Sentence Correction using Recurrent Neural Network

1.Social media text messages contains short form of english text and it leads to machine learning models to predict the wrong english sentences.

2.This case study helps in converting the social media text message into proper english text messages

SOURCE Data Description

1. Source Data Link http://www.comp.nus.edu.sg/~nlp/sw/sm_norm_mt.tar.gz
2. SourceFile Size:413KB
3. Description of the SourceData.This source file contains three types of data

```
1.Social Media Text messages

2.Chinese messages

3.Original English Text
```

4. Number of rows present in source data: 2000

Machine Learning Probelm

1. In this case study i am implementing research paper on sentence correction using RNN's Link:https://cs224d.stanford.edu/reports/Lewis.pdf
2. Using Encoder and Decoder Sequence to Sequenc network with LSTM which are types of RNN's
3. Loss function : categorical_crossentropy

```
%tensorflow_version 2.x
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
    raise SystemError('GPU device not found')
    raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
     Found GPU at: /device:GPU:0
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import pandas as pd
import re
import tensorflow as tf
from tensorflow.keras.layers import Embedding, LSTM, Dense,Input
from tensorflow.keras.models import Model,load_model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
```

Extracting the Data

```
file = open("/content/drive/MyDrive/CaseStudy2/en2cn-2k.en2nen2cn",'r',encoding = 'utf-8')
```

```
File_Lines=[]
with open("/content/drive/MyDrive/CaseStudy2/en2cn-2k.en2nen2cn") as fp:
    Lines = fp.readlines()
    File_Lines=Lines
```

```
# In this piece of code,looping through the source file and extracting the normalized text an
Original_English_Text=[]
Normalized_Text=[]
Chinese_Text=[]
Normalized_count=0
English_count=1
Chinese_count=2
for i in range(len(File_Lines)):
    if (Normalized_count==len(File_Lines)):
        break
    else:
        Normalized_Text.append(File_Lines[Normalized_count])
        Original_English_Text.append(File_Lines[English_count])
        Chinese_Text.append(File_Lines[Chinese_count])
        Normalized_count=Normalized_count+3
        English_count=English_count+3
        Chinese_count=Chinese_count+3
        Chinese_count=Chinese_count+3
```

```
print(len(Original_English_Text))
print(len(Normalized_Text))
print(len(Chinese_Text))
```

```
2000
2000
2000
```

```
print(Original_English_Text[5])
print(Normalized_Text[5])
```

```
Haha. Okay. Are you going to mail her? Or do you want me to reply?

Haha... Okay... You going to mail her? Or you want me to reply...
```

```
Data={"NormalizedText":Normalized_Text,"Original_English_Text":Original_English_Text}
```

```
import pandas as pd
SourceData=pd.DataFrame(Data)
```

```
print(SourceData.head(5))
```

```
                                      NormalizedText                              Origin
0                 U wan me to "chop" seat 4 u nt?\n   Do you want me to reserve seat fo
1  Yup. U reaching. We order some durian pastry a...  Yeap. You reaching? We ordered som
2  They become more ex oredi... Mine is like 25.....  They become more expensive already
3                          I'm thai. what do u do?\n                        I'm Thai. Wh
4  Hi! How did your week go? Haven heard from you...  Hi! How did your week go? Haven't
```

```
pd.to_pickle(SourceData, "/content/drive/MyDrive/CaseStudy2/SourceData.pkl")
```

Importing Source Data

```
# Reading the Source File

import pandas as pd
Data=pd.read_pickle(r"/content/drive/MyDrive/CaseStudy2/SourceData.pkl")
```

```
Data.head(5)
```

	NormalizedText	Original_English_Text
0	U wan me to "chop" seat 4 u nt?\n	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...
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?\n	I'm Thai. What do you do?\n

```
#Maximum length in words
Max_Length_Normalized_Text = Data['NormalizedText'].str.split().str.len().max()
print("The maximum length in words for NormalizedText before preprocessing: " +  str(Max_Leng

Max_Length_Original_English_Text = Data['Original_English_Text'].str.split().str.len().max()
print("The maximum length in words for Original English Text before preprocessing: " +  str(M
```

```
    The maximum length in words for NormalizedText before preprocessing: 49
    The maximum length in words for Original English Text before preprocessing: 59
```

```
#Minimum length in words

Min_Length_Normalized_Text = Data['NormalizedText'].str.split().str.len().min()
print("The minimum length in words for NormalizedText before preprocessing: " +  str(Min_Leng

Min_Length_Original_English_Text = Data['Original_English_Text'].str.split().str.len().min()
print("The minimum length in words for Original English Text before preprocessing: " +  str(M
```

```
    The minimum length in words for NormalizedText before preprocessing: 1
    The minimum length in words for Original English Text before preprocessing: 1
```

Data Augmentation using nlpaug Library

```
pip install nlpaug
```

```
    Collecting nlpaug
      Downloading https://files.pythonhosted.org/packages/eb/f8/b11caecdd19aa2b1b2cb46c6cbb
        |████████████████████████████████| 399kB 8.2MB/s
    Installing collected packages: nlpaug
    Successfully installed nlpaug-1.1.3
```

```
from nlpaug.util.file.download import DownloadUtil
DownloadUtil.download_fasttext(model_name='wiki-news-300d-1M', dest_dir='.')
```

```
Original_English_Text_Aug=list(Data.Original_English_Text.values)
import nlpaug.augmenter.char as nac
import nlpaug.augmenter.word as naw
```

```
Normalized_Text_Aug1=[]
Normalized_Text_Aug2=[]
#aug = naw.SpellingAug()
aug = naw.WordEmbsAug(
    model_type='fasttext', model_path='/content/wiki-news-300d-1M.vec',
    action="insert")
aug_syn = naw.SynonymAug(aug_src='wordnet')
#aug_ocr = nac.OcrAug()
for text in Original_English_Text_Aug:
  augmented_text = aug.augment(text)
  Normalized_Text_Aug1.append(augmented_text)
  augmented_syn = aug_syn.augment(text)
  Normalized_Text_Aug2.append(augmented_syn)
```

```
FastText_Augmented_Data=pd.DataFrame(list(zip(Normalized_Text_Aug1,Original_English_Text_Aug)
Syn_Augmented_Data=pd.DataFrame(list(zip(Normalized_Text_Aug2,Original_English_Text_Aug)),col
```

```
from sklearn.utils import shuffle
DataFramesList=[Data,FastText_Augmented_Data,Syn_Augmented_Data]
Data=pd.concat(DataFramesList)
Data = shuffle(Data)
pd.to_pickle(Data, "/content/drive/MyDrive/CaseStudy2/AugmentedData.pkl")
print(len(Data))
```

```
6000
```

```
Data=Data=pd.read_pickle(r"/content/drive/MyDrive/CaseStudy2/AugmentedData.pkl")
```

Preprocessing the Data

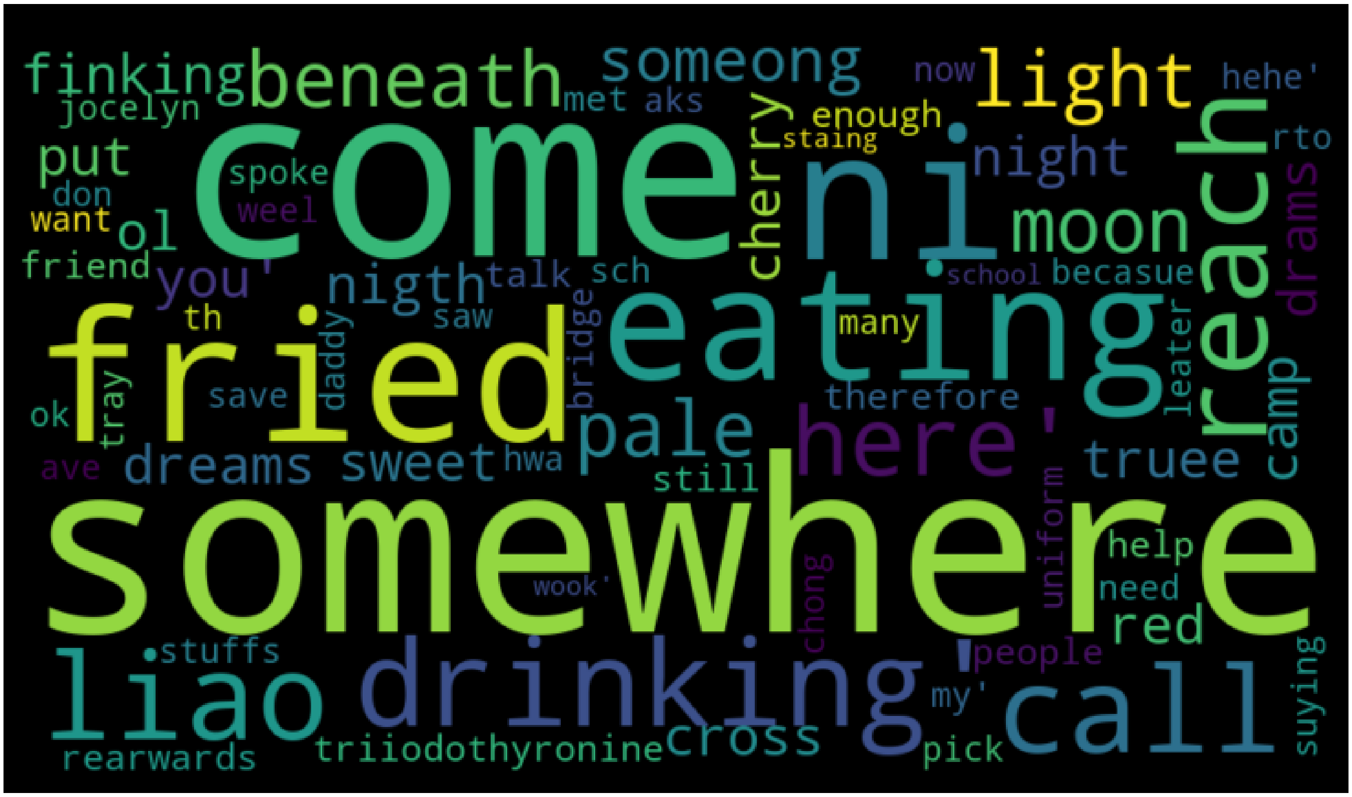
```
import re
def decontractions(phrase):
    """decontracted takes text and convert contractions into natural form.
     ref: https://stackoverflow.com/questions/19790188/expanding-english-language-contraction
    # specific
    phrase = re.sub(r"won\'t", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    phrase = re.sub(r"won\’t", "will not", phrase)
    phrase = re.sub(r"can\’t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)

    phrase = re.sub(r"n\’t", " not", phrase)
    phrase = re.sub(r"\’re", " are", phrase)
    phrase = re.sub(r"\’s", " is", phrase)
    phrase = re.sub(r"\’d", " would", phrase)
    phrase = re.sub(r"\’ll", " will", phrase)
    phrase = re.sub(r"\’t", " not", phrase)
    phrase = re.sub(r"\’ve", " have", phrase)
    phrase = re.sub(r"\’m", " am", phrase)

    return phrase

def preprocess(text):
    text = text.lower()
    text = decontractions(text)
    text = re.sub('[$)\?"’.°!;\'€%:,(/]', '', text)
    text = re.sub('[^A-Za-z0-9 ]+', '', text)
    text = re.sub('[0-9]','',text)
    return text
```

```
Data['Original_English_Text'] = Data['Original_English_Text'].apply(preprocess)
Data['NormalizedText'] = Data['NormalizedText'].apply(preprocess)
```

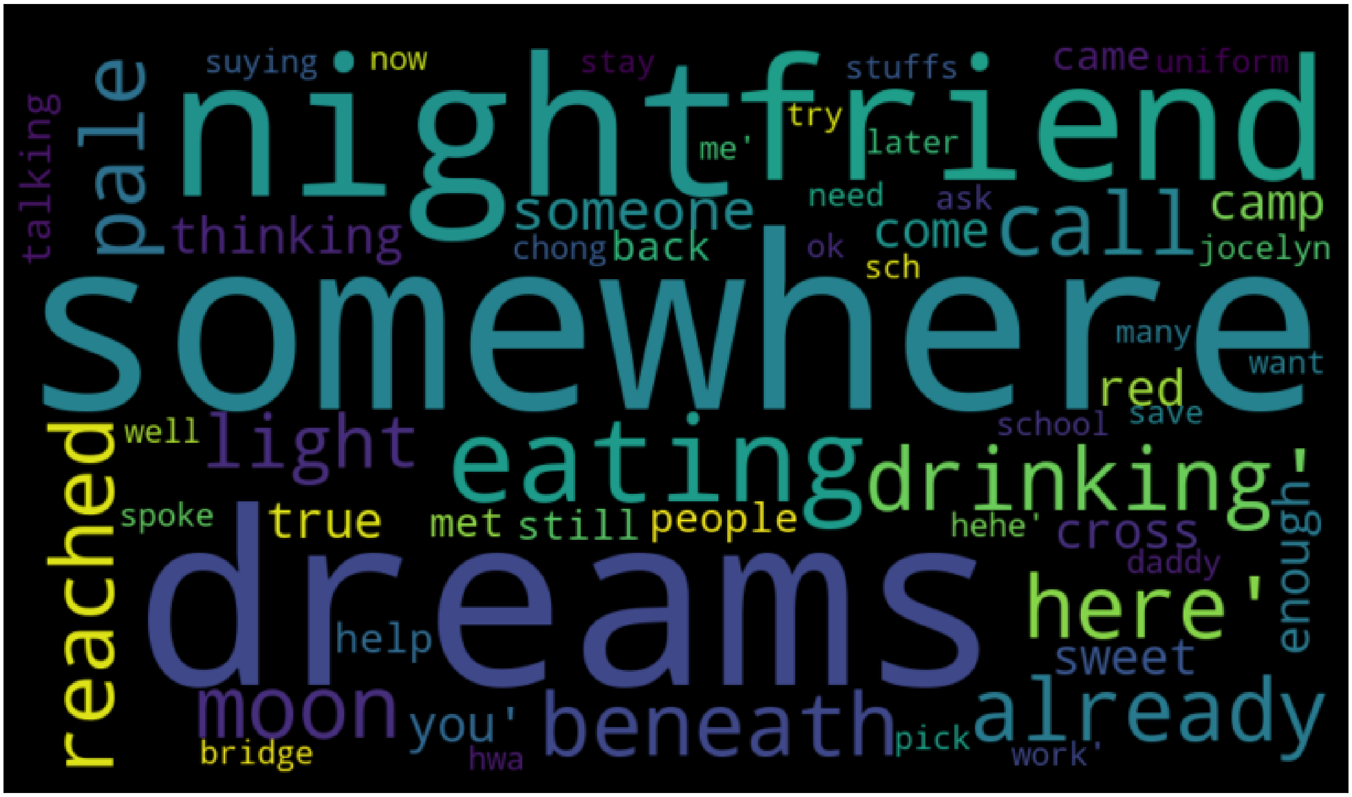
```
Data.head(5)
```

	NormalizedText	Original_English_Text
1923	hehe so how are you spending your sunday	hehe so how are you spending your sunday
410	pic ok kre i m treadle at city link already	ok i am at city link already
1452	joey be you from india	joey are you from india
691	watch jz married leihh	watch just married haha
464	oh tomorrow ve got driving object lesson can...	oh tomorrow i have got driving lesson can not ...

```
#Word cloud on NormalizedTextMessages
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
text = Data.NormalizedText.values
wordcloud = WordCloud(
    width = 900,
    height = 500,
    background_color = 'black',
    stopwords = STOPWORDS).generate(str(text))
fig = plt.figure(
    figsize = (10, 10),
    facecolor = 'k',
    edgecolor = 'k')
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.title("Normalized Text WordCloud")
plt.show()
```



```
#WordCloud on Original English Text
text = Data.Original_English_Text.values
wordcloud = WordCloud(
    width = 900,
    height = 500,
    background_color = 'black',
    stopwords = STOPWORDS).generate(str(text))
fig = plt.figure(
    figsize = (10, 10),
    facecolor = 'k',
    edgecolor = 'k')
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.title("Englisg Text WordCloud")
plt.show()
```



```
Data['Original_English_Text_inp'] = '<start> ' + Data['Original_English_Text'].astype(str)
Data['Original_English_Text_out'] = Data['Original_English_Text'].astype(str) + ' <end>'

Data.head()
```

	NormalizedText	Original_English_Text	Original_English_Text_inp	Original_Englis
1923	hehe so how are you spending your sunday	hehe so how are you spending your sunday	<start> hehe so how are you spending your sunday	hehe so how are y your su
410	pic ok kre i m treadle at city link already	ok i am at city link already	<start> ok i am at city link already	ok i am at city
1452	joey be you from india	joey are you from india	<start> joey are you from india	joey are you from
	watch iz married		<start> watch iust married	

```
Data = Data.drop(['Original_English_Text'], axis=1)
```

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(Data, test_size=0.2)
train,validation=train_test_split(train, test_size=0.075)
print(len(train))
print(len(test))
print(len(validation))
```

```
4440
1200
```

```
360
```

```
tknizer_english = Tokenizer(filters='!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n',char_level=False)
tknizer_english.fit_on_texts(train['Original_English_Text_inp'].values)
tknizer_normal_text = Tokenizer(filters='!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n',char_level=False
tknizer_normal_text.fit_on_texts(train['NormalizedText'].values)
```

```
encoder_vcab=tknizer_normal_text.word_index
decoder_vcab=tknizer_english.word_index
```

```
vocab_size_eng=len(tknizer_english.word_index.keys())
print(vocab_size_eng)
vocab_size_normal_text=len(tknizer_normal_text.word_index.keys())
print(vocab_size_normal_text)
```

```
2837
11971
```

```
embeddings_index = dict()
Decoder_embedding_index=dict()
f = open('/content/wiki-news-300d-1M.vec')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

Encoder_embedding_matrix = np.zeros((len(encoder_vcab)+1, 300))
for word, i in encoder_vcab.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        Encoder_embedding_matrix[i] = embedding_vector

Decoder_embedding_matrix = np.zeros((len(decoder_vcab)+1, 300))
for word, i in decoder_vcab.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        Decoder_embedding_matrix[i] = embedding_vector
```

```
print(Encoder_embedding_matrix.shape)
print(Decoder_embedding_matrix.shape)
```

```
(11972, 300)
(2838, 300)
```

```
class Dataset:
    def __init__(self, Data, tknizer_english, tknizer_normal_text, max_len):
        self.encoder_inps = Data['NormalizedText'].values
        self.decoder_inps = Data['Original_English_Text_inp'].values
        self.decoder_outs = Data['Original_English_Text_out'].values
        self.tknizer_eng = tknizer_english
        self.tknizer_nor= tknizer_normal_text
        self.max_len = max_len

    def __getitem__(self, i):
        self.encoder_seq = self.tknizer_nor.texts_to_sequences([self.encoder_inps[i]]) # need
        self.decoder_inp_seq = self.tknizer_eng.texts_to_sequences([self.decoder_inps[i]])
        self.decoder_out_seq = self.tknizer_eng.texts_to_sequences([self.decoder_outs[i]])


        self.encoder_seq = pad_sequences(self.encoder_seq, maxlen=self.max_len, dtype='int32'
        self.decoder_inp_seq = pad_sequences(self.decoder_inp_seq, maxlen=self.max_len, dtype
        self.decoder_out_seq = pad_sequences(self.decoder_out_seq, maxlen=self.max_len, dtype
        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 c
        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)


train_dataset = Dataset(train, tknizer_english, tknizer_normal_text, 50)
test_dataset  = Dataset(test, tknizer_english, tknizer_normal_text, 50)
Validation_dataset= Dataset(validation, tknizer_english, tknizer_normal_text, 50)
```

```
train_dataloader = Dataloder(train_dataset, batch_size=50)
test_dataloader = Dataloder(test_dataset, batch_size=50)
validation_dataloader=Dataloder(Validation_dataset, batch_size=50)


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

```
     (50, 50) (50, 50) (50, 50)
     (50, 50)
```

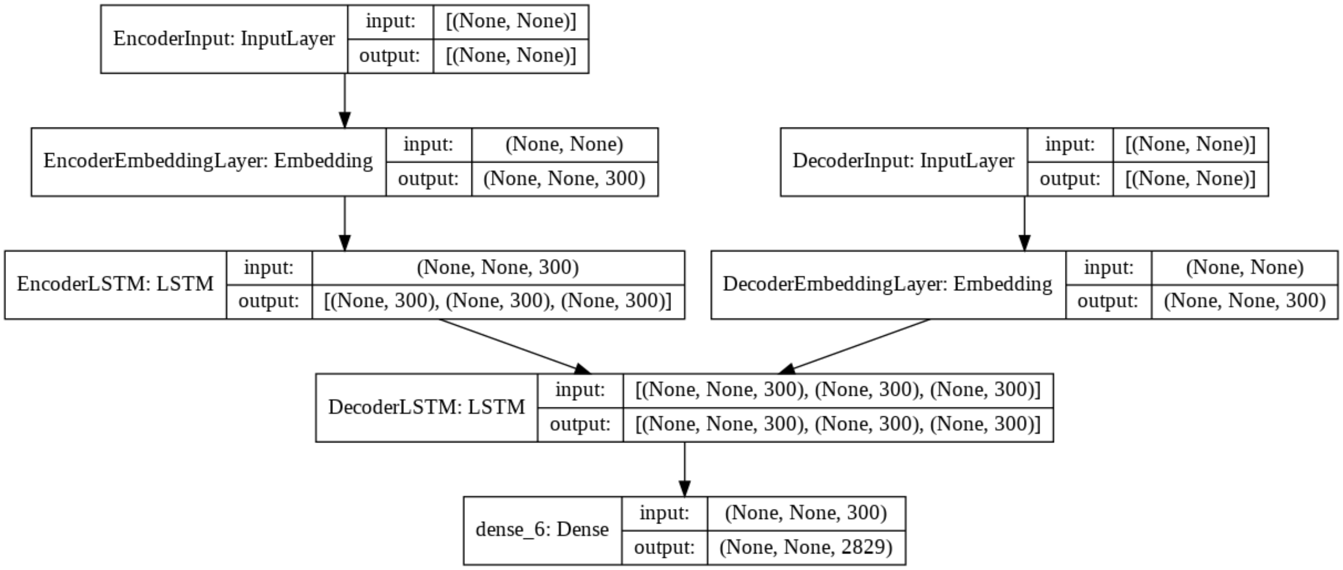
Encoder and Decoder Neural Network

```
# Encoder with LSTM
encoder_inputs = Input(shape=(None,),name="EncoderInput")
enc_emb =  Embedding(vocab_size_normal_text+1,300, mask_zero = True,name="EncoderEmbeddingLay
encoder_lstm = LSTM(300,activation='tanh',return_state=True,name="EncoderLSTM")

encoder_outputs, state_h, state_c = encoder_lstm(enc_emb)
encoder_states = [state_h, state_c]

# Setting up the decoder

decoder_inputs = Input(shape=(None,),name="DecoderInput")
dec_emb_layer = Embedding(vocab_size_eng+1,300,mask_zero = True,name="DecoderEmbeddingLayer",
dec_emb = dec_emb_layer(decoder_inputs)
decoder_lstm = LSTM(300,activation='tanh',return_sequences=True, return_state=True,name="Deco
decoder_outputs, _, _ = decoder_lstm(dec_emb,
                                     initial_state=encoder_states)

decoder_dense = Dense(vocab_size_eng, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)


model.summary()
```

```
     Model: "model_5"
     ________________________________________________________________________________________
     Layer (type)                    Output Shape         Param #     Connected to
     ========================================================================================
     EncoderInput (InputLayer)       [(None, None)]       0
     ________________________________________________________________________________________
     DecoderInput (InputLayer)       [(None, None)]       0
     ________________________________________________________________________________________
     EncoderEmbeddingLayer (Embeddin (None, None, 300)    3543900     EncoderInput[0][0]
     ________________________________________________________________________________________
     DecoderEmbeddingLayer (Embeddin (None, None, 300)    849000      DecoderInput[0][0]
     ________________________________________________________________________________________
     EncoderLSTM (LSTM)              [(None, 300), (None, 721200      EncoderEmbeddingLayer[
     ________________________________________________________________________________________
```

```
DecoderLSTM (LSTM)            [(None, None, 300),  721200      DecoderEmbeddingLayer[
                                                               EncoderLSTM[0][1]
                                                               EncoderLSTM[0][2]
____________________________________________________________________________________
dense_6 (Dense)               (None, None, 2829)   851529      DecoderLSTM[0][0]
====================================================================================
Total params: 6,686,829
Trainable params: 6,686,829
Non-trainable params: 0
____________________________________________________________________________________
```

```
from keras.utils import plot_model
plot_model(model, to_file='modelsummary.png', show_shapes=True, show_layer_names=True)
```


EncoderInput: InputLayer
input: [(None, None)]
output: [(None, None)]
EncoderEmbeddingLayer: Embedding
input: (None, None)
output: (None, None, 300)
DecoderInput: InputLayer
input: [(None, None)]
output: [(None, None)]
EncoderLSTM: LSTM
input: (None, None, 300)
output: [(None, 300), (None, 300), (None, 300)]
DecoderEmbeddingLayer: Embedding
input: (None, None)
output: (None, None, 300)
DecoderLSTM: LSTM
input: [(None, None, 300), (None, 300), (None, 300)]
output: [(None, None, 300), (None, 300), (None, 300)]
dense_6: Dense
input: (None, None, 300)
output: (None, None, 2829)

Defining the callbacks

```
import datetime
from tensorflow.keras.callbacks import ModelCheckpoint,LearningRateScheduler,EarlyStopping,Te
earlystop = EarlyStopping(monitor='val_loss', min_delta=0.001, patience=5, verbose=1)
filepath="/content/drive/MyDrive/CaseStudy2/Model-1/weights-{epoch:02d}-{val_acc:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_acc',  verbose=4, save_best_only
logdir = "/content/drive/MyDrive/CaseStudy2/Model-1/Logs/fit_model2/" + datetime.datetime.now
```

```
train_summary_writer = tf.summary.create_file_writer(logdir)
tensorboard_callback = TensorBoard(log_dir=logdir,histogram_freq=1,profile_batch = 100000000)
callback_list = [checkpoint,tensorboard_callback]


#Compiling the Model with Adam as optimizer and categorical cross entropy as loss function
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['acc'])


train_steps=train.shape[0]//100
valid_steps=validation.shape[0]//100

Model_Output=model.fit_generator(train_dataloader,steps_per_epoch=train_steps,epochs=50, vali
```

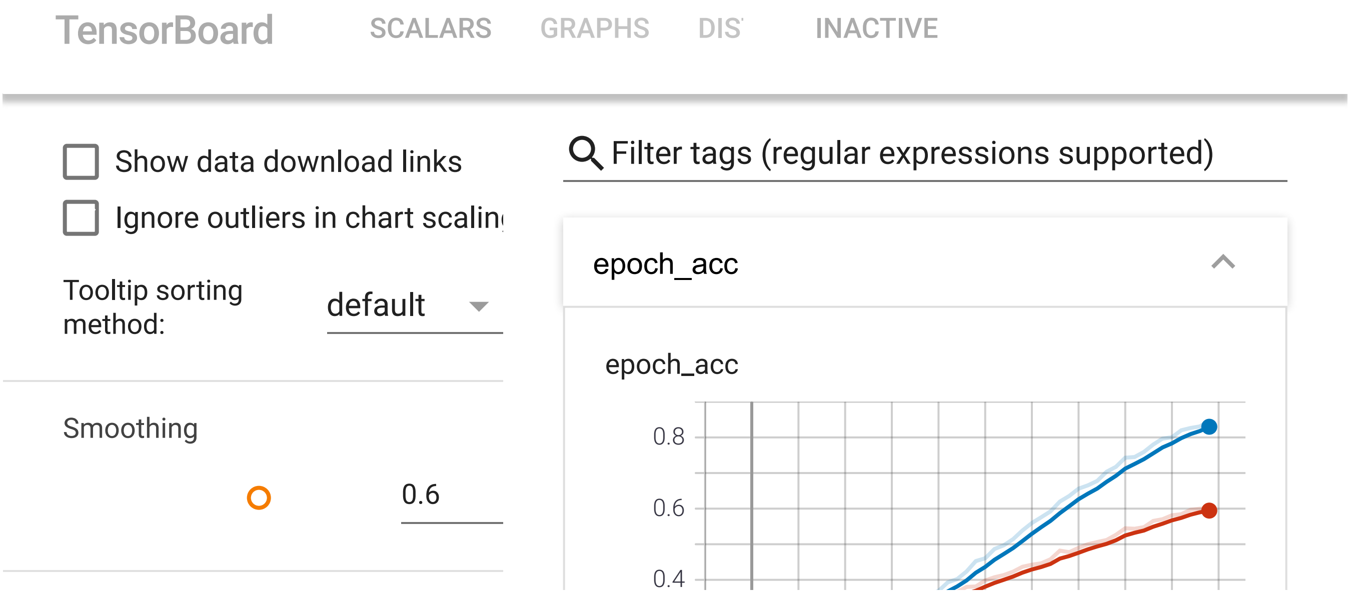
```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:18
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/50
44/44 [==============================] - 13s 137ms/step - loss: 2.1355 - acc: 0.0791

Epoch 00001: val_acc improved from -inf to 0.11248, saving model to /content/drive/M
Epoch 2/50
44/44 [==============================] - 4s 86ms/step - loss: 1.7453 - acc: 0.1079 -

Epoch 00002: val_acc did not improve from 0.11248
Epoch 3/50
44/44 [==============================] - 4s 84ms/step - loss: 1.7068 - acc: 0.1249 -

Epoch 00003: val_acc improved from 0.11248 to 0.15422, saving model to /content/driv
Epoch 4/50
44/44 [==============================] - 4s 83ms/step - loss: 1.5513 - acc: 0.1574 -

Epoch 00004: val_acc improved from 0.15422 to 0.18102, saving model to /content/driv
Epoch 5/50
44/44 [==============================] - 4s 87ms/step - loss: 1.5830 - acc: 0.1741 -

Epoch 00005: val_acc improved from 0.18102 to 0.19947, saving model to /content/driv
Epoch 6/50
44/44 [==============================] - 4s 86ms/step - loss: 1.5077 - acc: 0.1943 -

Epoch 00006: val_acc did not improve from 0.19947
Epoch 7/50
44/44 [==============================] - 4s 87ms/step - loss: 1.4761 - acc: 0.2007 -

Epoch 00007: val_acc improved from 0.19947 to 0.21046, saving model to /content/driv
Epoch 8/50
44/44 [==============================] - 4s 82ms/step - loss: 1.4629 - acc: 0.2121 -

Epoch 00008: val_acc improved from 0.21046 to 0.21837, saving model to /content/driv
Epoch 9/50
44/44 [==============================] - 4s 87ms/step - loss: 1.4006 - acc: 0.2221 -

Epoch 00009: val_acc improved from 0.21837 to 0.22803, saving model to /content/driv
Epoch 10/50
44/44 [==============================] - 4s 85ms/step - loss: 1.3672 - acc: 0.2314 -
```

```
Epoch 00010: val_acc improved from 0.22803 to 0.23418, saving model to /content/driv
Epoch 11/50
44/44 [==============================] - 4s 90ms/step - loss: 1.3000 - acc: 0.2413 -

Epoch 00011: val_acc improved from 0.23418 to 0.24033, saving model to /content/driv
Epoch 12/50
44/44 [==============================] - 4s 87ms/step - loss: 1.2470 - acc: 0.2575 -

Epoch 00012: val_acc improved from 0.24033 to 0.25439, saving model to /content/driv
Epoch 13/50
44/44 [==============================] - 4s 88ms/step - loss: 1.2385 - acc: 0.2644 -

Epoch 00013: val_acc improved from 0.25439 to 0.25615, saving model to /content/driv
Epoch 14/50
44/44 [==============================] - 4s 81ms/step - loss: 1.1766 - acc: 0.2807 -
```

TensorB

```
%load_ext tensorboard
```

```
%tensorboard --logdir '/content/drive/MyDrive/CaseStudy2/Model-1/Logs/fit_model2/'
```



□ Show data download links

□ Ignore outliers in chart scalin

Tooltip sorting method: default

Smoothing 0.6

Observation.

1. The Tensor board clear stats the there is gradual increasing the accuracy of the model and decrease in loss.
2. Due to less size of the source data ,the validation accuracy is keep increasing but it stays at particular limit

Write a regex to filter runs

epoch_loss

```
#Saving the Model Weights
model.save_weights('/content/drive/MyDrive/CaseStudy2/Model/Model_weights.h5')
```

20210515-121241/train

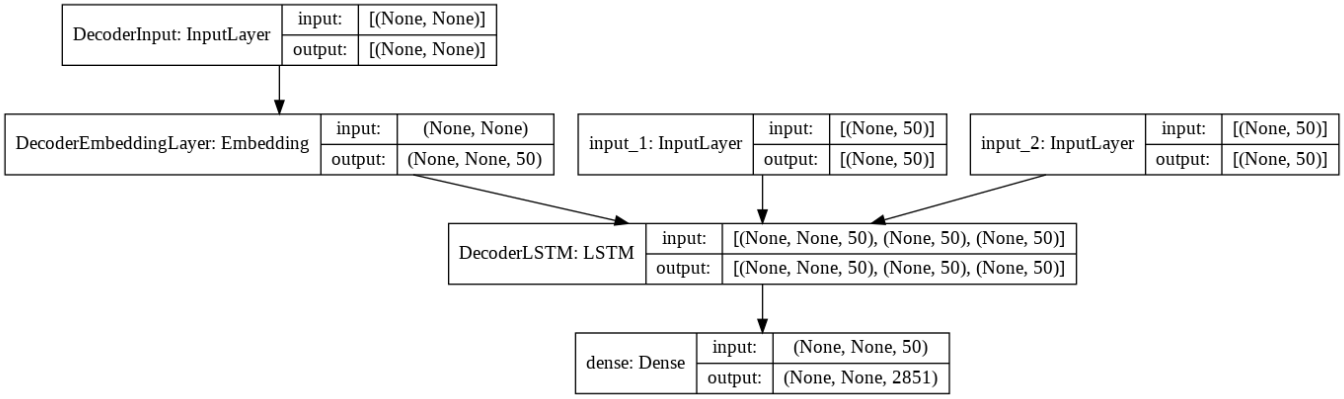
```
model.load_weights('/content/drive/MyDrive/CaseStudy2/Model/nmt_weights.h5')
```

Inference Setup

Model-1/Logs/fit_model2/

```
# Encode the input sequence
encoder_model = Model(encoder_inputs, encoder_states)

# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(300,))
decoder_state_input_c = Input(shape=(300,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]

#dec_emb2= Embedding(vocab_size_eng+1, 50, mask_zero = True,name="DecoderEmbeddingLayer")(dec
dec_emb2=dec_emb_layer(decoder_inputs)
# To predict the next word in the sequence, set the initial states to the states from the pre

decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=decoder_states_in
decoder_states2 = [state_h2, state_c2]
```

```
decoder_states2 = [state_h2, state_c2]
decoder_outputs2 = decoder_dense(decoder_outputs2) # A dense softmax layer to generate prob d

# Final decoder model
decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs2] + decoder_states2)


from keras.utils import plot_model
plot_model(decoder_model, to_file='modelsummary.png', show_shapes=True, show_layer_names=True
```



```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    states_value = encoder_model.predict(input_seq)
    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))
    # Populate the first character of target sequence with the start character.
    target_seq[0, 0] = tknizer_english.word_index['start']
    stop_condition = False
    result=''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] +states_value)
        # Sample a token
        predicted_id = np.argmax(output_tokens[0, -1, :])
        result += tknizer_english.index_word[predicted_id] + ' '
        if (tknizer_english.index_word[predicted_id] == 'end' or
           len(result) > 50):
            stop_condition = True
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = predicted_id

        # Update states
        # update states
        states_value = [h, c]

    return result


#This function used to convert the source sentence into respective tensors and getting the de
def evaluate(sentence):
  sentence = preprocess(sentence)
  result=''
  predicted_list=[]
  try:
    inputs = [tknizer_normal_text.word_index[i] for i in sentence.split(' ')]
    inputs =  tf.keras.preprocessing.sequence.pad_sequences([inputs],maxlen=50,padding='post'
    inputs = tf.convert_to_tensor(inputs)
    result=decode_sequence(inputs)
    predicted_list.append(result)
    print("Source Sentence:-->",sentence)
    print("predicted result:-->",result)
    print("="*50)
    return result,sentence
  except KeyError as e:
    pass
```

Top 5 predicted Sentences from Validation

```
InputText=list(validation['NormalizedText'].head(5).values)
for input in InputText:
  Predicted_list=evaluate(input)
```

```
Source Sentence:--> ohio okey ya sent you a chinese new year poem you too enjoy your chi
predicted result:--> oh okay ya sent you a new year poem it is good birthday
==================================================
Source Sentence:--> buckeye state they were just commenting on its popularity hey your c
predicted result:--> oh they were just commenting on my tests and only in
==================================================
Source Sentence:--> wow so other shuhui and ace later on maybe going to take exposure at
predicted result:--> so so early tomorrow you are going to my house but
==================================================
Source Sentence:--> ok haha by the mode you cut short hair is breadth already or not pre
predicted result:--> okay then when is the bidding is the bidding is very
==================================================
```

Encoder and Decoder with Attention machanisam

```
class Encoder(tf.keras.Model):
    '''
    Encoder model -- That takes a input sequence and returns output sequence
    '''
    def __init__(self,vocab_size,embedding_size,lstm_size,input_length):

        super().__init__()
        self.enc_units=lstm_size
        self.vocab_size = vocab_size
        self.embedding_dim = embedding_size
        self.input_length = input_length
        self.embedding = tf.keras.layers.Embedding(input_dim=self.vocab_size, output_dim=self
                                 input_length=self.input_length,
                           mask_zero=True, name="embedding_layer_encoder",weights=[Encoder_em
        self.lstm = LSTM(self.enc_units, return_state=True, return_sequences=True, name="Enco


    def call(self,input_sequence,states):
        embedding= self.embedding(input_sequence)
        self.lstm_output, self.lstm_state_h,self.lstm_state_c= self.lstm(embedding, initial_s

        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
      '''
      return tf.zeros((batch_size, self.enc_units)),tf.zeros((batch_size, self.enc_units))



class Attention(tf.keras.layers.Layer):

  '''
    Class the calculates score based on the scoring_function using Bahdanu attention mechanis
  '''
  def __init__(self,scoring_function, att_units):

    super(Attention, self).__init__()

    self.scoring_function=scoring_function

    # Please go through the reference notebook and research paper to complete the scoring fun

    if self.scoring_function=='dot':
      # Intialize variables needed for Dot score function here
      pass
    if scoring_function == 'general':
      self.dense = tf.keras.layers.Dense(att_units)

      # Intialize variables needed for General score function here
      pass
    elif scoring_function == 'concat':
```

```
    elif scoring_function == 'concat':

      self.dense = tf.keras.layers.Dense(att_units, activation='tanh')
      self.dense1 = tf.keras.layers.Dense(1)

      # Intialize variables needed for Concat score function here
      pass


  def call(self,decoder_hidden_state,encoder_output):

    if self.scoring_function == 'dot':
        decoder_hidden_state=tf.expand_dims(decoder_hidden_state, 1)
        score = tf.matmul(decoder_hidden_state,encoder_output,transpose_b=True)
        attention_weights = tf.keras.activations.softmax(score, axis=-1)
        context_vector = tf.matmul(attention_weights, encoder_output)

        context_vector=tf.reduce_sum(context_vector, axis=1)

        attention_weights=tf.reduce_sum(attention_weights, axis=1)
        attention_weights=tf.expand_dims(attention_weights, 1)

        return context_vector,attention_weights

        # Implement Dot score function here
        pass

    elif self.scoring_function == 'general':
        decoder_hidden_state=tf.expand_dims(decoder_hidden_state, 1)

        score = tf.matmul(decoder_hidden_state, self.dense(
                encoder_output), transpose_b=True)

        attention_weights = tf.keras.activations.softmax(score, axis=-1)

        context_vector = tf.matmul(attention_weights, encoder_output)

        context_vector=tf.reduce_sum(context_vector, axis=1)

        attention_weights=tf.reduce_sum(attention_weights, axis=1)

        attention_weights=tf.expand_dims(attention_weights, 1)

        return context_vector,attention_weights
      # Implement General score function here
        pass

    elif self.scoring_function == 'concat':

      decoder_hidden_state=tf.expand_dims(decoder_hidden_state, 1)

      decoder_hidden_state = tf.tile(
                decoder_hidden_state, [1,50, 1])

      score = self.dense1(
                self.dense(tf.concat((decoder_hidden_state, encoder_output), axis=-1)))

      score = tf.transpose(score, [0, 2, 1])

      attention_weights = tf.keras.activations.softmax(score, axis=-1)

      context_vector = tf.matmul(attention_weights, encoder_output)

      context_vector=tf.reduce_sum(context_vector, axis=1)

      attention_weights=tf.reduce_sum(attention_weights, axis=1)

      attention_weights=tf.expand_dims(attention_weights, 1)

      return context_vector,attention_weights

      # Implement General score function here
      pass


class One_Step_Decoder(tf.keras.Model):

      def __init__(self,tar_vocab_size, embedding_dim, input_length, dec_units ,score_fun ,at
            super(One_Step_Decoder, self).__init__()
            self.dec_units=dec_units
            self.vocab_size = tar_vocab_size
            self.embedding_dim = embedding_dim
            self.input_length = input_length
            self.attention = Attention(score_fun, dec_units)
            self.embedding = tf.keras.layers.Embedding(input_dim=self.vocab_size, output_dim=
                                   name="embedding_layer_encoder",weights=[Decoder_embedding_

            self.lstm = LSTM(self.dec_units, return_state=True, return_sequences=True, name="
           # Initialize decoder embedding layer, LSTM and any other objects needed
            self.DenseLayer = tf.keras.layers.Dense(self.vocab_size)


      def call(self,input_to_decoder, encoder_output, state_h,state_c):
        embedding= self.embedding(input_to_decoder)

        context_vector,attention_weights =self.attention(state_h,encoder_output)

        context_vector=tf.expand_dims(context_vector, 1)
        lstm_input = tf.concat(
                [tf.squeeze(context_vector, 1), tf.squeeze(embedding, 1)], 1)

        states=[state_h,state_c]

        lstm_input=tf.expand_dims(lstm_input, 1)
        self.lstm_output, self.lstm_state_h,self.lstm_state_c= self.lstm(lstm_input, initial_

        Output=self.DenseLayer(self.lstm_output)

        Output=tf.reduce_sum(Output, axis=1)

        return Output,self.lstm_state_h,self.lstm_state_c,attention_weights,context_vector



class Decoder(tf.keras.Model):
    def __init__(self,out_vocab_size, embedding_dim, input_length, dec_units ,score_fun ,att_

        super(Decoder, self).__init__()
        self.out_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

        #Intialize necessary variables and create an object from the class onestepdecoder
        self.onestepdecoder=One_Step_Decoder(self.out_vocab_size, self.embedding_dim,  self.i

    def call(self, input_to_decoder,encoder_output,decoder_hidden_state,decoder_cell_state ):


        all_outputs=tf.TensorArray(tf.float32,size=self.input_length,name="outputArray")

        for i in range(self.input_length):
          decoder_input = tf.expand_dims(input_to_decoder[:, i], 1)
          output,decoder_hidden_state,decoder_cell_state,attention_weights,context_vector=sel

          all_outputs=all_outputs.write(i,output)

        all_outputs=tf.transpose(all_outputs.stack(),[1,0,2])

        return all_outputs



class MyModel(tf.keras.Model):
    def __init__(self, encoder_inputs_length,decoder_inputs_length, output_vocab_size,score_f
        super().__init__() # https://stackoverflow.com/a/27134600/4084039
        self.batch_size=batch_size
        self.score_fun=score_fun
        self.attn_units=attn_units
        self.encoder = Encoder(vocab_size=vocab_size_normal_text+1, embedding_size=300,lstm_s
        self.decoder = Decoder(out_vocab_size=vocab_size_eng+1, embedding_dim=300, input_leng
    def call(self, data):
        input,output = data[0], data[1]


        initial_state=self.encoder.initialize_states(self.batch_size)
        encoder_output, encoder_h, encoder_c = self.encoder(input,initial_state)
        decoder_output                       = self.decoder(output,encoder_output, encoder_h,
        return decoder_output
```

Loss function

```
def lossfunction(y_true, y_pred):

    crossentropy = tf.keras.losses.SparseCategoricalCrossentropy(
        from_logits=True)

    mask = tf.math.logical_not(tf.math.equal(y_true, 0))
    mask = tf.cast(mask, dtype=tf.int64)
    loss = crossentropy(y_true, y_pred, sample_weight=mask)

    return loss
```

```
from tensorflow.keras import backend as K
def accuracy(y_true, y_pred):

    pred_value= K.cast(K.argmax(y_pred, axis=-1), dtype='float32')
    true_value = K.cast(K.equal(y_true, pred_value), dtype='float32')

    mask = K.cast(K.greater(y_true, 0), dtype='float32')
    n_correct = K.sum(mask * true_value)
    n_total = K.sum(mask)

    return n_correct / n_total
```

```
import datetime
from tensorflow.keras.callbacks import ModelCheckpoint,LearningRateScheduler,EarlyStopping,Te
earlystop = EarlyStopping(monitor='val_loss', min_delta=0.001, patience=5, verbose=1)
filepath="/content/drive/MyDrive/CaseStudy2/Model5_Att/weights-{epoch:02d}-{val_accuracy:.4f}
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',  verbose=4, save_best
logdir = "/content/drive/MyDrive/CaseStudy2/Model5_Att/Logs/fit_model2/" + datetime.datetime.
_
train_summary_writer = tf.summary.create_file_writer(logdir)
tensorboard_callback = TensorBoard(log_dir=logdir,histogram_freq=1,profile_batch = 100000000)
callback_list = [checkpoint,tensorboard_callback]
```

```
model  = MyModel(encoder_inputs_length=50,decoder_inputs_length=50,output_vocab_size=vocab_si

optimizer = tf.keras.optimizers.Adam()
```

```
model.compile(optimizer=optimizer,loss=lossfunction,metrics=[accuracy])
train_steps=train.shape[0]//50
valid_steps=validation.shape[0]//50

Output=model.fit_generator(train_dataloader,steps_per_epoch=train_steps,epochs=25, validation
```

```
    Epoch 00011: val_accuracy improved from 0.42143 to 0.46834, saving model to /content
    Epoch 12/25
    88/88 [==============================] - 16s 182ms/step - loss: 0.6690 - accuracy: 0

    Epoch 00012: val_accuracy improved from 0.46834 to 0.52856, saving model to /content
    Epoch 13/25
    88/88 [==============================] - 16s 180ms/step - loss: 0.5729 - accuracy: 0

    Epoch 00013: val_accuracy improved from 0.52856 to 0.56838, saving model to /content
    Epoch 14/25
    88/88 [==============================] - 16s 180ms/step - loss: 0.4929 - accuracy: 0

    Epoch 00014: val_accuracy improved from 0.56838 to 0.60940, saving model to /content
    Epoch 15/25
    88/88 [==============================] - 16s 180ms/step - loss: 0.4159 - accuracy: 0

    Epoch 00015: val_accuracy improved from 0.60940 to 0.65181, saving model to /content
    Epoch 16/25
    88/88 [==============================] - 16s 180ms/step - loss: 0.3403 - accuracy: 0

    Epoch 00016: val_accuracy improved from 0.65181 to 0.67310, saving model to /content
    Epoch 17/25
    88/88 [==============================] - 16s 180ms/step - loss: 0.2887 - accuracy: 0

    Epoch 00017: val_accuracy improved from 0.67310 to 0.67348, saving model to /content
    Epoch 18/25
    88/88 [==============================] - 16s 181ms/step - loss: 0.2540 - accuracy: 0

    Epoch 00018: val_accuracy improved from 0.67348 to 0.70884, saving model to /content
    Epoch 19/25
    88/88 [==============================] - 16s 181ms/step - loss: 0.2118 - accuracy: 0

    Epoch 00019: val_accuracy improved from 0.70884 to 0.72376, saving model to /content
    Epoch 20/25
    88/88 [==============================] - 16s 179ms/step - loss: 0.1747 - accuracy: 0

    Epoch 00020: val_accuracy improved from 0.72376 to 0.72681, saving model to /content
    Epoch 21/25
    88/88 [==============================] - 16s 180ms/step - loss: 0.1694 - accuracy: 0

    Epoch 00021: val_accuracy improved from 0.72681 to 0.73860, saving model to /content
    Epoch 22/25
    88/88 [==============================] - 16s 180ms/step - loss: 0.1455 - accuracy: 0

    Epoch 00022: val_accuracy improved from 0.73860 to 0.75954, saving model to /content
    Epoch 23/25
    88/88 [==============================] - 16s 179ms/step - loss: 0.1256 - accuracy: 0

    Epoch 00023: val_accuracy improved from 0.75954 to 0.76909, saving model to /content
    Epoch 24/25
    88/88 [==============================] - 16s 179ms/step - loss: 0.1050 - accuracy: 0
```

```
Epoch 00024: val_accuracy improved from 0.76909 to 0.78215, saving model to /content
Epoch 25/25
88/88 [==============================] - 16s 179ms/step - loss: 0.0850 - accuracy: 0

Epoch 00025: val_accuracy improved from 0.78215 to 0.79028, saving model to /content
```

Double-click (or enter) to edit

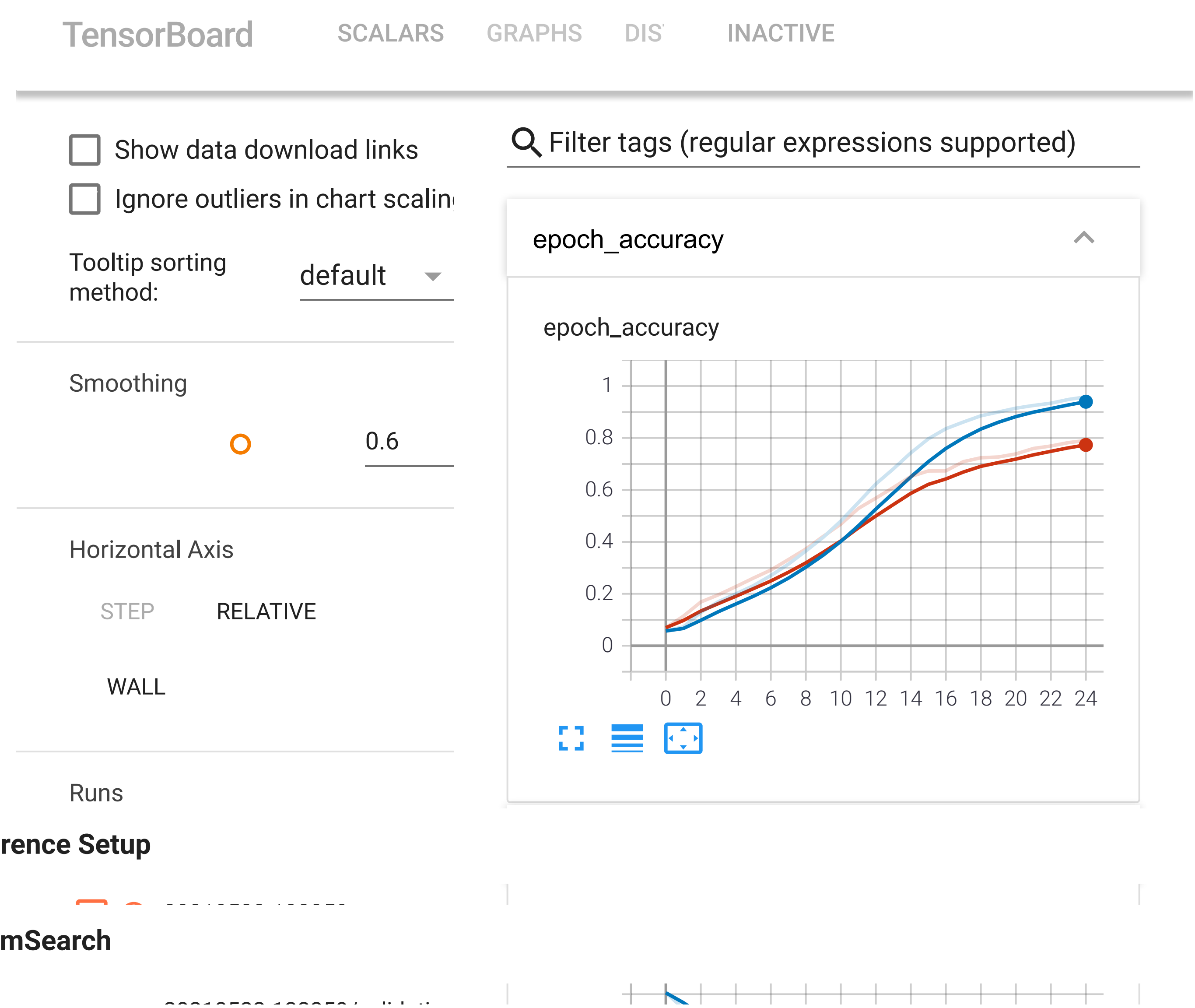
```
Probabilities=model.predict(validation_dataloader)
```

```
model.summary()
```

```
Model: "my_model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
encoder_1 (Encoder)          multiple                  4312800   
_________________________________________________________________
decoder_1 (Decoder)          multiple                  2967439   
=================================================================
Total params: 7,280,239
Trainable params: 7,280,239
Non-trainable params: 0
_________________________________________________________________
```

Tensor Board-2

```
%tensorboard --logdir '/content/drive/MyDrive/CaseStudy2/Model5_Att/Logs/fit_model2/'
```



TensorBoard SCALARS GRAPHS DIS INACTIVE

□ Show data download links

□ Ignore outliers in chart scalin

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis

STEP RELATIVE WALL

Runs

Filter tags (regular expressions supported)

epoch_accuracy

Inference Setup

BeamSearch

```
# https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/

from math import log
from numpy import array
from numpy import argmax

# beam search
def beam_search_decoder(data, k):
  sequences = [[list(), 0.0]]
  for row in data:
    all_candidates = list()
    for i in range(len(sequences)):
      seq, score = sequences[i]
      for j in range(len(row)):
        try:
          candidate = [seq + [j], score - log(row[j])]
          all_candidates.append(candidate)
        except ValueError as e:
          candidate = [seq + [j], 0]
          all_candidates.append(candidate)
```

```
   # order all candidates by score
  ordered = sorted(all_candidates, key=lambda tup:tup[1])
  # select k best
  sequences = ordered[:k]
 return sequences
```

```
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker

def evaluate(sentence):

  max_length_targ=10
  max_length_inp=50


  attention_plot = np.zeros((max_length_targ, max_length_inp))
  sentence = preprocess(sentence)
  inputs = [tknizer_normal_text.word_index[i] for i in sentence.split(' ')]
  inputs =  tf.keras.preprocessing.sequence.pad_sequences([inputs],maxlen=max_length_inp,pa
  inputs = tf.convert_to_tensor(inputs)
  result = ''
  #hidden = [tf.zeros((1, 20))]
  initial_state=model.layers[0].initialize_states(batch_size=1)
  encoder_output, encoder_h, encoder_c = model.layers[0](inputs,initial_state)
  dec_hidden = encoder_h
  dec_cellstate= encoder_c
  dec_input = tf.expand_dims([tknizer_english.word_index['start']], 0)
  for t in range(max_length_targ):
    Output, dec_hidden,dec_cellstate,attention_weights,context_vector = model.layers[1].o
    #Beam Search Decoder
    Result_beam_list=beam_search_decoder(Output,k=1)
    Result_beam=Result_beam_list[0][0]
    # storing the attention weights to plot later on
    attention_weights = tf.reshape(attention_weights, (-1, ))
    attention_plot[t] = attention_weights.numpy()
    predicted_id = tf.argmax(Output[0]).numpy()
    #Predicted ID using beam search decoder
    result += tknizer_english.index_word[Result_beam[0]] + ' '
    if tknizer_english.index_word[predicted_id] == 'end':
      return result, sentence, attention_plot
    # the predicted ID is fed back into the model
    dec_input = tf.expand_dims([predicted_id], 0)
  return result, sentence, attention_plot
```

Plotting Mechanisam

```
def plot_attention(attention, sentence, predicted_sentence):
  fig = plt.figure(figsize=(10,10))
```

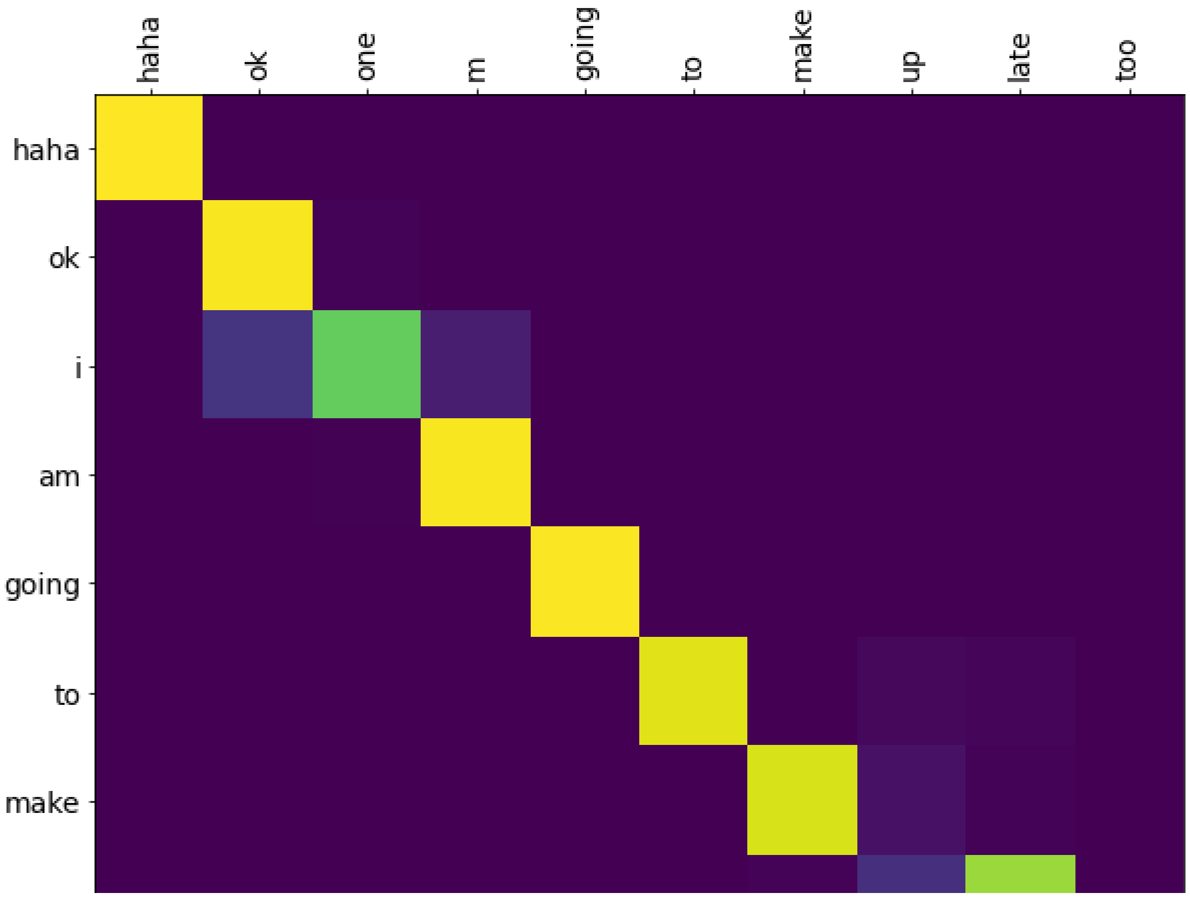
```
    ax = fig.add_subplot(1, 1, 1)
    ax.matshow(attention, cmap='viridis')
    fontdict = {'fontsize': 14}
    ax.set_xticklabels([''] + sentence, fontdict=fontdict, rotation=90)
    ax.set_yticklabels([''] + predicted_sentence, fontdict=fontdict)
    ax.xaxis.set_major_locator(mticker.MultipleLocator(1))
    ax.yaxis.set_major_locator(mticker.MultipleLocator(1))
    plt.show()


def translate(sentence):
    result, sentence, attention_plot = evaluate(sentence)
    print('Input: %s' % (sentence))
    print('Predicted translation: {}'.format(result))
    print("-"*50)
    attention_plot = attention_plot[:len(result.split(' ')), :len(sentence.split(' '))]
    plot_attention(attention_plot, sentence.split(' '),    result.split(' '))
    return result
```

```
validation.head(10)
```

	NormalizedText	Original_English_Text_inp	Original_English_Text_out
110	huh weerakoon oh poplawski that s the wooden ...	<start> huh oh that is the wooden one right th...	huh oh that is the wooden one right the alumin...
1991	wah so far dunno how ey pay	<start> wow so far i do not know how they pay	wow so far i do not know how they pay <end>
1654	get worth of free smsmms for months for just...	<start> get worth of free sms or mms for mon...	get worth of free sms or mms for months for ...
939	louis comfort tiffany hither from gek undertak...	<start> tiffany here from gek project group ar...	tiffany here from gek project group are you gu...
1811	yun i exactly realised i forget to zip my bloo...	<start> yun i just realised i forgot to zip my...	yun i just realised i forgot to zip my pants i...
1820	ok then when are you come back have a safe stu...	<start> okay then when are you coming back hav...	okay then when are you coming back have a safe...
1691	huh don t make never open or	<start> huh do not have never	huh do not have never open or

```
result=translate("haha ok one m going to make up late too")
```

```
Input: haha ok one m going to make up late too
Predicted translation: haha ok i am going to make late too too
--------------------------------------------------
```



Blue Score on validation data using Beam Search

```
import nltk.translate.bleu_score as bleu
def BleuScore(validation):
    input=list(validation['NormalizedText'])
    Y_true=list(validation['Original_English_Text_out'])
    bleuscores=[]
    for i in range(len(input)):
        try:
            result, sentence, attention_plot = evaluate(input[i])
        except KeyError as e:
            pass
    bleuscores.append(bleu.sentence_bleu(Y_true[i], result))
    return sum(bleuscores)/len(bleuscores)

AvearageScore=BleuScore(validation)

print("Avearage Bleuscore for dot score function :",AvearageScore)
```

```
Avearage Bleuscore for dot score function : 0.6844861471686758
/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)
```

Steps Performed:

1.Extracted scoial media text data(normal text data) and original english data from the given source file

2.Used data agumentation using NLPAUG library and used two types of augmentation methods 1.Synonym Agumentation and 2.FastText agumentation for the words and generated nearly 4000 data points which then concatenated with source data points and overall data set size is 6000 points.

3.Preprocessed data using re module and removed all puntuation marks and other special symbols.

4.Created embedding weights using fastext model and used this weights in enbedding layer in neural network

5. Using Data generators trained encoder and decoder model and get the train accuracy 85% and test accuracy 60%
6. Trained one more neural network model using Bhendu attention mechanisam and it gives 95% train accuracy and 75% test accuracy.
7. The Average Blue score of the neural network using beam search is 68% and it is pretty good

✓ 5s completed at 10:16 PM ● ×