###MACHINE LEARNING ASSIGNEMNET Grammatical Error Correction

Grammatical error correction is the task of automatically correcting grammatical errors in a text.

A grammatical error correction system takes an erroneous sentence as input and is expected to

find all the above errors to transform the sentence into the corrected version. For example –

- Incorrect Sentence 1: “She see Tom is catched by policeman in park at last night.”

- Corrects Sentence 1: “She saw Tom caught by a policeman in the park last night.”

- Incorrect Sentence 2: “It is not of only your business.”

- Corrects Sentence 2: “It is none of your business.”

In [48]:

```python
! pip install jsonlines
import pandas as pd
import numpy as np
import os
import jsonlines
import re
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from tqdm import tqdm
from nltk.corpus import stopwords
from wordcloud import WordCloud
import nltk
from nltk.tokenize import word_tokenize
from collections import Counter
from tqdm import tqdm
import tensorflow as tf
from  tensorflow.keras.preprocessing.sequence import pad_sequences
from  sklearn.model_selection import train_test_split
from tqdm import tqdm
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers
import nltk.translate.bleu_score as bleu
from tensorflow.keras import Model
```

```
Requirement already satisfied: jsonlines in /usr/local/lib/python3.7/dist-pa
ckages (3.0.0)
Requirement already satisfied: attrs>=19.2.0 in /usr/local/lib/python3.7/dis
t-packages (from jsonlines) (21.4.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.
7/dist-packages (from jsonlines) (3.10.0.2)
```

###Dataset Used

1. Lang-8
2. NUS Social Media Text Normalization and Translation Corpus

We would be combining both the datasets to train out model.

####Loading the dataset from my google drive.

In [49]:

```
f1 = open("/content/drive/MyDrive/DATASET/entries.train")
lines1 = f1.readlines()
inp1 = []
tgt1 = []

for i in lines1:
    lst = i.split("\t")


    if len(lst)>5  :
        inp1.append(lst[-2])
        tgt1.append(lst[-1])
```

In [ ]:

```
'''THIS CELL READS THE SMS_TEXT DATASET FILE AND EXTRACTS INCORRECT AND CORRECT SENTENC


f2 = open("/content/drive/MyDrive/DATASET/en2cn-2k.en2nen2cn","r",encoding="UTF-8") # 

lines2 = f2.readlines()
inp2 = []
tgt2 = []


for i in range(2000):
    inp2.append(lines2[i*3])
    tgt2.append(lines2[i*3+1])
```

In [ ]:

```
'''Combining two datasets'''

df = pd.DataFrame()
df["input"] = inp1+inp2
df["output"] =  tgt1+tgt2
df["y"] = list("1"*len(inp1)) + list("2"*len(inp2))
```

In [ ]:

```
df
```

This function removes the space between the words also between the sentences

```
ca n't ==> can't
I 'm ===> I'm ...etc
```

In [ ]:

```

def remove_spaces(text):
    text = re.sub(r" '(\w)",r"'\1",text)
    text = re.sub(r" \,",",",text)
    text = re.sub(r" \.+",".",text)
    text = re.sub(r" \!+","!",text)
    text = re.sub(r" \?+","?",text)
    text = re.sub(" n't","n't",text)
    text = re.sub("[\(\)\;\_\^\`\/]","",text)

    return text


'''THIS FUNCTION DECONTRACTS THE CONTRACTED WORDS'''
#REF : https://stackoverflow.com/questions/19790188/expanding-english-language-contract

def decontract(text):
    text = re.sub(r"won\'t", "will not", text)
    text = re.sub(r"can\'t", "can not", text)
    text = re.sub(r"n\'t", " not", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)
    return text


'''THIS FUNCTION PREPROCESSES THE TEXT '''
def preprocess(text):
    text = re.sub("\n","",text)
    text = remove_spaces(text)   # REMOVING UNWANTED SPACES
    text = re.sub(r"\.+",".",text)
    text = re.sub(r"\!+","!",text)
    text = decontract(text)    # DECONTRACTION
    text = re.sub("[^A-Za-z0-9 ]+","",text)
    text = text.lower()
    return text
```

In [ ]:

```
'''HERE WE ARE APPLYIN PREPROCESS FUNCTION TO INPUT AND OUTPUT SENTENCES'''

df["processed_input"] = df.input.apply(preprocess)
df["processed_output"] = df.output.apply(preprocess)
```

In [ ]:

```
dataset=df.drop(['input','output'],axis=1)
```

In [ ]:

```
dataset
```

In [ ]:

```
'''CHECK FOR NULL VALUES'''

dataset.info()
```

In [ ]:

```
'''THIS CELL REMOVES ROWS WITH NULL VALUES IN INPUT AND OUTPUT TEXT'''

dataset = dataset[dataset.processed_input.notnull()]
dataset = dataset[dataset.processed_output.notnull()]
```

In [ ]:

```
dataset = dataset.drop_duplicates()
```

In [ ]:

```
dataset
```

In [ ]:

```
dataset.info()
```

Data is preprocced and stored in to the drive with .csv format

In [ ]:

```
dataset[["processed_input","processed_output","y"]].to_csv("/content/drive/MyDrive/DATA
```

###Here in this program I have used Attention Mechanism for grammer correction

####Attention Mechanism Attention Mechanism is a very ingenious idea in Machine Learning which clones the humanistic way of grasping information.It can be implemented using the idea of Recurrent NN

In [ ]:

```

dataset= pd.read_csv("/content/drive/MyDrive/DATASET/processed_data.csv")
dataset.columns = ["enc_input","dec_input","y"]
dataset["dec_output"] = dataset.dec_input
dataset
```

Adding start and end to the o/p sentence for the usefull ness of the decoder

In [ ]:

```

dataset["dec_input"]= "<start> " + dataset["dec_input"]
dataset["dec_output"] =  dataset["dec_output"] + " <end>"
dataset
```

Splitting And Sampling around 100k datapoints

In [ ]:

```
df_sampled = pd.concat((dataset[dataset.y==1].sample(frac= 0.2,random_state=1),dataset
```

In [ ]:

```
## Spliting the data .
df_train ,df_val = train_test_split(df_sampled,test_size=0.2,random_state = 3, stratify
```

In [ ]:

```

df_train["dec_input"].iloc[0]  = df_train.iloc[0]["dec_input"] + " <end>"
df_train
```

In [54]:

```

df_val
```

Out[54]:

	enc_input	dec_input	y	dec_output
71342	in the last stage add the mixture of potato an...	<start> in the last stage add the mixture of p...	1	in the last stage add the mixture of potato an...
395034	of course the experiences of japanese are also...	<start> of course the experiences of the japan...	1	of course the experiences of the japanese are ...
208880	but watching other is play gives me many stimu...	<start> but watching others play is exciting a...	1	but watching others play is exciting after all...
198826	and this is my new tattoos	<start> and these are my new tattoos	1	and these are my new tattoos <end>
313016	i write a diary in english for the first time	<start> i am writing a diary in english for th...	1	i am writing a diary in english for the first ...
...	...	...	...	...
247300	i love family	<start> i love my family	1	i love my family <end>
505055	ok i going soon and also send xyan home at the...	<start> ok i am going soon and also send xyan ...	2	ok i am going soon and also send xyan home at ...
126924	house car people had been washed away by the b...	<start> houses cars and people had been washed...	1	houses cars and people had been washed away by...
53162	on the weekend i had to go back to my home to ...	<start> on the weekend i had to go back to my ...	1	on the weekend i had to go back to my home to ...
10995	she lilves in us and studys to be nurse	<start> she lives in us and is studying to be ...	1	she lives in us and is studying to be a nurse ...

20556 rows × 4 columns

In [ ]:

```
## Here  1000 points are uses from the dataframe as test data.
np.random.seed(5)
df_test = dataset.loc[np.random.choice(np.array([x for x in dataset.index.values if x
df_test
```

In [ ]:

```
## Tokenizer for encoder I/P
tk_inp = Tokenizer()
tk_inp.fit_on_texts(df_train.enc_input.apply(str))
```

In [ ]:

```
## Tokenizer for decoder I/P
tk_out = Tokenizer(filters='!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n' )
tk_out.fit_on_texts(df_train.dec_input.apply(str))
```

This class converts the text data to int seq. and returns the padded seq.

In [ ]:

```

class Dataset :
    def __init__(self, data , tk_inp ,tk_out, max_len):

        self.encoder_inp = data["enc_input"].apply(str).values
        self.decoder_inp = data["dec_input"].apply(str).values
        self.decoder_out = data["dec_output"].apply(str).values
        self.tk_inp = tk_inp
        self.tk_out = tk_out
        self.max_len = max_len

    def __getitem__(self,i):

        self.encoder_seq = self.tk_inp.texts_to_sequences([self.encoder_inp[i]])
        self.decoder_inp_seq = self.tk_out.texts_to_sequences([self.decoder_inp[i]])
        self.decoder_out_seq = self.tk_out.texts_to_sequences([self.decoder_out[i]])
        self.encoder_seq = pad_sequences(self.encoder_seq, padding="post",maxlen = sel
        self.decoder_inp_seq = pad_sequences(self.decoder_inp_seq, padding="post",maxle
        self.decoder_out_seq = pad_sequences(self.decoder_out_seq ,padding="post", max
        return self.encoder_seq ,  self.decoder_inp_seq,  self.decoder_out_seq

    def __len__(self):
        return len(self.encoder_inp)
```

Coverting data into batches

In [ ]:

```
class Dataloader(tf.keras.utils.Sequence):
    def __init__(self,batch_size,dataset):

        self.dataset = dataset
        self.batch_size = batch_size
        self.totl_points = self.dataset.encoder_inp.shape[0]

    def __getitem__(self,i):

        start = i * self.batch_size
        stop = (i+1)*self.batch_size
        batch_enc =[]
        batch_dec_input = []
        batch_dec_out =[]

        for j in range(start,stop):

            a,b,c = self.dataset[j]
            batch_enc.append(a[0])
            batch_dec_input.append(b[0])
            batch_dec_out.append(c[0])

        batch_enc = (np.array(batch_enc))
        batch_dec_input = np.array(batch_dec_input)
        batch_dec_out = np.array(batch_dec_out)

        return [batch_enc , batch_dec_input],batch_dec_out

    def __len__(self):
        return int(self.totl_points/self.batch_size)
```

In [ ]:

```
train_dataset = Dataset(df_train,tk_inp,tk_out,35)
train_dataloader = Dataloader( batch_size = 32, dataset=train_dataset)

val_dataset = Dataset(df_val , tk_inp,tk_out,35)
val_dataloader = Dataloader(batch_size=32 , dataset=val_dataset)
```

Encoder Class

In [ ]:

```
## DEFINING THE ENCODER LAYER AS A FUNCTION
class Encoder(tf.keras.layers.Layer):

    def __init__(self, vocab_size,emb_dims, enc_units, input_length,batch_size):
        super().__init__()

        self.batch_size=batch_size
        self.enc_units = enc_units
        self.embedding= layers.Embedding(vocab_size ,emb_dims)

        self.lstm = layers.LSTM(self.enc_units,return_state= True,return_sequences =  T

    def call(self, enc_input , states):
        emb = self.embedding(enc_input)
        enc_output,state_h,state_c = self.lstm(emb,initial_state=states)
        return enc_output,state_h,state_c
    def initialize(self,batch_size):

        return tf.zeros(shape=(batch_size,self.enc_units)),tf.zeros(shape=(batch_size,s
```

###Attention Mechanism

In [ ]:

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

    def __init__(self,units):
        super().__init__()

        self.dense = layers.Dense(units)

    def call(self,enc_output,dec_state):

        dec_state =  tf.expand_dims(dec_state,axis=-1)
        dense_output = self.dense(enc_output)
        score = tf.matmul(dense_output , dec_state)
        att_weights = tf.nn.softmax(score,axis=1)
        context_vec  = att_weights* enc_output
        context_vec = tf.reduce_sum(context_vec,axis=1)
        return context_vec,att_weights
```

In [ ]:

```
class Onestepdecoder(tf.keras.Model):
    '''THIS MODEL OUTPUTS THE RESULT OF DECODER FOR ONE TIME SETP GIVEN THE INPUT FOR F

    def __init__(self, vocab_size,emb_dims, dec_units, input_len,att_units,batch_size):
        super().__init__()
        self.emb = layers.Embedding(vocab_size,emb_dims,input_length= input_len)
        # ATTENTION LAYER
        self.att = Attention(att_units)
        # LSTM LAYER
        self.lstm = layers.LSTM(dec_units,return_sequences=True,return_state=True)
        # DENSE LAYER
        self.dense = layers.Dense(vocab_size,activation="softmax")

    def call(self, encoder_output , input , state_h,state_c):
        emb = self.emb(input)

        dec_output,dec_state_h,dec_state_c = self.lstm( emb , initial_state = [state_h,
        context_vec,alphas = self.att(encoder_output,dec_state_h)
        dense_input =  tf.concat([tf.expand_dims(context_vec,1),dec_output],axis=-1)
        fc = self.dense(dense_input)
        return fc , dec_state_h , dec_state_c , alphas
```

###Decoder Class

In [ ]:

```
class Decoder(tf.keras.Model):
    '''THIS MODEL PERFORMS THE WHOLE DECODER OPERATION FOR THE COMPLETE SENTENCE'''
    def __init__(self, vocab_size,emb_dims, dec_units, input_len,att_units,batch_size):
        super().__init__()

        self.input_len = input_len
        self.onestepdecoder = Onestepdecoder(vocab_size,emb_dims, dec_units, input_len,

    def call(self,dec_input,enc_output,state_h,state_c):
        current_state_h = state_h
        current_state_c = state_c
        pred = []
        alpha_values = []
        for i in range(self.input_len):
            current_vec = dec_input[:,i]
            current_vec = tf.expand_dims(current_vec,axis=-1)
            dec_output,dec_state_h,dec_state_c,alphas = self.onestepdecoder(enc_output
            current_state_h = dec_state_h
            current_state_c = dec_state_c
            pred.append(dec_output)
            alpha_values.append(alphas)
        output = tf.concat(pred,axis=1)
        alpha_values = tf.concat(alpha_values,axis = -1)
        return output , alpha_values
```

In [ ]:

```
class encoder_decoder(tf.keras.Model):
    '''THIS MODEL COMBINES ALL THE LAYERS AND FORM IN ENCODER DECODER MODEL WITH ATTENT
    def __init__(self,enc_vocab_size,enc_emb_dim,enc_units,enc_input_length,
             dec_vocab_size,dec_emb_dim,dec_units,dec_input_length ,att_units, batch_si
        super().__init__()
        self.batch_size = batch_size
        self.encoder = Encoder(enc_vocab_size, enc_emb_dim,enc_units, enc_input_length,
        self.decoder = Decoder(dec_vocab_size ,dec_emb_dim,dec_units,dec_input_length

    def call(self,data):

        inp1 , inp2 = data
        enc_output, enc_state_h, enc_state_c = self.encoder(inp1,self.encoder.initializ
        dec_output , alphas = self.decoder(inp2 , enc_output,enc_state_h,enc_state_c)
        return dec_output
```

Initailizing model

In [ ]:

```

model = encoder_decoder(enc_vocab_size=len(tk_inp.word_index)+1,
                         enc_emb_dim = 300,
                         enc_units=256,enc_input_length=35,
                         dec_vocab_size =len(tk_out.word_index)+1,
                         dec_emb_dim =300,
                         dec_units=256,
                         dec_input_length = 35,

                         att_units=256,
                         batch_size=32)
```

Defining callbacks and storing the values into the drive for future use

In [ ]:

```

callback =[ tf.keras.callbacks.ModelCheckpoint( "/content/drive/MyDrive/ColabNotebooks/
           tf.keras.callbacks.EarlyStopping(monitor='val_loss',patience=5,verbose=1,mi
            tf.keras.callbacks.TensorBoard("/content/drive/MyDrive/ColabNotebooks/cs2/m
]

train_steps = train_dataloader.__len__()
val_steps  = val_dataloader.__len__()
# Compiling the model
model.compile(optimizer="adam",loss='sparse_categorical_crossentropy')
```

In [ ]:

```
model.fit(train_dataloader, steps_per_epoch=train_steps,epochs= 50,validation_data = va
```

In [50]:

```
%load_ext tensorboard
%tensorboard --logdir /content/drive/MyDrive/ColabNotebooks/cs2/model_save/attention_ge
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard


<IPython.core.display.Javascript object>
```

In [56]:

```
model.build([(32,35),(32,35)])
model.summary()
```

```
Model: "encoder_decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 encoder (Encoder)           multiple                  10583768  
                                                                 
 decoder (Decoder)           multiple                  22835938  
                                                                 
=================================================================
Total params: 33,419,706
Trainable params: 33,419,706
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model.load_weights("/content/drive/MyDrive/ColabNotebooks/cs2/model_save/attention_gerr
```

This function is used in inference time which given any sentence in italian outputs the english sentence and alpha values

In [ ]:

```
def predict(ita_text,model):


    seq = tk_inp.texts_to_sequences([ita_text])
    seq = pad_sequences(seq,maxlen = 20 , padding="post")
    state = model.layers[0].initialize(1)
    enc_output,state_h,state_c= model.layers[0](seq,state)
    pred = []
    input_state_h = state_h
    input_state_c = state_c
    current_vec = tf.ones((1,1))
    alpha_values = []

    for i in range(20):
        fc , dec_state_h ,dec_state_c, alphas = model.layers[1].layers[0](enc_output ,
        alpha_values.append(alphas)
        current_vec = np.argmax(fc , axis = -1)
        input_state_h = dec_state_h
        input_state_c = dec_state_c
        pred.append(tk_out.index_word[current_vec[0][0]])
        if tk_out.index_word[current_vec[0][0]]=="<end>":
              break
    pred_sent = " ".join(pred)
    alpha_values = tf.squeeze(tf.concat(alpha_values,axis=-1),axis=0)
    return  pred_sent , alpha_values
```

Calculating BLUE score

In [ ]:

```
BLEU = []
np.random.seed(1)
test_data = df_val.loc[np.random.choice(df_val.index,size = 2000,replace=False)]
for ind,i in tqdm(test_data.iterrows(),position=0):
    try:
        pred = predict(str(i.enc_input),model)[0].split()
        act = [str(i.dec_output).split()]
        b = bleu.sentence_bleu(act,pred)
        BLEU.append(b)
    except:
          continue
print("BELU = ", np.mean(BLEU))
```

##Testing the model

In [51]:

```
print("INPUT SENTENCE ===> ","it is not of only your business")
print("PREDICTED SENTENCE ===> ",predict("it is not of only your business",model)[0])
print("ACTUAL SENTENCE ===> ","It is none of your business.")
```

```
INPUT SENTENCE ===>  it is not of only your business
PREDICTED SENTENCE ===>  it is not of only your business <end>
ACTUAL SENTENCE ===>  It is none of your business.
```

In [53]:

```
print("INPUT SENTENCE ===> ","She see Tom is catched by policeman in park at last night
print("PREDICTED SENTENCE ===> ",predict("She see Tom is catched by policeman in park a
print("ACTUAL SENTENCE ===> ","She saw Tom caught by a policeman in the park last night
```

```
INPUT SENTENCE ===>  She see Tom is catched by policeman in park at last nig
ht.
PREDICTED SENTENCE ===>  she saw tom is caught by a policeman in the park at
last night <end>
ACTUAL SENTENCE ===>  She saw Tom caught by a policeman in the park last nig
ht.
```

Conclusion: If this model will train for a large number epoches and if proper dataset can be found this model will work better.

In [ ]:

```

```