the decoder_outputs into Dense layer

        Return decoder_outputs
        '''
        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,decoder_final_state_h,decoder_final_state_c= self.decoder(output,[encoder_h, encod
        output                               = self.dense(decoder_output)
        return output
```

In [31]:

```
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 [32]:

```
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 [33]:

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

```
WARNING:tensorflow:From <ipython-input-33-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 [34]:

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

In [35]:

```
#Create an object of encoder_decoder Model class,
# Compile the model and fit the model
model  = Encoder_decoder(encoder_inputs_length=39,decoder_inputs_length=43,output_vocab_size=vocab_size_t
optimizer = tf.keras.optimizers.Adam(0.01)
model.compile(optimizer=optimizer,loss='sparse_categorical_crossentropy',metrics=['accuracy'])
train_steps=train.shape[0]//512
valid_steps=validation.shape[0]//20
model.fit_generator(train_dataloader, steps_per_epoch=train_steps, epochs=40, validation_data=test_datalo
model.summary()
```

```
/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 1/40
3/3 [==============================] - 1s 190ms/step - loss: 2.7104 - accuracy: 0.0480 - val_loss:
1.8951 - val_accuracy: 0.0794
Epoch 2/40
3/3 [==============================] - 1s 174ms/step - loss: 2.2126 - accuracy: 0.0686 - val_loss:
1.6150 - val_accuracy: 0.0833
Epoch 3/40
3/3 [==============================] - 1s 158ms/step - loss: 2.1212 - accuracy: 0.0788 - val_loss:
1.5956 - val_accuracy: 0.0913
Epoch 4/40
3/3 [==============================] - 1s 156ms/step - loss: 2.0691 - accuracy: 0.0857 - val_loss:
1.5894 - val_accuracy: 0.1071
Epoch 5/40
3/3 [==============================] - 1s 156ms/step - loss: 2.0377 - accuracy: 0.0935 - val_loss:
1.5779 - val_accuracy: 0.0992
Epoch 6/40
3/3 [==============================] - 1s 162ms/step - loss: 2.0195 - accuracy: 0.0941 - val_loss:
1.5682 - val_accuracy: 0.1071
Epoch 7/40
3/3 [==============================] - 1s 163ms/step - loss: 1.9993 - accuracy: 0.0988 - val_loss:
1.5554 - val_accuracy: 0.1071
Epoch 8/40
3/3 [==============================] - 1s 154ms/step - loss: 1.9775 - accuracy: 0.1074 - val_loss:
1.5461 - val_accuracy: 0.1071
Epoch 9/40
3/3 [==============================] - 1s 154ms/step - loss: 1.9560 - accuracy: 0.1099 - val_loss:
1.5306 - val_accuracy: 0.1071
Epoch 10/40
3/3 [==============================] - 1s 154ms/step - loss: 1.9366 - accuracy: 0.1152 - val_loss:
1.5196 - val_accuracy: 0.1151
Epoch 11/40
3/3 [==============================] - 1s 153ms/step - loss: 1.9129 - accuracy: 0.1230 - val_loss:
1.5068 - val_accuracy: 0.1310
```

```
Epoch 12/40
3/3 [==============================] - 1s 155ms/step - loss: 1.8937 - accuracy: 0.1250 - val_loss:
1.4931 - val_accuracy: 0.1349
Epoch 13/40
3/3 [==============================] - 1s 156ms/step - loss: 1.8701 - accuracy: 0.1321 - val_loss:
1.4781 - val_accuracy: 0.1429
Epoch 14/40
3/3 [==============================] - 1s 153ms/step - loss: 1.8460 - accuracy: 0.1379 - val_loss:
1.4648 - val_accuracy: 0.1508
Epoch 15/40
3/3 [==============================] - 1s 155ms/step - loss: 1.8223 - accuracy: 0.1407 - val_loss:
1.4581 - val_accuracy: 0.1508
Epoch 16/40
3/3 [==============================] - 1s 154ms/step - loss: 1.7985 - accuracy: 0.1443 - val_loss:
1.4456 - val_accuracy: 0.1508
Epoch 17/40
3/3 [==============================] - 1s 152ms/step - loss: 1.7770 - accuracy: 0.1483 - val_loss:
1.4381 - val_accuracy: 0.1548
Epoch 18/40
3/3 [==============================] - 1s 158ms/step - loss: 1.7559 - accuracy: 0.1503 - val_loss:
1.4315 - val_accuracy: 0.1587
Epoch 19/40
3/3 [==============================] - 1s 158ms/step - loss: 1.7326 - accuracy: 0.1544 - val_loss:
1.4174 - val_accuracy: 0.1627
Epoch 20/40
3/3 [==============================] - 1s 155ms/step - loss: 1.7083 - accuracy: 0.1573 - val_loss:
1.4157 - val_accuracy: 0.1706
Epoch 21/40
3/3 [==============================] - 1s 153ms/step - loss: 1.6890 - accuracy: 0.1622 - val_loss:
1.4114 - val_accuracy: 0.1746
Epoch 22/40
3/3 [==============================] - 1s 155ms/step - loss: 1.6656 - accuracy: 0.1671 - val_loss:
1.4038 - val_accuracy: 0.1746
Epoch 23/40
3/3 [==============================] - 1s 155ms/step - loss: 1.6411 - accuracy: 0.1704 - val_loss:
1.3938 - val_accuracy: 0.1786
Epoch 24/40
3/3 [==============================] - 1s 158ms/step - loss: 1.6217 - accuracy: 0.1729 - val_loss:
1.3865 - val_accuracy: 0.1865
Epoch 25/40
3/3 [==============================] - 1s 156ms/step - loss: 1.5984 - accuracy: 0.1767 - val_loss:
1.3786 - val_accuracy: 0.1786
Epoch 26/40
3/3 [==============================] - 1s 154ms/step - loss: 1.5754 - accuracy: 0.1793 - val_loss:
1.3770 - val_accuracy: 0.1825
Epoch 27/40
3/3 [==============================] - 1s 159ms/step - loss: 1.5548 - accuracy: 0.1835 - val_loss:
1.3704 - val_accuracy: 0.1786
Epoch 28/40
3/3 [==============================] - 1s 156ms/step - loss: 1.5350 - accuracy: 0.1871 - val_loss:
1.3655 - val_accuracy: 0.1825
Epoch 29/40
3/3 [==============================] - 1s 156ms/step - loss: 1.5126 - accuracy: 0.1907 - val_loss:
1.3592 - val_accuracy: 0.2024
Epoch 30/40
3/3 [==============================] - 1s 160ms/step - loss: 1.4912 - accuracy: 0.1950 - val_loss:
1.3575 - val_accuracy: 0.1865
Epoch 31/40
3/3 [==============================] - 1s 155ms/step - loss: 1.4684 - accuracy: 0.1984 - val_loss:
1.3488 - val_accuracy: 0.1905
Epoch 32/40
3/3 [==============================] - 1s 166ms/step - loss: 1.4476 - accuracy: 0.2053 - val_loss:
1.3444 - val_accuracy: 0.1984
Epoch 33/40
3/3 [==============================] - 1s 156ms/step - loss: 1.4318 - accuracy: 0.2081 - val_loss:
1.3375 - val_accuracy: 0.2063
Epoch 34/40
3/3 [==============================] - 1s 155ms/step - loss: 1.4081 - accuracy: 0.2126 - val_loss:
1.3277 - val_accuracy: 0.2024
Epoch 35/40
3/3 [==============================] - 1s 155ms/step - loss: 1.3885 - accuracy: 0.2192 - val_loss:
1.3305 - val_accuracy: 0.1984
Epoch 36/40
3/3 [==============================] - 1s 155ms/step - loss: 1.3665 - accuracy: 0.2229 - val_loss:
1.3255 - val_accuracy: 0.1984
Epoch 37/40
3/3 [==============================] - 1s 154ms/step - loss: 1.3456 - accuracy: 0.2299 - val_loss:
```

```
1.3225 - val_accuracy: 0.2063
Epoch 38/40
3/3 [==============================] - 1s 156ms/step - loss: 1.3283 - accuracy: 0.2342 - val_loss:
1.3228 - val_accuracy: 0.2063
Epoch 39/40
3/3 [==============================] - 1s 153ms/step - loss: 1.3114 - accuracy: 0.2372 - val_loss:
1.3169 - val_accuracy: 0.1984
Epoch 40/40
3/3 [==============================] - 1s 159ms/step - loss: 1.2887 - accuracy: 0.2447 - val_loss:
1.3125 - val_accuracy: 0.2103
Model: "encoder_decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder (Encoder)            multiple                  1330848
_________________________________________________________________
decoder (Decoder)            multiple                  1131048
_________________________________________________________________
dense (Dense)                multiple                  391773
=================================================================
Total params: 2,853,669
Trainable params: 831,069
Non-trainable params: 2,022,600
_________________________________________________________________
```

In [36]:

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

In [37]:

```
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 decode
  D. till we reach max_length of decoder or till the model predicted word <end>:
         predicted_out,state_h,state_c=model.layers[1](dec_input,states)
         pass the predicted_out to the dense layer
         update the states=[state_h,state_c]
         And get the index of the wordcc with maximum probability of the dense layer output, using the to
         Update the input_to_decoder with current predictions
  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):
    infe_output, state_h, state_c = model.layers[1](cur_vec,[state_h,state_c])
    infe_output = model.layers[2](infe_output)
    cur_vec = np.reshape(np.argmax(infe_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 [38]:

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

Out[38]:

```
1266          <start> PJ. You're a Malay or Chinese, Rin?
1469                  <start> Help me look out for tubes.
769     <start> My sister and I are eating breakfast. ...
1800                 <start> Make that 3! For God's sake!
1064                <start> Are you all coming to school?
1622      <start> Hee. Ok. See you another time. Big hug.
1695    <start> Yes, I thought of it also but scare me...
1675    <start> Are you going to send a mail? Tomorrow...
1613                               <start> 26th of July.
1726                <start> Today mango got 10% discount.
821     <start> Don't ask. It is for a stupid reason. ...
1227    <start> Okay. But Tuesday I've got dinner. So ...
1286    <start> Oh. I see I see. I don't know. Message...
785     <start> Haha. I'm going to buy sandals. How to...
1107    <start> Ok, thanks. So do you think they would...
282     <start> XY and I are meeting for dinner. I'm i...
1151                                   <start> Yes. You?
1252    <start> Hey hey, you are invited to my place t...
1267    <start> Oh my, why is she like that? Is she ve...
1323    <start> Thanks for the goodies! They taste rea...
Name: target_in, dtype: object
```

In [39]:

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

```
The predicted output is:  i am still going to introduce
The predicted output is:  i think i will be late
The predicted output is:  i am going to go to the place to you
The predicted output is:  i will be late
The predicted output is:  i am going to chat
The predicted output is:  hi care to chat with you
The predicted output is:  hey i don't know i will be able to go out for the way
The predicted output is:  i don't know i will be able to go out
The predicted output is:  ok
The predicted output is:  no need to go
The predicted output is:  i don't know i will be able to go out for the way i don't know to go to go
The predicted output is:  hey i don't know i will be able to go out for the way i think i think i think
i will go to go to go to go
The predicted output is:  hey i am going to go to watch
The predicted output is:  haha ok i don't know i will be able to go out for the place
The predicted output is:  hey i don't know i will be late
The predicted output is:  i don't know i will be able to go out for the way i am not going to go
The predicted output is:  ok
The predicted output is:  hey i don't know i don't know i will be late late i am not going to see you
The predicted output is:  hey i don't know i will be able to go out for the way i think i think i will b
e 35
The predicted output is:  hey i don't know i will be able to go out for the way
```

In [40]:

```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)
/usr/local/lib/python3.7/dist-packages/nltk/translate/bleu_score.py:490: UserWarning:
Corpus/Sentence contains 0 counts of 3-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)
```

In [43]:

```python
average_bleu_scores=sum(bleu_scores_lst)/len(bleu_scores_lst)
print("Average BLEU score of these 20 test data sentences is: ",average_bleu_scores)
```

```
Average BLEU score of these 20 test data sentences is:  0.17936797861081147
```

In [42]:

```
bleu_scores_lst
```

Out[42]:

```
[0,
 0,
 0,
 0,
 0.668740304976422,
 0,
 0.5724063666159062,
 0.6051012508914458,
 0,
 0,
 0.4728708045015879,
 0.43092381945890607,
 0,
 0.23927141250362965,
 0,
 0,
 0,
 0.5980456132683322,
 0,
 0]
```

Character_Level:

In [11]:

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

Out[11]:

	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 [12]:

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

In [13]:

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

In [14]:

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

Out[14]:

	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 [15]:

```
df.shape
```

Out[15]:

```
(2000, 2)
```

In [16]:

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

In [17]:

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

In [18]:

```
df.shape
```

Out[18]:

```
(1990, 2)
```

In [19]:

```
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[19]:

	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 [20]:

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

In [21]:

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

Out[21]:

	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 [22]:

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

In [23]:

```
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'
```

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

In [24]:

```
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 [25]:

```
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)
```

```
91
101
```

In [26]:

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

Out[26]:

```
(20, 85)
```

In [48]:

```
class Encoder(tf.keras.Model):
    '''
    Encoder model -- That takes a input sequence and returns encoder-outputs,encoder_final_state_h,encode
    '''

    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.lstm_state_h=0
        self.lstm_state_c=0
        self.embedding = tf.keras.layers.Embedding(input_dim=self.vocab_size, output_dim=self.embedding_s
                           mask_zero=True, name="embedding_layer_encoder",trainable=True)
        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 ouput to encode
          returns -- encoder_output, last time step's 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 [3
      '''
      return [np.zeros((batch_size,self.lstm_size)),np.zeros((batch_size,self.lstm_size))]
```

In [50]:

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

    def __init__(self,out_vocab_size,embedding_size,lstm_size,input_length):

        #Initialize Embedding layer
        #Intialize Decoder LSTM layer
        super().__init__()
        self.out_vocab_size = out_vocab_size
        self.embedding_size = embedding_size
        self.lstm_size = lstm_size
        self.input_length = input_length
        # we are using embedding_matrix and not training the embedding layer
        self.embedding = tf.keras.layers.Embedding(input_dim=self.out_vocab_size, output_dim=self.embeddi
                           mask_zero=True, name="embedding_layer_decoder", trainable=True)
        self.lstm = tf.keras.layers.LSTM(self.lstm_size, return_sequences=True, return_state=True, name="

    def call(self,input_sequence,initial_states):
        '''
          This function takes a sequence input and the initial states of the encoder.
          Pass the input_sequence input to the Embedding layer, Pass the embedding layer ouput to decode

          returns -- decoder_output,decoder_final_state_h,decoder_final_state_c
        '''
        target_embedd = self.embedding(input_sequence)
        decoder_output,decoder_final_state_h,decoder_final_state_c = self.lstm(target_embedd, initial_sta
        return decoder_output,decoder_final_state_h,decoder_final_state_c
```

In [53]:

```python
class Encoder_decoder(tf.keras.Model):

    def __init__(self,encoder_inputs_length,decoder_inputs_length, output_vocab_size,batch_size):

        #Create encoder object
        #Create decoder object
        #Intialize Dense layer(out_vocab_size) with activation='softmax'
        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+1,300,100,decoder_inputs_length)
        self.dense   = tf.keras.layers.Dense(output_vocab_size, activation='softmax')


    def call(self,data):
        '''
        A. Pass the input sequence to Encoder layer -- Return encoder_output,encoder_final_state_h,encode
        B. Pass the target sequence to Decoder layer with intial states as encoder_final_state_h,encoder_
        C. Pass the decoder_outputs into Dense layer

        Return decoder_outputs
        '''
        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,decoder_final_state_h,decoder_final_state_c= self.decoder(output,[encoder_h, encod
        output                                = self.dense(decoder_output)
        return output
```

In [54]:

```python
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
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='sparse_categorical_crossentropy',metrics=['accuracy'])
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
model.summary()
```

```
/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 [==============================] - 1s 348ms/step - loss: 1.4916 - accuracy: 0.1226 - val_loss:
1.0614 - val_accuracy: 0.1744
Epoch 2/100
3/3 [==============================] - 1s 341ms/step - loss: 1.1787 - accuracy: 0.1884 - val_loss:
1.0143 - val_accuracy: 0.1744
Epoch 3/100
3/3 [==============================] - 1s 342ms/step - loss: 1.1388 - accuracy: 0.1823 - val_loss:
0.9851 - val_accuracy: 0.1776
Epoch 4/100
3/3 [==============================] - 1s 332ms/step - loss: 1.1065 - accuracy: 0.1901 - val_loss:
0.9493 - val_accuracy: 0.2036
Epoch 5/100
3/3 [==============================] - 1s 338ms/step - loss: 1.0596 - accuracy: 0.2329 - val_loss:
0.9110 - val_accuracy: 0.2466
Epoch 6/100
3/3 [==============================] - 1s 330ms/step - loss: 1.0142 - accuracy: 0.2621 - val_loss:
0.8720 - val_accuracy: 0.2620
Epoch 7/100
3/3 [==============================] - 1s 330ms/step - loss: 0.9679 - accuracy: 0.2843 - val_loss:
0.8407 - val_accuracy: 0.2960
Epoch 8/100
3/3 [==============================] - 1s 330ms/step - loss: 0.9318 - accuracy: 0.2999 - val_loss:
0.8208 - val_accuracy: 0.2952
Epoch 9/100
3/3 [==============================] - 1s 320ms/step - loss: 0.9056 - accuracy: 0.3050 - val_loss:
0.8032 - val_accuracy: 0.2952
Epoch 10/100
3/3 [==============================] - 1s 323ms/step - loss: 0.8851 - accuracy: 0.3097 - val_loss:
0.7873 - val_accuracy: 0.3082
Epoch 11/100
3/3 [==============================] - 1s 335ms/step - loss: 0.8671 - accuracy: 0.3281 - val_loss:
0.7721 - val_accuracy: 0.3204
Epoch 12/100
3/3 [==============================] - 1s 323ms/step - loss: 0.8507 - accuracy: 0.3348 - val_loss:
0.7567 - val_accuracy: 0.3236
Epoch 13/100
3/3 [==============================] - 1s 330ms/step - loss: 0.8355 - accuracy: 0.3404 - val_loss:
0.7452 - val_accuracy: 0.3301
Epoch 14/100
3/3 [==============================] - 1s 332ms/step - loss: 0.8214 - accuracy: 0.3482 - val_loss:
0.7328 - val_accuracy: 0.3447
```

```
Epoch 15/100
3/3 [==============================] - 1s 331ms/step - loss: 0.8085 - accuracy: 0.3593 - val_loss:
0.7227 - val_accuracy: 0.3439
Epoch 16/100
3/3 [==============================] - 1s 334ms/step - loss: 0.7969 - accuracy: 0.3653 - val_loss:
0.7145 - val_accuracy: 0.3536
Epoch 17/100
3/3 [==============================] - 1s 340ms/step - loss: 0.7862 - accuracy: 0.3718 - val_loss:
0.7059 - val_accuracy: 0.3642
Epoch 18/100
3/3 [==============================] - 1s 330ms/step - loss: 0.7761 - accuracy: 0.3820 - val_loss:
0.6978 - val_accuracy: 0.3715
Epoch 19/100
3/3 [==============================] - 1s 326ms/step - loss: 0.7662 - accuracy: 0.3865 - val_loss:
0.6915 - val_accuracy: 0.3698
Epoch 20/100
3/3 [==============================] - 1s 336ms/step - loss: 0.7572 - accuracy: 0.3901 - val_loss:
0.6842 - val_accuracy: 0.3771
Epoch 21/100
3/3 [==============================] - 1s 327ms/step - loss: 0.7483 - accuracy: 0.3978 - val_loss:
0.6780 - val_accuracy: 0.3788
Epoch 22/100
3/3 [==============================] - 1s 331ms/step - loss: 0.7399 - accuracy: 0.4009 - val_loss:
0.6714 - val_accuracy: 0.3893
Epoch 23/100
3/3 [==============================] - 1s 327ms/step - loss: 0.7318 - accuracy: 0.4066 - val_loss:
0.6652 - val_accuracy: 0.3917
Epoch 24/100
3/3 [==============================] - 1s 336ms/step - loss: 0.7237 - accuracy: 0.4116 - val_loss:
0.6582 - val_accuracy: 0.4006
Epoch 25/100
3/3 [==============================] - 1s 313ms/step - loss: 0.7162 - accuracy: 0.4175 - val_loss:
0.6519 - val_accuracy: 0.3958
Epoch 26/100
3/3 [==============================] - 1s 326ms/step - loss: 0.7083 - accuracy: 0.4243 - val_loss:
0.6453 - val_accuracy: 0.4006
Epoch 27/100
3/3 [==============================] - 1s 331ms/step - loss: 0.7016 - accuracy: 0.4272 - val_loss:
0.6392 - val_accuracy: 0.4039
Epoch 28/100
3/3 [==============================] - 1s 341ms/step - loss: 0.6950 - accuracy: 0.4314 - val_loss:
0.6360 - val_accuracy: 0.4112
Epoch 29/100
3/3 [==============================] - 1s 337ms/step - loss: 0.6886 - accuracy: 0.4386 - val_loss:
0.6301 - val_accuracy: 0.4128
Epoch 30/100
3/3 [==============================] - 1s 335ms/step - loss: 0.6824 - accuracy: 0.4435 - val_loss:
0.6247 - val_accuracy: 0.4193
Epoch 31/100
3/3 [==============================] - 1s 328ms/step - loss: 0.6766 - accuracy: 0.4463 - val_loss:
0.6205 - val_accuracy: 0.4217
Epoch 32/100
3/3 [==============================] - 1s 330ms/step - loss: 0.6715 - accuracy: 0.4504 - val_loss:
0.6148 - val_accuracy: 0.4225
Epoch 33/100
3/3 [==============================] - 1s 337ms/step - loss: 0.6659 - accuracy: 0.4551 - val_loss:
0.6117 - val_accuracy: 0.4274
Epoch 34/100
3/3 [==============================] - 1s 338ms/step - loss: 0.6609 - accuracy: 0.4594 - val_loss:
0.6070 - val_accuracy: 0.4242
Epoch 35/100
3/3 [==============================] - 1s 330ms/step - loss: 0.6555 - accuracy: 0.4644 - val_loss:
0.6031 - val_accuracy: 0.4404
Epoch 36/100
3/3 [==============================] - 1s 333ms/step - loss: 0.6507 - accuracy: 0.4677 - val_loss:
0.6006 - val_accuracy: 0.4363
Epoch 37/100
3/3 [==============================] - 1s 323ms/step - loss: 0.6464 - accuracy: 0.4707 - val_loss:
0.5968 - val_accuracy: 0.4331
Epoch 38/100
3/3 [==============================] - 1s 328ms/step - loss: 0.6422 - accuracy: 0.4739 - val_loss:
0.5943 - val_accuracy: 0.4371
Epoch 39/100
3/3 [==============================] - 1s 330ms/step - loss: 0.6382 - accuracy: 0.4781 - val_loss:
0.5930 - val_accuracy: 0.4453
Epoch 40/100
3/3 [==============================] - 1s 325ms/step - loss: 0.6342 - accuracy: 0.4797 - val_loss:
```

```
0.5899 - val_accuracy: 0.4396
Epoch 41/100
3/3 [==============================] - 1s 329ms/step - loss: 0.6300 - accuracy: 0.4843 - val_loss:
0.5866 - val_accuracy: 0.4485
Epoch 42/100
3/3 [==============================] - 1s 325ms/step - loss: 0.6263 - accuracy: 0.4878 - val_loss:
0.5839 - val_accuracy: 0.4469
Epoch 43/100
3/3 [==============================] - 1s 325ms/step - loss: 0.6221 - accuracy: 0.4913 - val_loss:
0.5809 - val_accuracy: 0.4558
Epoch 44/100
3/3 [==============================] - 1s 330ms/step - loss: 0.6187 - accuracy: 0.4937 - val_loss:
0.5791 - val_accuracy: 0.4590
Epoch 45/100
3/3 [==============================] - 1s 337ms/step - loss: 0.6153 - accuracy: 0.4975 - val_loss:
0.5760 - val_accuracy: 0.4623
Epoch 46/100
3/3 [==============================] - 1s 330ms/step - loss: 0.6122 - accuracy: 0.4993 - val_loss:
0.5754 - val_accuracy: 0.4688
Epoch 47/100
3/3 [==============================] - 1s 331ms/step - loss: 0.6093 - accuracy: 0.5002 - val_loss:
0.5729 - val_accuracy: 0.4639
Epoch 48/100
3/3 [==============================] - 1s 335ms/step - loss: 0.6058 - accuracy: 0.5028 - val_loss:
0.5707 - val_accuracy: 0.4663
Epoch 49/100
3/3 [==============================] - 1s 314ms/step - loss: 0.6032 - accuracy: 0.5049 - val_loss:
0.5684 - val_accuracy: 0.4720
Epoch 50/100
3/3 [==============================] - 1s 321ms/step - loss: 0.6005 - accuracy: 0.5077 - val_loss:
0.5670 - val_accuracy: 0.4761
Epoch 51/100
3/3 [==============================] - 1s 320ms/step - loss: 0.5977 - accuracy: 0.5086 - val_loss:
0.5649 - val_accuracy: 0.4753
Epoch 52/100
3/3 [==============================] - 1s 334ms/step - loss: 0.5949 - accuracy: 0.5115 - val_loss:
0.5635 - val_accuracy: 0.4801
Epoch 53/100
3/3 [==============================] - 1s 316ms/step - loss: 0.5922 - accuracy: 0.5123 - val_loss:
0.5631 - val_accuracy: 0.4745
Epoch 54/100
3/3 [==============================] - 1s 324ms/step - loss: 0.5900 - accuracy: 0.5138 - val_loss:
0.5628 - val_accuracy: 0.4801
Epoch 55/100
3/3 [==============================] - 1s 334ms/step - loss: 0.5870 - accuracy: 0.5169 - val_loss:
0.5593 - val_accuracy: 0.4874
Epoch 56/100
3/3 [==============================] - 1s 344ms/step - loss: 0.5847 - accuracy: 0.5187 - val_loss:
0.5575 - val_accuracy: 0.4834
Epoch 57/100
3/3 [==============================] - 1s 341ms/step - loss: 0.5820 - accuracy: 0.5206 - val_loss:
0.5566 - val_accuracy: 0.4907
Epoch 58/100
3/3 [==============================] - 1s 326ms/step - loss: 0.5801 - accuracy: 0.5225 - val_loss:
0.5552 - val_accuracy: 0.4891
Epoch 59/100
3/3 [==============================] - 1s 344ms/step - loss: 0.5773 - accuracy: 0.5243 - val_loss:
0.5529 - val_accuracy: 0.4915
Epoch 60/100
3/3 [==============================] - 1s 330ms/step - loss: 0.5757 - accuracy: 0.5261 - val_loss:
0.5519 - val_accuracy: 0.4923
Epoch 61/100
3/3 [==============================] - 1s 329ms/step - loss: 0.5735 - accuracy: 0.5274 - val_loss:
0.5520 - val_accuracy: 0.4923
Epoch 62/100
3/3 [==============================] - 1s 324ms/step - loss: 0.5714 - accuracy: 0.5295 - val_loss:
0.5487 - val_accuracy: 0.5077
Epoch 63/100
3/3 [==============================] - 1s 345ms/step - loss: 0.5691 - accuracy: 0.5314 - val_loss:
0.5475 - val_accuracy: 0.4972
Epoch 64/100
3/3 [==============================] - 1s 335ms/step - loss: 0.5673 - accuracy: 0.5326 - val_loss:
0.5464 - val_accuracy: 0.5077
Epoch 65/100
3/3 [==============================] - 1s 329ms/step - loss: 0.5647 - accuracy: 0.5347 - val_loss:
0.5465 - val_accuracy: 0.5061
Epoch 66/100
Epoch 66/100
3/3 [==============================] - 1s 328ms/step - loss: 0.5631 - accuracy: 0.5356 - val_loss:
0.5466 - val_accuracy: 0.5069
Epoch 67/100
3/3 [==============================] - 1s 337ms/step - loss: 0.5615 - accuracy: 0.5368 - val_loss:
0.5437 - val_accuracy: 0.5061
Epoch 68/100
3/3 [==============================] - 1s 325ms/step - loss: 0.5593 - accuracy: 0.5391 - val_loss:
0.5416 - val_accuracy: 0.5118
Epoch 69/100
3/3 [==============================] - 1s 326ms/step - loss: 0.5576 - accuracy: 0.5407 - val_loss:
0.5412 - val_accuracy: 0.5061
Epoch 70/100
3/3 [==============================] - 1s 322ms/step - loss: 0.5554 - accuracy: 0.5417 - val_loss:
0.5389 - val_accuracy: 0.5101
Epoch 71/100
3/3 [==============================] - 1s 319ms/step - loss: 0.5537 - accuracy: 0.5434 - val_loss:
0.5380 - val_accuracy: 0.5109
Epoch 72/100
3/3 [==============================] - 1s 324ms/step - loss: 0.5521 - accuracy: 0.5448 - val_loss:
0.5376 - val_accuracy: 0.5126
Epoch 73/100
3/3 [==============================] - 1s 316ms/step - loss: 0.5501 - accuracy: 0.5470 - val_loss:
0.5368 - val_accuracy: 0.5101
Epoch 74/100
3/3 [==============================] - 1s 333ms/step - loss: 0.5478 - accuracy: 0.5486 - val_loss:
0.5335 - val_accuracy: 0.5134
Epoch 75/100
3/3 [==============================] - 1s 329ms/step - loss: 0.5454 - accuracy: 0.5500 - val_loss:
0.5337 - val_accuracy: 0.5199
Epoch 76/100
3/3 [==============================] - 1s 330ms/step - loss: 0.5442 - accuracy: 0.5516 - val_loss:
0.5329 - val_accuracy: 0.5191
Epoch 77/100
3/3 [==============================] - 1s 334ms/step - loss: 0.5421 - accuracy: 0.5530 - val_loss:
0.5296 - val_accuracy: 0.5223
Epoch 78/100
3/3 [==============================] - 1s 335ms/step - loss: 0.5404 - accuracy: 0.5544 - val_loss:
0.5298 - val_accuracy: 0.5255
Epoch 79/100
3/3 [==============================] - 1s 332ms/step - loss: 0.5391 - accuracy: 0.5552 - val_loss:
0.5290 - val_accuracy: 0.5215
Epoch 80/100
3/3 [==============================] - 1s 339ms/step - loss: 0.5371 - accuracy: 0.5566 - val_loss:
0.5262 - val_accuracy: 0.5280
Epoch 81/100
3/3 [==============================] - 1s 328ms/step - loss: 0.5353 - accuracy: 0.5576 - val_loss:
0.5277 - val_accuracy: 0.5272
Epoch 82/100
3/3 [==============================] - 1s 326ms/step - loss: 0.5341 - accuracy: 0.5593 - val_loss:
0.5241 - val_accuracy: 0.5280
Epoch 83/100
3/3 [==============================] - 1s 330ms/step - loss: 0.5316 - accuracy: 0.5605 - val_loss:
0.5231 - val_accuracy: 0.5361
Epoch 84/100
3/3 [==============================] - 1s 336ms/step - loss: 0.5301 - accuracy: 0.5627 - val_loss:
0.5212 - val_accuracy: 0.5369
Epoch 85/100
3/3 [==============================] - 1s 344ms/step - loss: 0.5279 - accuracy: 0.5633 - val_loss:
0.5198 - val_accuracy: 0.5320
Epoch 86/100
3/3 [==============================] - 1s 330ms/step - loss: 0.5268 - accuracy: 0.5645 - val_loss:
0.5211 - val_accuracy: 0.5264
Epoch 87/100
3/3 [==============================] - 1s 342ms/step - loss: 0.5249 - accuracy: 0.5658 - val_loss:
0.5199 - val_accuracy: 0.5377
Epoch 88/100
3/3 [==============================] - 1s 344ms/step - loss: 0.5239 - accuracy: 0.5660 - val_loss:
0.5180 - val_accuracy: 0.5385
Epoch 89/100
3/3 [==============================] - 1s 335ms/step - loss: 0.5222 - accuracy: 0.5679 - val_loss:
0.5167 - val_accuracy: 0.5377
Epoch 90/100
3/3 [==============================] - 1s 331ms/step - loss: 0.5206 - accuracy: 0.5696 - val_loss:
0.5181 - val_accuracy: 0.5353
Epoch 91/100
3/3 [==============================] - 1s 336ms/step - loss: 0.5192 - accuracy: 0.5697 - val_loss:
0.5157 - val_accuracy: 0.5353
```

```
Epoch 92/100
3/3 [==============================] - 1s 328ms/step - loss: 0.5171 - accuracy: 0.5709 - val_loss:
0.5174 - val_accuracy: 0.5369
Epoch 93/100
3/3 [==============================] - 1s 325ms/step - loss: 0.5162 - accuracy: 0.5722 - val_loss:
0.5170 - val_accuracy: 0.5377
Epoch 94/100
3/3 [==============================] - 1s 338ms/step - loss: 0.5142 - accuracy: 0.5745 - val_loss:
0.5153 - val_accuracy: 0.5369
Epoch 95/100
3/3 [==============================] - 1s 333ms/step - loss: 0.5122 - accuracy: 0.5761 - val_loss:
0.5129 - val_accuracy: 0.5418
Epoch 96/100
3/3 [==============================] - 1s 331ms/step - loss: 0.5108 - accuracy: 0.5773 - val_loss:
0.5131 - val_accuracy: 0.5458
Epoch 97/100
3/3 [==============================] - 1s 315ms/step - loss: 0.5086 - accuracy: 0.5787 - val_loss:
0.5143 - val_accuracy: 0.5401
Epoch 98/100
3/3 [==============================] - 1s 332ms/step - loss: 0.5076 - accuracy: 0.5793 - val_loss:
0.5122 - val_accuracy: 0.5483
Epoch 99/100
3/3 [==============================] - 1s 331ms/step - loss: 0.5064 - accuracy: 0.5798 - val_loss:
0.5124 - val_accuracy: 0.5377
Epoch 100/100
3/3 [==============================] - 1s 338ms/step - loss: 0.5043 - accuracy: 0.5827 - val_loss:
0.5106 - val_accuracy: 0.5442
Model: "encoder_decoder_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_2 (Encoder)          multiple                  191000
_________________________________________________________________
decoder_2 (Decoder)          multiple                  188000
_________________________________________________________________
dense_2 (Dense)              multiple                  9191
=================================================================
Total params: 388,191
Trainable params: 388,191
Non-trainable params: 0
_________________________________________________________________
```

In [59]:

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

In [60]:

```python
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 decode
  D. till we reach max_length of decoder or till the model predicted word <end>:
         predicted_out,state_h,state_c=model.layers[1](dec_input,states)
         pass the predicted_out to the dense layer
         update the states=[state_h,state_c]
         And get the index of the wordcc with maximum probability of the dense layer output, using the to
         Update the input_to_decoder with current predictions
  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 200 is the max_length of the sequence
  for i in range(200):
    infe_output, state_h, state_c = model.layers[1](cur_vec,[state_h,state_c])
    infe_output = model.layers[2](infe_output)
    cur_vec = np.reshape(np.argmax(infe_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 [69]:

```
 for i in validation['source']:
    print("The Actual Output is:")
    print(i)
    print("The Predicted Output is:")
    pred=predict(i)
    print(pred)
    print('>'*100)
```

```
The Actual Output is:
Y sad sad
The Predicted Output is:
Yes. I am not to meet you are to meet you are to meet you are to meet you are to meet you are to meet yo
u are to meet you are to meet you are to meet you are to meet you are to meet you are to meet yo
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
Ding me on ya break fassyole! Blacko from londn
The Predicted Output is:
Anyone meet you all the meet you all the meet you are to meet you are to meet you are to meet you are to
meet you are to meet you are to meet you are to meet you are to meet you are to meet you are to
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
OH YEAH,AND HAV A GREAT TIME IN NEWQUAY-SEND ME A POSTCARD !1 LOOK AFTER ALL THE GIRLS WHILE IM GONE(U K
NOW THE 1IM TALKIN BOUT!)
The Predicted Output is:
Hell you go to go to meet you all the meet you all the meet you all the meet you all the meet you all th
e meet you all the meet you all going to go to go to go to go to go to go to go to go to go to g
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
Haha... My pleasure lah... Muaks! Enjoy ùrself!
The Predicted Output is:
Helly are are are going to go to meet you are to meet you are to meet you are to meet you are to meet yo
u are to meet you are to meet you are to meet you are to meet you are to meet you are to meet yo
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
Going to reach already
The Predicted Output is:
I am not to meet you all the meet you are to meet you are to meet you are to meet you are to meet you ar
e to meet you are to meet you are to meet you are to meet you are to meet you are to meet you ar
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
Hey think tmr i will take bus down. Dont be late ah.... Cya
The Predicted Output is:
Hey, I don't know what to some already. I am not to meet you are to meet you are to meet you are to meet
you are to meet you are to meet you are to meet you are to meet you are to meet you are to meet
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
Hmmm b7l_jammer that is L ..Hotmail...Hmmm you add me msn ba...
The Predicted Output is:
Haha. I am not to meet you all the meet you all the meet you are to meet you are to meet you are to meet
you are to meet you are to meet you are to meet you are to meet you are to meet you are to meet
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
Wat buses go to ur sch frm amk huh...
The Predicted Output is:
What then you want to go to meet you all the meet you are to meet you are to meet you are to meet you ar
e to meet you are to meet you are to meet you are to meet you are to meet you are to meet you ar
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
Yupz...Hehe u like den gd lor...Hehe =)
The Predicted Output is:
Can meet you all the meet you all the meet you are to meet you are to meet you are to meet you are to me
et you are to meet you are to meet you are to meet you are to meet you are to meet you are to me
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
Ger v wan to go cheong today? zouk .
The Predicted Output is:
No. I am not to meet you all the meet you are to meet you are to meet you are to meet you are to meet yo
u are to meet you are to meet you are to meet you are to meet you are to meet you are to meet yo
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
Haha... Use ur imagination la... Cya tmr...
The Predicted Output is:
Haha. I am not to meet you all the meet you are to meet you are to meet you are to meet you are to meet
you are to meet you are to meet you are to meet you are to meet you are to meet you are to meet
```

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
No nd me to intro someone oso got lotsa admirer liao wat... K la, thurs no changes liao ah...
The Predicted Output is:
No here to meet you all the meet you all the meet you all the meet you all the meet you all the meet you
are to meet you are to meet you are to meet you are to meet you are to meet you are to meet you
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
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:
Ok, I am not to meet you all the be all the meet you all the meet you all the meet you all the meet you
are to meet you are to meet you are to meet you are to meet you are to meet you are to meet you
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
Both me n leona will b late. Me going bedok mrt take train down, she stil in office. Mayb u go shop look
4 ideas first. She wan those working bag.
The Predicted Output is:
I am not to meet you all the meet you all the meet you all the meet you all the meet you all the meet yo
u all the meet you all the meet you all the meet you are to meet you are to meet you are to meet
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
i'll be late...will call u
The Predicted Output is:
Yes. I am not to meet you all the meet you are to meet you are to meet you are to meet you are to meet y
ou are to meet you are to meet you are to meet you are to meet you are to meet you are to meet y
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
Me 25 male...Chinese.Malaysian.
The Predicted Output is:
Then you want to go to meet you are to meet you are to meet you are to meet you are to meet you are to m
eet you are to meet you are to meet you are to meet you are to meet you are to meet you are to m
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
HäPpY ChiLDrEN's DäE!! (. ^_ ^.) dun b shy to admit ür a kid coz i believe derez always a childish side
to every1...enjoy ürself ñ relive those kiddish dayz!
The Predicted Output is:
Hi, what the some already. I am not to meet you all the be all the meet you all the meet you all the mee
t you all the meet you all the meet you all the meet you all going to go to go to go to go to go
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
got how many brand & model?
The Predicted Output is:
I am not to meet you all the meet you are to meet you are to meet you are to meet you are to meet you ar
e to meet you are to meet you are to meet you are to meet you are to meet you are to meet you ar
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
Ok then i settle my own
The Predicted Output is:
Okay, I don't know what to you are going to go to go to go to go to go to go to go to go to go to go to g
o to go to go to go to go to go to go to go to go to go to go to go to go to go to go to go to
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
The Actual Output is:
What are u doing tonight.Go Geylang eat.
The Predicted Output is:
What then you are all be at the meet you all the meet you are to meet you are to meet you are to meet yo
u are to meet you are to meet you are to meet you are to meet you are to meet you are to meet yo
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

In [70]:

```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)
/usr/local/lib/python3.7/dist-packages/nltk/translate/bleu_score.py:490: UserWarning:
Corpus/Sentence contains 0 counts of 3-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)
```

In [71]:

```
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:  0.1318701360495755
```

In [72]:

```
bleu_scores_lst
```

Out[72]:

```
[0,
 0,
 0,
 0,
 0.3742031646082125,
 0,
 0.3760603093086394,
 0.1690308509457033,
 0,
 0.3742031646082125,
 0,
 0.4494780405208269,
 0,
 0,
 0,
 0,
 0.447213595499958,
 0,
 0,
 0.447213595499958]
```

In [ ]: