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 [4]:

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

In [5]:

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

Out[5]:

	corupted_text	normal_text
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 [6]:

```
normal_text = list(data["normal_text"].values)
aug_text = []
aug = naw.SynonymAug()
for sentence in tqdm(normal_text):
    aug_text.append(aug.augment(sentence))
df_synaug = pd.DataFrame({"corupted_text":aug_text,"normal_text":aug_text})
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 1993/1993 [00:12
<00:00, 160.57it/s]
```

In [11]:

```
normal_text = list(data["normal_text"].values)
aug_text = []
aug = naw.RandomWordAug()
for sentence in tqdm(normal_text):
    aug_text.append(aug.augment(sentence))
df_random = pd.DataFrame({"corupted_text":aug_text,"normal_text":aug_text})
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 1993/1993 [00:00<
00:00, 3138.82it/s]
```

In [13]:

```
df = pd.concat([data,df_synaug,df_random])
```

In [14]:

```
df.shape
```

Out[14]:

```
(5979, 2)
```

In [15]:

```
df["normal_text_input"] = "<start>"+df["normal_text"].astype(str)
df["normal_text_output"] = df["normal_text"].astype(str) + "<end>"
```

In [16]:

```
df.drop(["normal_text"],axis=1,inplace=True)
```

In [17]:

```
df.shape
```

Out[17]:

```
(5979, 3)
```

In [18]:

```
train,test = train_test_split(df, 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>'
```

```
(5919, 3)
```

In [19]:

```
%%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: 598 ms
```

In [20]:

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

```
Wall time: 19.5 s

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

In [21]:

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

In [22]:

```
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 [23]:

```
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 [24]:

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

```
7259    (7259, 300)
5866    (5866, 300)
```

In [25]:

```
%%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 [26]:

```
%%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: 637 ms

In [27]:

```
%%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: 4.78 ms

In [28]:

```
'''
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[28]:

54

In [29]:

```
'''
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[29]:

54

In [30]:

```
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 [31]:

```
%%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: 242 ms
```

In [32]:

```
%%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: 4.3 ms
```

In [47]:

```
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 [48]:

```
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 [49]:

```
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 [50]:

```
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 [51]:

```
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 [52]:

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

In [53]:

```
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 [54]:

```
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 [55]:

```
%%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
333/333 [==============================] - 371s 1s/step - loss: 1.4737 - val_loss: 1.1096 - lr: 0.00
10
Epoch 2/100
333/333 [==============================] - 279s 839ms/step - loss: 0.9117 - val_loss: 0.7554 - lr: 0
.0010
Epoch 3/100
333/333 [==============================] - 283s 850ms/step - loss: 0.6171 - val_loss: 0.5623 - lr: 0
.0010
Epoch 4/100
333/333 [==============================] - 281s 843ms/step - loss: 0.4425 - val_loss: 0.4334 - lr: 0
.0010
Epoch 5/100
333/333 [==============================] - 281s 843ms/step - loss: 0.3215 - val_loss: 0.3584 - lr: 0
.0010
Epoch 6/100
333/333 [==============================] - 283s 849ms/step - loss: 0.2453 - val_loss: 0.3086 - lr: 0
.0010
Epoch 7/100
333/333 [==============================] - 310s 931ms/step - loss: 0.1781 - val_loss: 0.2640 - lr: 0
.0010
Epoch 8/100
333/333 [==============================] - 305s 917ms/step - loss: 0.1292 - val_loss: 0.2334 - lr: 0
.0010
Epoch 9/100
333/333 [==============================] - 306s 918ms/step - loss: 0.0945 - val_loss: 0.2088 - lr: 0
.0010
Epoch 10/100
333/333 [==============================] - 304s 912ms/step - loss: 0.0709 - val_loss: 0.1916 - lr: 0
.0010
Epoch 11/100
333/333 [==============================] - 295s 885ms/step - loss: 0.0576 - val_loss: 0.1827 - lr: 0
.0010
Epoch 12/100
333/333 [==============================] - 304s 912ms/step - loss: 0.0447 - val_loss: 0.1749 - lr: 0
.0010
Epoch 13/100
333/333 [==============================] - 300s 902ms/step - loss: 0.0365 - val_loss: 0.1697 - lr: 0
.0010
Epoch 14/100
333/333 [==============================] - 298s 894ms/step - loss: 0.0306 - val_loss: 0.1688 - lr: 0
.0010
Epoch 15/100
333/333 [==============================] - 299s 898ms/step - loss: 0.0267 - val_loss: 0.1657 - lr: 0
.0010
Epoch 16/100
333/333 [==============================] - 309s 927ms/step - loss: 0.0236 - val_loss: 0.1639 - lr: 0
.0010
Epoch 17/100
333/333 [==============================] - 294s 884ms/step - loss: 0.0232 - val_loss: 0.1624 - lr: 0
.0010
Epoch 18/100
333/333 [==============================] - 311s 935ms/step - loss: 0.0191 - val_loss: 0.1627 - lr: 0
.0010
Epoch 19/100
333/333 [==============================] - 279s 837ms/step - loss: 0.0173 - val_loss: 0.1641 - lr: 0
.0010
Epoch 20/100
333/333 [==============================] - ETA: 0s - loss: 0.0152
Epoch 00020: ReduceLROnPlateau reducing learning rate to 0.0009800000465475024.
333/333 [==============================] - 279s 836ms/step - loss: 0.0152 - val_loss: 0.1630 - lr: 0
.0010
Epoch 21/100
333/333 [==============================] - 279s 837ms/step - loss: 0.0136 - val_loss: 0.1634 - lr: 9
.8000e-04
Epoch 22/100
333/333 [==============================] - 278s 836ms/step - loss: 0.0123 - val_loss: 0.1633 - lr: 9
.8000e-04
Wall time: 1h 53min 37s
```

In [56]:

```
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 [57]:

```
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: Toysarus? The place that sell balloon? Merely no point. Queensway? Ar you at Penisula?
Prediction: The committees promoting in the mambo stuff is cheap Torquay is at 2 <end>
Actual: Toysarus? The place that sell balloon? Merely no point. Queensway? Ar you at Penisula?<end>
****
Input: 1: 15pm. Reached about.
Prediction: The place balloons But <end>
Actual: 1: 15pm. Reached about.<end>
****
Input: Really only today? Topshop and miss self ridgeline likewise got store wide deduction. ..
Prediction: Yes Both of ' s later later later later later later <end>
Actual: Really only today? Topshop and miss self ridgeline likewise got store wide deduction. ..<end
>
****
Input: Do you want to come to my? But I got to after school, while only. ' not fall at this time, '
m enough.
Prediction: How do you ' t buy sandals How ' t buy sandals How ' t buy sandals How ' m buy sandals H
ow ' <end>
Actual: Do you want to come to my? But I got to after school, while only. ' not fall at this time, '
m enough.<end>
****
Input: We are outside.
Prediction: Where are you <end>
Actual: We are outside.<end>
****
Input: You got add me or not? Ane can insure you.
Prediction: You have have you have to the wrong you been <end>
Actual: You got add me or not? Ane can insure you.<end>
****
Input: But I ' t look fringe! . well, ' s cut. So you cut, we and dye our hair together? Good luck y
our papers!
Prediction: But Tuesday I ' s normal your timetable Anyway ' s normal will the same Audrey them with
with with our and <end>
Actual: But I ' t look fringe! . well, ' s cut. So you cut, we and dye our hair together? Good luck
your papers!<end>
****
Input: Buckeye state no, that means you aren ' t hail for statistic? Then you can ' t help pine tree
state print, because I need it right after that. Okay, Entirely the best for you test! Don ' t worry
.
Prediction: Grounded but I ' m buy sandals How ' t buy always send the wrong then ' s normal normal
normal normal normal normal normal ' t bother tester And how ' s normal will to out about ' s
Actual: Buckeye state no, that means you aren ' t hail for statistic? Then you can ' t help pine tre
e state print, because I need it right after that. Okay, Entirely the best for you test! Don ' t wor
ry.<end>
****
Input: Don act stupid !
Prediction: wasn ' s <end>
Actual: Don't act stupid!<end>
****
Input: What are you eating?
Prediction: are are you you <end>
Actual: What are you eating?<end>
****