In [2]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import tensorflow as tf
import keras
import pickle
import fasttext
from keras.callbacks import LearningRateScheduler
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.layers import *
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard, ReduceLROnPlateau
from nlpaug.util.file.download import DownloadUtil

import nlpaug.augmenter.word as naw
from tqdm import tqdm
# pd.options.mode.chained_assignment = None
```

In [3]:

```
data = pd.read_csv("final_data.csv")
data.drop(['Unnamed: 0'],axis=1,inplace=True)
```

In [4]:

```
data.head(5)
```

Out[4]:

	corupted_text	normal_text_input	normal_text_output
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>
4	Hi! How did your week go? Haven heard from you...	<start> Hi! How did your week go? Haven't hear...	Hi! How did your week go? Haven't heard from y...

In [6]:

```
train,test = train_test_split(data, test_size=0.01)
print(train.shape)
train.iloc[0]['normal_text_input']=str(train.iloc[0]['normal_text_input'])+' <end>'
train.iloc[0]['normal_text_output']=str(train.iloc[0]['normal_text_output'])+' <end>'
```

```
(1973, 3)
```

In [7]:

```
%%time
tokenizer_source = Tokenizer(filters='!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n', oov_token='ukn',lower=False)
tokenizer_source.fit_on_texts(train['corupted_text'].values)

tokenizer_target = Tokenizer(filters='!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n', oov_token='ukn',lower=False)
tokenizer_target.fit_on_texts(train['normal_text_input'].values)
```

```
Wall time: 95.4 ms
```

In [8]:

```
%%time
fast_text_model = fasttext.load_model('cc.en.300.bin')
```

```
Wall time: 15.1 s

Warning : `load_model` does not return WordVectorModel or SupervisedModel any more, but a `FastText`
object which is very similar.
```

In [9]:

```
vocab_size_encoder=(len(tokenizer_source.word_index)+1)
vocab_size_decoder = (len(tokenizer_target.word_index)+1)
```

In [10]:

```
embedding_matrix_encoder = np.zeros((vocab_size_encoder,300))
for word, i in tokenizer_source.word_index.items():
    embedding_vector = fast_text_model.get_word_vector(word)
    if embedding_vector is not None:
    # words not found in embedding index will be all-zeros.
       embedding_matrix_encoder[i] = embedding_vector
```

In [11]:

```
embedding_matrix_decoder = np.zeros((vocab_size_decoder,300))
for word, i in tokenizer_target.word_index.items():
    embedding_vector = fast_text_model.get_word_vector(word)
    if embedding_vector is not None:
    # words not found in embedding index will be all-zeros.
       embedding_matrix_decoder[i] = embedding_vector
```

In [12]:

```
print(vocab_size_encoder,"  ",embedding_matrix_encoder.shape)
print(vocab_size_decoder,"  ",embedding_matrix_decoder.shape)
```

```
4615    (4615, 300)
3599    (3599, 300)
```

In [13]:

```
%%time
def convert_word_number(tokenizer,dataframe):
    '''
    heere we convert the each word to a digiti
    '''
    return tokenizer.texts_to_sequences(dataframe)
```

```
Wall time: 0 ns
```

In [14]:

```
%%time
corupted_text_seq_train = convert_word_number(tokenizer_source,train["corupted_text"])
normal_text_seq_input_train = convert_word_number(tokenizer_target,train["normal_text_input"])
normal_text_seq_output_train = convert_word_number(tokenizer_target,train["normal_text_output"])
```

```
Wall time: 101 ms
```

In [15]:

```
%%time
corupted_text_seq_test = convert_word_number(tokenizer_source,test["corupted_text"])
normal_text_seq_input_test = convert_word_number(tokenizer_target,test["normal_text_input"])
normal_text_seq_output_test = convert_word_number(tokenizer_target,test["normal_text_output"])
```

```
Wall time: 997 µs
```

In [16]:

```
'''
finding maximum length of encoder input for padding values
'''
max_len = 0
for i in corupted_text_seq_train:
    if max_len < len(i):
        max_len = len(i)
max_len
```

Out[16]:

```
39
```

In [17]:

```
'''
as we have decoder input seq and decoder ouput seq we need to find the which seq have maximumn length so that we
can pad values
'''

max_len_dec_input = 0
for i in normal_text_seq_input_train:
    if max_len_dec_input < len(i):
        max_len_dec_input = len(i)

max_len_dec_output = 0
for i in normal_text_seq_input_test:
    if max_len_dec_output < len(i):
        max_len_dec_output = len(i)
max_len_dec = 0
if max_len_dec_input < max_len_dec_output:
    max_len_dec = max_len_dec_output
else:
    max_len_dec = max_len_dec_input
max_len_dec
```

Out[17]:

```
44
```

In [18]:

```
def get_pad_sequence(seq,length):
    '''
    here we are doing post padding to every sequence
    '''
    temp = pad_sequences(seq,maxlen=length,padding="post")
    return temp
```

In [19]:

```
%%time
source_seq_input_train = get_pad_sequence(corupted_text_seq_train,max_len)
target_seq_input_train = get_pad_sequence(normal_text_seq_input_train,max_len_dec)
target_seq_ouput_train = get_pad_sequence(normal_text_seq_output_train,max_len_dec)
```

```
Wall time: 48.9 ms
```

In [20]:

```
%%time
source_seq_input_test = get_pad_sequence(corupted_text_seq_test,max_len)
target_seq_input_test = get_pad_sequence(normal_text_seq_input_test,max_len_dec)
target_seq_ouput_test = get_pad_sequence(normal_text_seq_output_test,max_len_dec)
```

```
Wall time: 997 µs
```

In [95]:

```
class Encoder(tf.keras.layers.Layer):
    '''
    takes the input seq and returns the output,hidden and final state
    '''
    def __init__(self, vocab_size, embedding_dim, enc_units,input_length):
        '''
        here we initlaize the necessary attributes
        '''

        super().__init__()
        self.vocab_size = vocab_size
        self.embedding_dim = embedding_dim
        self.input_length = input_length
        self.enc_units= enc_units
        self.lstm_output = 0
        self.lstm_state_h=0
        self.lstm_state_c=0
#         intialize embedding
        self.embedding = Embedding(input_dim=self.vocab_size, output_dim=self.embedding_dim, input_length=self.in
put_length,
                           mask_zero=True, trainable=True, weights=[embedding_matrix_encoder], name="embedding_la
yer_encoder")
        self.lstm = Bidirectional(LSTM(self.enc_units, return_state=True, return_sequences=True, name='Encoder_LS
TM2',dropout=0.2))

    def call(self, input_sentances, training=True):
        '''
        This function takes a sequence input
        Pass the input_sequence input to the Embedding layer, Pass the embedding layer ouput to encoder_lstm
         returns -- All encoder_outputs, last time steps hidden and cell state
        '''
        input_embedd= self.embedding(input_sentances)
        self.lstm_output, lstm_state_h_f, lstm_state_c_f, lstm_state_h_b, lstm_state_c_b = self.lstm(input_embedd
)
        self.lstm_state_h = Concatenate()([lstm_state_h_f, lstm_state_h_b])
        self.lstm_state_c = Concatenate()([lstm_state_c_f, lstm_state_c_b])
        return self.lstm_output, self.lstm_state_h,self.lstm_state_c

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

    def get_states(self):
        return self.lstm_state_h,self.lstm_state_c
```

In [96]:

```
class Attention(tf.keras.layers.Layer):
    '''
    attention layer
    '''

    def __init__(self,scoring_function, att_units):
        super(Attention,self).__init__()
        self.scoring_function = scoring_function
        self.att_units = att_units
        if self.scoring_function=='dot':
            self.dot = tf.keras.layers.Dot(axes=[2,2])
        elif scoring_function == 'general':
            self.WG = Dense(self.att_units)
        elif scoring_function == 'concat':
            self.W1 = Dense(att_units)
            self.W2 = Dense(att_units)
            self.V = Dense(1)

    def call(self,inp):
        decoder_hidden_state,encoder_output=inp
        decoder_hidden_state = tf.expand_dims(decoder_hidden_state, axis=1)
        if self.scoring_function == 'dot':
            score = self.dot([encoder_output, decoder_hidden_state])
        elif self.scoring_function == 'general':
            score = tf.keras.layers.Dot(axes=[2, 2])([self.WG(encoder_output), decoder_hidden_state])
        elif self.scoring_function == 'concat':
            score = self.V(tf.nn.tanh(self.W1(decoder_hidden_state) + self.W2(encoder_output)))
        attention_weights = Softmax(axis=1)(score)
        context_vector = attention_weights * encoder_output
        # shape = (batch_size, dec lstm units)
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights
```

In [97]:

```
class One_Step_Decoder(tf.keras.Model):
    def __init__(self, tar_vocab_size, embedding_dim, input_length, dec_units ,score_fun ,att_units):
        super(One_Step_Decoder,self).__init__()
        self.vocab_size = tar_vocab_size
        self.embedding_dim = embedding_dim
        self.input_length = input_length
        self.dec_units = dec_units
        self.score_fun = score_fun
        self.att_units = att_units
        self.embedding = Embedding(self.vocab_size, self.embedding_dim, trainable=True, weights=[embedding_matrix
_decoder], input_length=self.input_length, mask_zero=True, name="Att_Dec_Embedding")
        self.lstm = LSTM(self.dec_units, return_sequences=True, return_state=True, name="Att_Dec_LSTM",dropout=0.
2)
        self.fc = Dense(self.vocab_size)
        self.attention = Attention(self.score_fun,self.att_units)

    def call(self, inputs2):
         # One step decoder mechanisim step by step:
        # A. Pass the input_to_decoder to the embedding layer and then get the output(batch_size,1,embedding_dim)
        #B. Using the encoder_output and decoder hidden state, compute the context vector.
        # context_vector = tf.expand_dims(context_vector,axis=1)
        # C. Concat the context vector with the step A output
                # D. Pass the Step-C output to LSTM/GRU and get the decoder output and states(hidden and cell sta
te)
#         print("state_h",state_h.shape)
#         print("state_c",state_c.shape)
 # E. Pass the decoder output to dense layer(vocab size) and store the result into output.
    # F. Return the states from step D, output from Step E, attention weights from Step -B
        input_to_decoder, encoder_output, state_h, state_c=inputs2
        embedded_input = self.embedding(input_to_decoder)
        context_vector, attention_weights = self.attention((state_h, encoder_output))
        decoder_input = tf.concat([tf.expand_dims(context_vector, 1), embedded_input], axis=-1)
        decoder_output, dec_state_h, dec_state_c = self.lstm(decoder_input, initial_state=[state_h, state_c])
        decoder_output = tf.reshape(decoder_output, (-1, decoder_output.shape[2]))
        output = self.fc(decoder_output)
        return output, dec_state_h, dec_state_c, attention_weights, context_vector
```

In [98]:

```
class Decoder(tf.keras.Model):
    def __init__(self,out_vocab_size, embedding_dim, input_length, dec_units ,score_fun ,att_units):
        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.score_fun = score_fun
        self.att_units = att_units
        self.onestepdecoder = One_Step_Decoder(self.vocab_size, self.embedding_dim, self.input_length,
                                               self.dec_units, self.score_fun, self.att_units)
    @tf.function
    def call(self, inputs):
        input_to_decoder,encoder_output,decoder_hidden_state,decoder_cell_state=inputs
        all_outputs = tf.TensorArray(tf.float32, size=input_to_decoder.shape[1])
        for timestep in range(input_to_decoder.shape[1]):
            output, decoder_hidden_state, decoder_cell_state, attention_weights, context_vector = self.onestepdec
oder((
                input_to_decoder[:, timestep:timestep+1], encoder_output, decoder_hidden_state, decoder_cell_stat
e))
            all_outputs = all_outputs.write(timestep, output)
        all_outputs = tf.transpose(all_outputs.stack(), [1,0,2])
        return all_outputs
```

In [104]:

```
class Encoder_decoder(Model):
    def __init__(self, encoder_inputs_length,decoder_inputs_length,batch_size,score_fun):
        super().__init__() # https://stackoverflow.com/a/27134600/4084039
        self.batch_size=batch_size
        self.encoder = Encoder(vocab_size= vocab_size_encoder, embedding_dim=300, input_length=encoder_inputs_len
gth, enc_units=128)
        self.decoder = Decoder(out_vocab_size= vocab_size_decoder, embedding_dim=300, input_length=decoder_inputs
_length, dec_units=256, score_fun=score_fun, att_units=256)
    @tf.function
    def call(self, data):
        input,output = data[0], data[1]
        enc_initial_states = self.encoder.initialize_states(self.batch_size)
        encoder_output, encoder_h, encoder_c = self.encoder(input)
        decoder_output   = self.decoder((output, encoder_output, encoder_h, encoder_c))
        return decoder_output
```

In [105]:

```
model = Encoder_decoder(encoder_inputs_length=max_len,decoder_inputs_length=max_len_dec,batch_size=16,score_fun =
"concat")
```

In [106]:

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.98, patience=3, mode="min", verbose=1)
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=5,
    verbose=0,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
)
```

In [107]:

```
def custom_lossfunction(real, pred):
      # Custom loss function that will not consider the loss for padded zeros.
    #https://www.tensorflow.org/tutorials/text/nmt_with_attention#define_the_optimizer_and_the_loss_function
    loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True, reduction='none')
    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_)
optimizer = tf.keras.optimizers.Adam(lr=0.001)
model.compile(optimizer=optimizer,loss=custom_lossfunction)
```

In [108]:

```
%%time
history=model.fit([source_seq_input_train, target_seq_input_train],target_seq_ouput_train, epochs=100,batch_size=
16,validation_split = 0.1,callbacks=[reduce_lr,early_stop])
```

Epoch 1/100

```
Epoch 1/100
111/111 [==============================] - 71s 643ms/step - loss: 2.2244 - val_loss: 2.1952 - lr: 0.
0010
Epoch 2/100
111/111 [==============================] - 60s 542ms/step - loss: 2.0185 - val_loss: 2.1426 - lr: 0.
0010
Epoch 3/100
111/111 [==============================] - 60s 544ms/step - loss: 1.9206 - val_loss: 2.0832 - lr: 0.
0010
Epoch 4/100
111/111 [==============================] - 66s 591ms/step - loss: 1.8132 - val_loss: 2.0181 - lr: 0.
0010
Epoch 5/100
111/111 [==============================] - 74s 664ms/step - loss: 1.7100 - val_loss: 1.9669 - lr: 0.
0010
Epoch 6/100
111/111 [==============================] - 69s 619ms/step - loss: 1.6134 - val_loss: 1.9336 - lr: 0.
0010
Epoch 7/100
111/111 [==============================] - 64s 572ms/step - loss: 1.5197 - val_loss: 1.8930 - lr: 0.
0010
Epoch 8/100
111/111 [==============================] - 65s 587ms/step - loss: 1.4269 - val_loss: 1.8676 - lr: 0.
0010
Epoch 9/100
111/111 [==============================] - 60s 543ms/step - loss: 1.3393 - val_loss: 1.8302 - lr: 0.
0010
Epoch 10/100
111/111 [==============================] - 61s 551ms/step - loss: 1.2498 - val_loss: 1.8208 - lr: 0.
0010
Epoch 11/100
111/111 [==============================] - 62s 559ms/step - loss: 1.1645 - val_loss: 1.7910 - lr: 0.
0010
Epoch 12/100
111/111 [==============================] - 61s 549ms/step - loss: 1.0808 - val_loss: 1.7750 - lr: 0.
0010
Epoch 13/100
111/111 [==============================] - 60s 543ms/step - loss: 0.9964 - val_loss: 1.7838 - lr: 0.
0010
Epoch 14/100
111/111 [==============================] - 61s 551ms/step - loss: 0.9211 - val_loss: 1.7351 - lr: 0.
0010
Epoch 15/100
111/111 [==============================] - 61s 553ms/step - loss: 0.8390 - val_loss: 1.7331 - lr: 0.
0010
Epoch 16/100
111/111 [==============================] - 61s 547ms/step - loss: 0.7726 - val_loss: 1.7114 - lr: 0.
0010
Epoch 17/100
111/111 [==============================] - 61s 547ms/step - loss: 0.7090 - val_loss: 1.7162 - lr: 0.
0010
Epoch 18/100
111/111 [==============================] - 61s 548ms/step - loss: 0.6439 - val_loss: 1.6999 - lr: 0.
0010
Epoch 19/100
111/111 [==============================] - 61s 549ms/step - loss: 0.5972 - val_loss: 1.7067 - lr: 0.
0010
Epoch 20/100
111/111 [==============================] - 61s 548ms/step - loss: 0.5267 - val_loss: 1.6978 - lr: 0.
0010
Epoch 21/100
111/111 [==============================] - 60s 543ms/step - loss: 0.4620 - val_loss: 1.6898 - lr: 0.
0010
Epoch 22/100
111/111 [==============================] - 60s 538ms/step - loss: 0.4180 - val_loss: 1.6848 - lr: 0.
0010
Epoch 23/100
111/111 [==============================] - 60s 537ms/step - loss: 0.3797 - val_loss: 1.6868 - lr: 0.
0010
Epoch 24/100
111/111 [==============================] - 60s 537ms/step - loss: 0.3405 - val_loss: 1.7014 - lr: 0.
0010
Epoch 25/100
111/111 [==============================] - ETA: 0s - loss: 0.3058
Epoch 00025: ReduceLROnPlateau reducing learning rate to 0.0009809000004654750 24.
111/111 [==============================] - 61s 546ms/step - loss: 0.3058 - val_loss: 1.7034 - lr: 0.
0010
Epoch 26/100
111/111 [==============================] - 60s 545ms/step - loss: 0.2700 - val_loss: 1.7089 - lr: 9.
8000e-04
Epoch 27/100
111/111 [==============================] - 61s 546ms/step - loss: 0.2526 - val_loss: 1.7423 - lr: 9.
8000e-04
Wall time: 30min 16s
```

In [109]:

```
def predict(input_sentence):

  input_sequence=tokenizer_source.texts_to_sequences([input_sentence])
  inputs=pad_sequences(input_sequence, maxlen=max_len, padding='post')
  inputs=tf.convert_to_tensor(inputs)
  result=''
  units=128
  hidden=[tf.zeros((1,units))]
  encoder_output,hidden_state,cell_state=model.encoder(inputs)
  dec_hidden=hidden_state
  dec_input=tf.expand_dims([tokenizer_target.word_index['<start>']],0)
  for t in range(40):
      predictions,dec_hidden,cell_state,attention_weights,context_vector=model.decoder.onestepdecoder((dec_input,
encoder_output,dec_hidden,cell_state))

      predicted_id=tf.argmax(predictions[0]).numpy()
      result+=tokenizer_target.index_word[predicted_id]+' '
      if tokenizer_target.word_index['<end>']==predicted_id:
          return result
      dec_input= tf.expand_dims([predicted_id],0)
  return result
```

In [110]:

```
for i in range(0,10):
    input_sentence=test["corupted_text"].iloc[i]
    print('Input:',input_sentence)
    print('Prediction:',predict(input_sentence))
    print('Actual:',test["normal_text_output"].iloc[i])
    print('*'*100)
```

```
Input: Rain,can u call mi nw?97482959.
Prediction: Rain you call you now <end>
Actual: Rain, can you call me now? 97482959. <end>
****************************************************************************************************
Input: Roy intro pls
Prediction: Mimi40 please introduce please <end>
Actual: Roy introduce please. <end>
****************************************************************************************************
Input: Dear.... Miss you.
Prediction: Hey how you <end>
Actual: Dear. Miss you. <end>
****************************************************************************************************
Input: What're you doing tonight?
Prediction: Jess you are here <end>
Actual: What're you doing tonight? <end>
****************************************************************************************************
Input: hi roy ! Intro pls ..... Pls sms at 016 5419814
Prediction: Hi Roy care please introduce Please chat anyone <end>
Actual: Hi Roy! Introduce please. Please SMS at 016 5419814. <end>
****************************************************************************************************
Input: U having ur lunch at home or in school? I'm having lunch now.
Prediction: You are you free at home or in school or not at home now <end>
Actual: You are having your lunch at home or in school? I'm having lunch now. <end>
****************************************************************************************************
Input: hai sine? intro pls?
Prediction: Hi JOY introduce please <end>
Actual: Hi Sine? Introduce please? <end>
****************************************************************************************************
Input: Okie...Sian rite? Hafta go back to e hw days...Wat u taking? N how's yr timetable? Anyway u t
ink i shld bring it to c doc? My sis say will hef disease 1...
Prediction: Ok since right Have you back to the way Because what you are born And when your other wa
y else do you want to see twice My sister say will scold me asap <end>
Actual: Ok. Sian right? I have to go back to the whole days. What are you taking? And how's your tim
etable? Anyway you think I should bring it to see doctor? My sister says it will have disease. <end>
****************************************************************************************************
Input: Hi jeff! Can intro pls? Phone no if any? My no is 0166305681 ok?
Prediction: Hi Everybody can I come please introduce number if if no point an number ok <end>
Actual: Hi jeff! Can introduce please? Phone number if any? My no is 0166305681 ok? <end>
****************************************************************************************************
Input: You have to bring your's newspaper 12 behind.
Prediction: You have to bring 3 months for quarantine <end>
Actual: You have to bring your newspaper to behind. <end>
****************************************************************************************************
```