In [1]:

```
import tensorflow as tf
import pandas as pd
from tqdm import tqdm
import pickle
import matplotlib.pyplot as plt
%matplotlib inline
# import seaborn as sns
import re
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
import seaborn as sns
import pickle
from transformers import AutoTokenizer, TFBertForMaskedLM,TFDistilBertForMaskedLM
import tensorflow as tf
```

```
2023-03-15 10:12:28.121490: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow
binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU in
structions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-03-15 10:12:29.051532: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loade
r.cc:64] Could not load dynamic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: cannot open
shared object file: No such file or directory; LD_LIBRARY_PATH: :/home/josephnadar1998/miniconda
3/envs/tf/lib/
2023-03-15 10:12:29.051662: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loade
r.cc:64] Could not load dynamic library 'libnvinfer_plugin.so.7'; dlerror: libnvinfer_plugin.so.
7: cannot open shared object file: No such file or directory; LD_LIBRARY_PATH: :/home/josephnada
r1998/miniconda3/envs/tf/lib/
2023-03-15 10:12:29.051674: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warni
ng: Cannot dlopen some TensorRT libraries. If you would like to use Nvidia GPU with TensorRT, pl
ease make sure the missing libraries mentioned above are installed properly.
```

Importing all the pickle files

In [2]:

```
[vocab_size_correct,vocab_size_incorrect,correct_tk,incorrect_tk]=pickle.load(open('tokenizer_files.pkl','rb'))
```

In [3]:

```
[train_m_1,test_m_1, validation_m_1]=pickle.load(open('main_data_2.pkl','rb'))
```

In [4]:

```
[train_m_2,test_m_2, validation_m_2]=pickle.load(open('main_data_2_reverse.pkl','rb'))
```

Model:

In [125]:

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

    def __init__(self,inp_vocab_size,embedding_size,lstm_size,input_length):
        super().__init__()
        #Initialize Embedding layer
        #Intialize Encoder LSTM layer
        self.embedding_layer=Embedding(input_dim=inp_vocab_size,output_dim=embedding_size,input_length=input_l
        self.lstm_layer=LSTM(lstm_size, return_sequences=True, return_state=True)
        self.lstm_size=lstm_size

    def call(self,input_sequence,states):
        '''
        This function takes a sequence input and the initial states of the encoder.
        Pass the input_sequence input to the Embedding layer, Pass the embedding layer ouput to encoder_lstm
        returns -- All encoder_outputs, last time steps hidden and cell state
        '''
        input_1=self.embedding_layer(input_sequence)
        output, output_h, output_c=self.lstm_layer(input_1, initial_state=states)
        return output, output_h, output_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,lst
        '''
        output_h, output_c=tf.zeros([batch_size,self.lstm_size]), tf.zeros([batch_size,self.lstm_size])
        return output_h, output_c

#Attention#
class Attention(tf.keras.layers.Layer):
    '''
    Class the calculates score based on the scoring_function using Bahdanu attention mechanism.
    '''
    def __init__(self,scoring_function, att_units):
        super().__init__()
        # Please go through the reference notebook and research paper to complete the scoring functions

        self.att_units=att_units
        self.scoring_function=scoring_function
        self.dot=tf.keras.layers.Dot(axes=(1,2))
        self.mult=tf.keras.layers.Multiply()
        self.add=tf.keras.layers.Add()

        pass

    def call(self,decoder_hidden_state,encoder_output):
      '''
      Attention mechanism takes two inputs current step -- decoder_hidden_state and all the encoder_outputs.
      * Based on the scoring function we will find the score or similarity between decoder_hidden_state and enc
      Multiply the score function with your encoder_outputs to get the context vector.
      Function returns context vector and attention weights(softmax - scores)
      '''
      # Implement Dot score function here
      #print('decoder_hidden_state',tf.expand_dims(decoder_hidden_state,1).shape, 'encoder_output', encoder_out
      alphas=tf.matmul(encoder_output,tf.expand_dims(decoder_hidden_state,-1))
      alphas=tf.nn.softmax(alphas)
      context_vector=alphas*encoder_output
      context_vector=tf.reduce_sum(context_vector, axis=1)
      return context_vector,alphas

class One_Step_Decoder(tf.keras.Model):
    def __init__(self,tar_vocab_size, embedding_dim, input_length, dec_units ,score_fun ,att_units):
        super().__init__()
        # Initialize decoder embedding layer, LSTM and any other objects needed #, mask_zero=True, trainable=Tr
        self.embedding_layer=Embedding(input_dim=tar_vocab_size, output_dim=embedding_dim, input_length=input_
        self.lstm_layer=LSTM(dec_units, return_state=True, return_sequences=True)
        self.att_units=att_units
        self.score_fun=score_fun
        self.tar_vocab_size=tar_vocab_size
```

```
        self.dec_units=dec_units
        self.dense_layer=tf.keras.layers.Dense(tar_vocab_size)
        self.attention=Attention(score_fun,att_units)


    def call(self,input_to_decoder, encoder_output, state_h,state_c):
        '''
        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_di
          B. Using the encoder_output and decoder hidden state, compute the context vector.
          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 state)
          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
        '''
        result=self.embedding_layer(input_to_decoder)
        result=tf.squeeze(result, axis=1)

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

        output_1=tf.concat([context_vector, result],axis=1)
        output_1=tf.expand_dims(output_1,1)

        decoder_outputs, decoder_h, decoder_c=self.lstm_layer(output_1, initial_state=[state_h,state_c])


        final_output=self.dense_layer(decoder_outputs)
        final_output=tf.squeeze(final_output,axis=1)

        return final_output,decoder_h, decoder_c, weights,context_vector

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

      #Intialize necessary variables and create an object from the class onestepdecoder

      self.input_length=input_length
      self.dec_units=dec_units
      self.score_fun=score_fun
      self.att_units=att_units
      self.out_vocab_size=out_vocab_size
      self.embedding_dim=embedding_dim
      self.osd=One_Step_Decoder(tar_vocab_size=self.out_vocab_size, embedding_dim=self.embedding_dim,
                                                                                  input_length=self.input_leng
      pass
    tf.config.run_functions_eagerly(True)
    @tf.function
    def call(self, input_to_decoder,encoder_output,decoder_hidden_state,decoder_cell_state ):

        #Initialize an empty Tensor array, that will store the outputs at each and every time step
        #Create a tensor array as shown in the reference notebook

        #Iterate till the length of the decoder input
            # Call onestepdecoder for each token in decoder_input
            # Store the output in tensorarray
        # Return the tensor array
        #print(input_to_decoder.shape)

        output_array=tf.TensorArray(tf.float32,size=input_to_decoder.shape[1])
        #print('input_to_decoder',input_to_decoder.shape)
        for timestep in range(input_to_decoder.shape[1]):
          #print(input_to_decoder.shape, encoder_output.shape, decoder_hidden_state.shape,decoder_cell_state.sh
          output,decoder_hidden_state,decoder_cell_state,attention_weights,context_vector=self.osd(input_to_dec
          output_array = output_array.write(timestep, output)
          #output_array.write(timestep,output).mark_used()
        #.mark_used()
        all_output=tf.transpose(output_array.stack(), [1,0,2])
        #print(all_output.shape)
        return all_output

class encoder_decoder(tf.keras.Model):
  def __init__(self,inp_vocab_size,out_vocab_size, embedding_size, lstm_size, input_length_l1, input_length_l2,
    super().__init__()
```

```
    #Intialize objects from encoder decoder
    self.encoder_block=Encoder(inp_vocab_size=inp_vocab_size,embedding_size=embedding_size, lstm_size=lstm_size
    self.decoder_block=Decoder(out_vocab_size=out_vocab_size, embedding_dim=embedding_size, input_length=input_
    self.batch_size=batch_size
    pass


  def call(self,data):
    #Intialize encoder states, Pass the encoder_sequence to the embedding layer
    # Decoder initial states are encoder final states, Initialize it accordingly
    # Pass the decoder sequence,encoder_output,decoder states to Decoder
    # return the decoder output
    input_sequence=data[0]
    output_sequence=data[1]
    #print(input_sequence.shape)
    encoder_h, encoder_c=self.encoder_block.initialize_states(self.batch_size)
    encoder_output, encoder_h, encoder_c=self.encoder_block(input_sequence, states=[encoder_h, encoder_c])
    #input_to_decoder,encoder_output,decoder_hidden_state,decoder_cell_state
    dec_h,dec_c=encoder_h, encoder_c
    output_decoder =self.decoder_block(input_to_decoder=output_sequence,encoder_output=encoder_output,decoder_h
    #output_decoder=self.soft_max(output_decoder)

    return output_decoder
```

This model takes input and output in a normal manner(Left to Right)

In [10]:

```
input_vocab_size= vocab_size_incorrect+1
output_vocab_size=vocab_size_correct+1
embedding_size=300
lstm_size=512
input_len=16
output_len=16
dec_units=512
score_fun='dot'
att_units=512
BATCH_SIZE=512

model_1 = encoder_decoder(input_vocab_size,output_vocab_size,embedding_size,lstm_size,input_len,output_len,dec
```

```
2023-03-15 10:12:36.042024: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:
981] successful NUMA node read from SysFS had negative value (-1), but there must be at least on
e NUMA node, so returning NUMA node zero
2023-03-15 10:12:36.186131: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:
981] successful NUMA node read from SysFS had negative value (-1), but there must be at least on
e NUMA node, so returning NUMA node zero
2023-03-15 10:12:36.186976: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:
981] successful NUMA node read from SysFS had negative value (-1), but there must be at least on
e NUMA node, so returning NUMA node zero
2023-03-15 10:12:36.191682: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow
binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU in
structions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-03-15 10:12:36.193972: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:
981] successful NUMA node read from SysFS had negative value (-1), but there must be at least on
e NUMA node, so returning NUMA node zero
2023-03-15 10:12:36.194712: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:
981] successful NUMA node read from SysFS had negative value (-1), but there must be at least on
e NUMA node, so returning NUMA node zero
2023-03-15 10:12:36.195346: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:
981] successful NUMA node read from SysFS had negative value (-1), but there must be at least on
e NUMA node, so returning NUMA node zero
2023-03-15 10:12:38.711546: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:
981] successful NUMA node read from SysFS had negative value (-1), but there must be at least on
e NUMA node, so returning NUMA node zero
2023-03-15 10:12:38.714288: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:
981] successful NUMA node read from SysFS had negative value (-1), but there must be at least on
e NUMA node, so returning NUMA node zero
2023-03-15 10:12:38.714956: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:
981] successful NUMA node read from SysFS had negative value (-1), but there must be at least on
e NUMA node, so returning NUMA node zero
2023-03-15 10:12:38.718268: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1613] Created dev
ice /job:localhost/replica:0/task:0/device:GPU:0 with 14620 MB memory:  -> device: 0, name: Tesl
a V100-SXM2-16GB, pci bus id: 0000:00:04.0, compute capability: 7.0
```

In [11]:

```
model_1.build((None,512,16))
model_1.load_weights('model_2/model_2_epoch_17.h5')
#Left to right
```

This model takes input in reverse and output in a normal manner(Left to Right)

In [13]:

```
model_2 = encoder_decoder(input_vocab_size,output_vocab_size,embedding_size,lstm_size,input_len,output_len,dec
model_2.build((None,512,16))
model_2.load_weights('model_3/model_3_epoch_46.h5')
```

Language Model to Check the Fluency Score

- In the paper, the author have suggested to use a language model to get a fluency score of a sentence at each level to decide whether a sentence needs to be trained again and other aspects.

- In the paper they have used a 5-gram model, but here I have chosen to use a BERT(Transformer) based model.

BERT Model

In [14]:

```
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = TFBertForMaskedLM.from_pretrained("bert-base-uncased")

def p_x(word, sentence):
    inputs = tokenizer(sentence, return_tensors="tf")
    logits = model(**inputs).logits

    #The word I am expecting

    mask_token_index = tf.where((inputs.input_ids == tokenizer.mask_token_id)[0])
    selected_logits = tf.gather_nd(logits[0], indices=mask_token_index)
    selected_logits=tf.math.sigmoid(selected_logits)

    w=tokenizer.encode(word)[1]
    return selected_logits[0][w]
```

```
All model checkpoint layers were used when initializing TFBertForMaskedLM.

All the layers of TFBertForMaskedLM were initialized from the model checkpoint at bert-base-unca
sed.
If your task is similar to the task the model of the checkpoint was trained on, you can already
use TFBertForMaskedLM for predictions without further training.
```

In [15]:

```
def fluency_score(sent):
    words=sent.split(' ')
    l=len(words)
    scores=[]
    context=['[MASK]']
    i=0
    for w in words:
        if i==0:
            #print(w)
            p=p_x(w,'[MASK]')
            context.insert(0,w)
        else:
            #print(w,' '.join(context))
            p=p_x(w,' '.join(context))
        i+=1
        context.insert(0,w)
        #print(p)
        scores.append(p)
    h_x=-(sum(scores)/l)
    #print(h_x)
    #print(scores)
    f_score=1/(1+h_x)
    return f_score
```

In [16]:

```
import time
i=time.time()
print('Fluency Score is',fluency_score('Today is a great day'))
print(time.time()-i)
```

```
Fluency Score is tf.Tensor(2.651366, shape=(), dtype=float32)
0.8333444595336914
```

In [17]:

```
i=time.time()
print('Fluency Score is',fluency_score('Today great day'))
print(time.time()-i)
```

```
Fluency Score is tf.Tensor(1.7072877, shape=(), dtype=float32)
0.49550795555114746
```

Using more faster transformers

- The BERT is taking more time, and hence using DistilBERT which has less parameter and hence is fast.

In [169]:

```
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
model = TFDistilBertForMaskedLM.from_pretrained("distilbert-base-uncased")

def p_x(word, sentence):
    inputs = tokenizer(sentence, return_tensors="tf")
    logits = model(**inputs).logits

    #The word I am expecting

    mask_token_index = tf.where((inputs.input_ids == tokenizer.mask_token_id)[0])
    selected_logits = tf.gather_nd(logits[0], indices=mask_token_index)
    selected_logits=tf.math.sigmoid(selected_logits)

    w=tokenizer.encode(word)[1]
    return selected_logits[0][w]
```

```
Some layers from the model checkpoint at distilbert-base-uncased were not used when initializing
TFDistilBertForMaskedLM: ['activation_13']
- This IS expected if you are initializing TFDistilBertForMaskedLM from the checkpoint of a mode
l trained on another task or with another architecture (e.g. initializing a BertForSequenceClass
ification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing TFDistilBertForMaskedLM from the checkpoint of a
model that you expect to be exactly identical (initializing a BertForSequenceClassification mode
l from a BertForSequenceClassification model).
All the layers of TFDistilBertForMaskedLM were initialized from the model checkpoint at distilbe
rt-base-uncased.
If your task is similar to the task the model of the checkpoint was trained on, you can already
use TFDistilBertForMaskedLM for predictions without further training.
```

In [170]:

```
import time
i=time.time()
fluency_score('Today is a great day')
print(time.time()-i)
```

```
0.42206382751464844
```

In [171]:

```
i=time.time()
print('Fluency Score is',fluency_score('Today great day'))
print(time.time()-i)
```

```
Fluency Score is tf.Tensor(2.579536, shape=(), dtype=float32)
0.269848108291626
```

DistilBERT takes half the time of BERT

Note:

- Here we can clearly see, that a poor sentence gets a low fluency score, whereas a good sentence gets a high fluency score.

Checking Individual Performance for each Model alone

1. Left to Right Model

- Where the sentence are feeded from left to right.

In [165]:

```
def predict_m1(input_sentence):

    words=[]
    input_sentence=[input_sentence]
    batch_size=1
    tokenized_sent=incorrect_tk.texts_to_sequences(input_sentence)
    #print(tokenized_sent)
    padded_sent=tf.keras.utils.pad_sequences(tokenized_sent, maxlen=16,padding='post' )
    encoder_h, encoder_c=model_1.layers[0].initialize_states(batch_size)
    encoder_output,encoder_h, encoder_c= model_1.layers[0](padded_sent, states=[encoder_h, encoder_c])

    start_index=correct_tk.word_index.get('<start>')
    end_index=correct_tk.word_index.get('<end>')
    for i in range(20):
        decoder_output, decoder_h, decoder_c, attention_weights, context_vector = model_1.layers[1].osd(tf.conv

        output_index=np.argmax(decoder_output[0])
        start_index=output_index
        #print(output_index)
        encoder_h, encoder_c=decoder_h, decoder_c

        if output_index==end_index:
            break;

        if i==0:
            words.append(correct_tk.index_word[output_index])
        else:
            if words[-1]!=correct_tk.index_word[output_index]:
                words.append(correct_tk.index_word[output_index])

        #print(list(tknizer_eng.word_index.keys())[output_index])


    return ' '.join(words)

i='she like dance'
print(i)
predict_m1(i)
```

```
she like dance
```

Out[165]:

```
'she likes to dance'
```

In [120]:

```
#Results on using Capital Sentences and fullstops
samples=test_m_1['incorrect'].sample(1000)
predicted_samples_model_2=samples.apply(predict_m1)

import nltk.translate.bleu_score as bleu
from tqdm import tqdm

blue_scores=[]

for i in tqdm(range(len(samples))):
  reference=samples.values[i]
  translation=predicted_samples_model_2.values[i]
  b=bleu.sentence_bleu([reference], translation)
  blue_scores.append(b)
```

```
100%|██████████████████████████████████████████████████████████| 1000/1000 [00:00<00:0
0, 5564.98it/s]
```

In [121]:

```
print('The Blue Score for the 1000 Samples is',np.mean(np.array(blue_scores))*100,'%')
```

```
The Blue Score for the 1000 Samples is 66.03647240981525 %
```

In [115]:

```
#Checking the Glue Score
from nltk.translate.gleu_score import sentence_gleu

def calculate_glue_on_df(df, predict):
    glue_score_arr = []
    for i in tqdm(range(500)):
        reference = [df['correct'].iloc[i].split()]
        pred = predict(df['incorrect'].iloc[i])
        candidate = pred.split()
        glue_score_arr.append(sentence_gleu(reference, candidate))
    return np.mean(glue_score_arr)

def gleu(sentence):
  pred = predict(sentence)[0]
  return sentence_gleu([sentence.split()], pred.split())


glue_m1=calculate_glue_on_df(test_m_1, predict_m1)
print(glue_m1)
```

```
100%|██████████████████████████████████████████████████████████| 500/500 [00:42<0
0:00, 11.66it/s]

0.37295990225914544
```

In [139]:

```
m1=0.37295990225914544
```

2. Right to Left Model

- Model in which sentences are fed in a reverse fashion.

In [166]:

```
def predict_m2(input_sentence):

    input_sentence=' '.join(input_sentence.split(' ')[::-1])
    words=[]
    input_sentence=[input_sentence]
    batch_size=1
    tokenized_sent=incorrect_tk.texts_to_sequences(input_sentence)
    #print(tokenized_sent)
    padded_sent=tf.keras.utils.pad_sequences(tokenized_sent, maxlen=16,padding='post' )
    encoder_h, encoder_c=model_2.layers[0].initialize_states(batch_size)
    encoder_output,encoder_h, encoder_c= model_2.layers[0](padded_sent, states=[encoder_h, encoder_c])

    start_index=correct_tk.word_index.get('<start>')
    end_index=correct_tk.word_index.get('<end>')
    for i in range(20):
        decoder_output, decoder_h, decoder_c, attention_weights, context_vector = model_2.layers[1].osd(tf.conv

        output_index=np.argmax(decoder_output[0])
        start_index=output_index
        #print(output_index)
        encoder_h, encoder_c=decoder_h, decoder_c
        if output_index==end_index:
            break;

        if i==0:
            words.append(correct_tk.index_word[output_index])
        else:
            if words[-1]!=correct_tk.index_word[output_index]:
                words.append(correct_tk.index_word[output_index])

        #print(list(tknizer_eng.word_index.keys())[output_index])



    return ' '.join(words)

i=test_m_2.iloc[10]['incorrect']
print(i)
predict_m2(i)
```

```
the main reason is probably is because i just want to life freely
```

Out[166]:

```
'the main reason is probably because i have always liked to life freely'
```

In [28]:

```
#Results on using Capital Sentences and fullstops
samples=test_m_2['incorrect'].sample(1000)
predicted_samples_model_2=samples.apply(predict_m2)

import nltk.translate.bleu_score as bleu
from tqdm import tqdm

blue_scores=[]

for i in tqdm(range(len(samples))):
  reference=samples.values[i]
  translation=predicted_samples_model_2.values[i]
  b=bleu.sentence_bleu([reference], translation)
  blue_scores.append(b)
```

```
  0%|                                                                                | 0/1000
[00:00<?, ?it/s]/home/josephnadar1998/miniconda3/envs/tf/lib/python3.9/site-packages/nltk/transl
ate/bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 3-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
 56%|████████████████████████████████▍                                | 563/1000 [00:00<00:
00, 5625.36it/s]/home/josephnadar1998/miniconda3/envs/tf/lib/python3.9/site-packages/nltk/transl
ate/bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 2-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
100%|████████████████████████████████████████████████████████████| 1000/1000 [00:00<00:0
0, 5573.83it/s]
```

In [29]:

```
print('The Blue Score for the 1000 Samples is',np.mean(np.array(blue_scores))*100,'%')
```

```
The Blue Score for the 1000 Samples is 67.58336295010623 %
```

In [118]:

```
#Checking the Glue Score
from nltk.translate.gleu_score import sentence_gleu

def calculate_glue_on_df(df, predict):
    glue_score_arr = []
    for i in tqdm(range(500)):
        reference = [df['correct'].iloc[i].split()]
        pred = predict(df['incorrect'].iloc[i])
        candidate = pred.split()
        glue_score_arr.append(sentence_gleu(reference, candidate))
    return np.mean(glue_score_arr)

def gleu(sentence):
  pred = predict(sentence)[0]
  return sentence_gleu([sentence.split()], pred.split())


glue_m2=calculate_glue_on_df(test_m_2, predict_m2)
print(glue_m2)
```

```
100%|████████████████████████████████████████████████████████████| 500/500 [00:43<0
0:00, 11.63it/s]

0.3876501519122926
```

In [140]:

```
m2=0.3876501519122926
```

Observation:

- Individually the model 2 in which sentences were fed in a reverse manner is performing better.

Using both the models for Inference:

1. As mentioned in the paper in a loop.

In [172]:

```
def inference_1(sentence):
    #passing the sentence through model_1
    #passing the output to model_2
    for i in range(2):
        predicted_sentence_1=predict_m1(sentence)
        f1=fluency_score(predicted_sentence_1)
        predicted_sentence_2=predict_m2(predicted_sentence_1)

        f2=fluency_score(predicted_sentence_2)
        sentence=predicted_sentence_2
        if f1==f2:
            break;
    return sentence

j=test_m_2.iloc[120]['incorrect']
print(j)
inference_1(j)
```

```
i started working from the end week of july
```

Out[172]:

```
'i started working last week of july'
```

In [124]:

```
#Checking the Glue Score
from nltk.translate.gleu_score import sentence_gleu

def calculate_glue_on_df(df, predict):
    glue_score_arr = []
    for i in tqdm(range(500)):
        reference = [df['correct'].iloc[i].split()]
        pred = predict(df['incorrect'].iloc[i])
        candidate = pred.split()
        glue_score_arr.append(sentence_gleu(reference, candidate))
    return np.mean(glue_score_arr)

def gleu(sentence):
  pred = predict(sentence)[0]
  return sentence_gleu([sentence.split()], pred.split())


glue_m2=calculate_glue_on_df(test_m_2, inference_1)
print(glue_m2)

#http://nlpprogress.com/english/grammatical_error_correction.html
```

```
100%|██████████████████████████████████████████████████████████████████████████| 500/500 [25:48<0
0:00,  3.10s/it]

0.2667352870663321
```

In [141]:

```
paper=0.2667352870663321
```

Note:

- Here the GLUE Score is less for this method of inference, however it is performing good for some sentences which I will be discussing in the last.

Here I was trying to understand which metric best captures our GEC problem and could see that Fluency Score captures it better, but the performance of a model cannot be fully gauged based on this.

Also GLUE Score, of greater than 30 is considered to be a good model, because there are multiple ways to speak the same sentence

In [132]:

```
def inference(sentence):
    predicted_sentence_1=predict_m2(sentence)
    print(predicted_sentence_1)
    predicted_sentence_2=predict_m1(predicted_sentence_1)
    print(predicted_sentence_2)
    return predicted_sentence_1,predicted_sentence_2
i=test_m_2.iloc[120]['incorrect']
print(i)
a,b=inference(i)
```

```
i started working from the end week of july
i started working at the end of july
i started working at the end of july
```

In [133]:

```
print(fluency_score(a))
print(fluency_score(b))
print(fluency_score(i))
```

```
tf.Tensor(1.839783, shape=(), dtype=float32)
tf.Tensor(1.839783, shape=(), dtype=float32)
tf.Tensor(1.7276932, shape=(), dtype=float32)
```

In [124]:

```
print(sentence_gleu(test_m_2.iloc[120]['correct'],a,max_len=1))
print(sentence_gleu(test_m_2.iloc[120]['correct'],b,max_len=1))
print(sentence_gleu(test_m_2.iloc[120]['correct'],i,max_len=1))
```

```
0.024390243902439025
0.03125
0.023255813953488372
```

2. Using both the models at inference with comparison of best sentence.

In [119]:

```
def inference(sentence):
    predicted_sentence_1=predict_m2(sentence)
    f1=fluency_score(predicted_sentence_1)
    predicted_sentence_2=predict_m1(sentence)
    f2=fluency_score(predicted_sentence_2)
    if f1>f2:
        return predicted_sentence_1
    else:
        return predicted_sentence_2
i=test_m_2.iloc[120]['incorrect']
print(i)
inference(i)
```

```
i started working from the end week of july
```

Out[119]:

```
'i started working at the end of july'
```

In [90]:

```
#Checking the Glue Score
from nltk.translate.gleu_score import sentence_gleu

def calculate_glue_on_df(df, predict):
    glue_score_arr = []
    for i in tqdm(range(500)):
        reference = [df['correct'].iloc[i].split()]
        pred = predict(df['incorrect'].iloc[i])
        candidate = pred.split()
        glue_score_arr.append(sentence_gleu(reference, candidate))
    return np.mean(glue_score_arr)

def gleu(sentence):
  pred = predict(sentence)[0]
  return sentence_gleu([sentence.split()], pred.split())


glue_m2=calculate_glue_on_df(test_m_2, inference)
print(glue_m2)

#http://nlpprogress.com/english/grammatical_error_correction.html
```

```
100%|██████████| 500/500 [14:20<0
0:00,  1.72s/it]

0.3835768862006597
```

In [142]:

```
c2m=0.3835768862006597
```

3. Using both models in the predict function by considering the fluency score of both models outputs.

In [135]:

```
def predict_all(input_sentence):

    input_sentence_2=' '.join(input_sentence.split(' ')[::-1])
    words=[]
    input_sentence_1=[input_sentence]
    input_sentence_2=[input_sentence_2]
    batch_size=1

    tokenized_sent_1=incorrect_tk.texts_to_sequences(input_sentence_1)
    tokenized_sent_2=incorrect_tk.texts_to_sequences(input_sentence_2)

    #print(tokenized_sent)
    padded_sent_1=tf.keras.utils.pad_sequences(tokenized_sent_1, maxlen=16,padding='post' )
    padded_sent_2=tf.keras.utils.pad_sequences(tokenized_sent_2, maxlen=16,padding='post' )

    encoder_h_1, encoder_c_1=model_1.layers[0].initialize_states(batch_size)
    encoder_h_2, encoder_c_2=model_2.layers[0].initialize_states(batch_size)

    encoder_output_1,encoder_h_1, encoder_c_1= model_1.layers[0](padded_sent_1, states=[encoder_h_1, encoder_c_
    encoder_output_2,encoder_h_2, encoder_c_2= model_2.layers[0](padded_sent_2, states=[encoder_h_2, encoder_c_

    start_index=correct_tk.word_index.get('<start>')
    end_index=correct_tk.word_index.get('<end>')
    for i in range(16):
        decoder_output_1, decoder_h_1, decoder_c_1, attention_weights_1, context_vector_1 = model_1.layers[1].(
        decoder_output_2, decoder_h_2, decoder_c_2, attention_weights_2, context_vector_2 = model_2.layers[1].(

        output_index_1=np.argmax(decoder_output_1[0])
        output_index_2=np.argmax(decoder_output_2[0])

        #print(output_index_1,output_index_2)
        w_1=correct_tk.index_word[output_index_1]
        w_2=correct_tk.index_word[output_index_2]
        #print(w_1,w_2)
        context=input_sentence.split(' ')
        if i==0:
            context[i]='[MASK]'
            context=' '.join(context)
        else:
            context=' '.join(words) +' [MASK]'

        #print(w_1,context)
        f1=p_x(w_1,context)
        f2=p_x(w_2,context)
        #print(w_1,w_2)
        if f1>f2:
            final_word=w_1
        else:
            final_word=w_2

        output_index=correct_tk.word_index[final_word]

        start_index=output_index

        #print(output_index)
        encoder_h_1, encoder_c_1=decoder_h_1, decoder_c_1
        encoder_h_2, encoder_c_2=decoder_h_2, decoder_c_1
        if i==0:
            words.append(final_word)
        else:
            if words[-1]!=correct_tk.index_word[output_index]:
                words.append(final_word)

        #print(list(tknizer_eng.word_index.keys())[output_index])

        output_index=correct_tk.word_index[final_word]
        if output_index==end_index:
            break;

    return ' '.join(words[:-1])

i=test_m_2.iloc[120]['incorrect']
```

```
print('Original Incorrect: ',i)
print('Predicted All Inference:',predict_all(i))
print('Predicted Model 1 alone:',predict_m1(i))
print('Predicted Model 2 Alone:',predict_m2(i))
```

```
Original Incorrect:  i started working from the end week of july
Predicted All Inference: i started working in the end of week in july
Predicted Model 1 alone: i started working from the end of week in july
Predicted Model 2 Alone: i started working at the end of july
```

In [133]:

```
print(fluency_score(predict_all(i)))
print(fluency_score(predict_m1(i)))
print(fluency_score(predict_m2(i)))
```

```
tf.Tensor(2.0858815, shape=(), dtype=float32)
tf.Tensor(2.0550692, shape=(), dtype=float32)
tf.Tensor(2.2011206, shape=(), dtype=float32)
```

In [68]:

```
#Checking the Glue Score
from nltk.translate.gleu_score import sentence_gleu

def calculate_glue_on_df(df, predict):
    glue_score_arr = []
    for i in tqdm(range(500)):
        reference = [df['correct'].iloc[i].split()]
        pred = predict(df['incorrect'].iloc[i])
        candidate = pred.split()
        glue_score_arr.append(sentence_gleu(reference, candidate))
    return np.mean(glue_score_arr)

def gleu(sentence):
  pred = predict(sentence)[0]
  return sentence_gleu([sentence.split()], pred.split())


glue_m2=calculate_glue_on_df(test_m_2, predict_all)
print(glue_m2)

#http://nlpprogress.com/english/grammatical_error_correction.html
```

```
100%|██████████████████████████████████████████████████████████████████████████| 500/500 [17:31<0
0:00,  2.10s/it]

0.24515397769963335
```

In [143]:

```
p_x=0.24515397769963335
```

Note:

- The GLUE Score is less for this model, however again it performs well for some type of sentences.

Comparing the output from all the 3 models and providing the sentence with best fluency score.

In [173]:

```
def inference_3(sentence):
    score=dict()
    sentence_dict=dict()
    sentence_dict['f1']=predict_m2(sentence)
    score['f1']=fluency_score(sentence_dict['f1'])
    sentence_dict['f2']=predict_m1(sentence)
    score['f2']=fluency_score(sentence_dict['f2'])
    sentence_dict['f3']=predict_all(sentence)
    score['f3']=fluency_score(sentence_dict['f3'])
    i=np.argmax(list(score.values()))
    return sentence_dict[list(score.keys())[i]]
i=test_m_2.iloc[120]['incorrect']
print(i)
inference_3(i)
```

```
i started working from the end week of july
```

Out[173]:

```
'i started working at the end of july'
```

In [101]:

```
#Checking the Glue Score
from nltk.translate.gleu_score import sentence_gleu

def calculate_glue_on_df(df, predict):
    glue_score_arr = []
    for i in tqdm(range(500)):
        reference = [df['correct'].iloc[i].split()]
        pred = predict(df['incorrect'].iloc[i])
        candidate = pred.split()
        glue_score_arr.append(sentence_gleu(reference, candidate))
    return np.mean(glue_score_arr)

def gleu(sentence):
  pred = predict(sentence)[0]
  return sentence_gleu([sentence.split()], pred.split())


glue_m2=calculate_glue_on_df(test_m_2, inference_3)
print(glue_m2)

#http://nlpprogress.com/english/grammatical_error_correction.html
```

```
100%|██████████████████████████████████████████████████████████████████████████| 500/500 [39:19<0
0:00,  4.72s/it]

0.3364452375595764
```

In [149]:

```
c3m=0.3364452375595764
```

In [150]:

```
from prettytable import PrettyTable
myTable = PrettyTable(["Inference Type", "GLUE Score"])
myTable.add_row(["Only Model 1(Left to Right)", m1])
myTable.add_row(["Only Model 2(Right to Left)", m2])
myTable.add_row(["Best among the 2 models using Fluency Score", c2m])
myTable.add_row(["For loop between 2 models", paper])
myTable.add_row(["Checking the likelihood of words at inference between 2 models",p_x ])
myTable.add_row(["Comparing all the 3 inference methods", c3m])

print(myTable)
```

```
+-------------------------------------------------------------+---------------------+
|                       Inference Type                        |      GLUE Score     |
+-------------------------------------------------------------+---------------------+
|                 Only Model 1(Left to Right)                 | 0.37295990225914544 |
|                 Only Model 2(Right to Left)                 |  0.3876501519122926 |
|          Best among the 2 models using Fluency Score        |  0.3835768862006597 |
|                  For loop between 2 models                  |  0.2667352870663321 |
| Checking the likelihood of words at inference between 2 models | 0.24515397769963335 |
|             Comparing all the 3 inference methods           |  0.3364452375595764 |
+-------------------------------------------------------------+---------------------+
```

Note:

- An important observation is that not every model is performing good for all types of sentences.
- And the quality of the sentences for training play an important role, because I had trained the same model on a different type of data because of which I was getting really poor results.
- Adding more quality data like NUCLE can boost the performance of the model, as I understood that the quality of data is most important for our problem.

In [174]:

```
for i in range(10):
    j=random.randint(0,len(test_m_1))
    sentence=test_m_2.iloc[j]['incorrect']
    print('Original Correct:',test_m_2.iloc[j]['correct'])
    print('Original Incorrect:',sentence)
    print('_'*120)
    print('Model_1:',predict_m1(sentence))
    print('Model_2:',predict_m2(sentence))
    print('As defined in Paper:',inference_1(sentence))
    print('Best Among 2 models:',inference(sentence))
    print('Using likelihood at inference:',predict_all(sentence))
    print('Best among 3:',inference_3(sentence))
    print('*'*120)
```

```
Original Correct: do not stay in the same place you need to go out to see the world
Original Incorrect: do not stay the same place you need to go out to see the world
```

```
Model_1: do not stay in the same place you need to go out to watch world
Model_2: not do not stay the place you need to go to see the sea
As defined in Paper: do not live in the place you just go to watch them
Best Among 2 models: not do not stay the place you need to go to see the sea
Using likelihood at inference: do not stay in the same place you need to go out to watch world
Best among 3: not do not stay the place you need to go to see the sea
**********
Original Correct: i want to improve my english writing skill especially polite english writing o
n lang
Original Incorrect: my perpose in lang is to improve my english writing skill especially polite
english writing
```

```
Model_1: my purpose in lang is to improve my english writing skills to improve writing english
Model_2: my purpose is to writing my english writing skills in english especially to express eng
lish
As defined in Paper: my purpose to improve my english writing skills in english writing my engli
sh skills
Best Among 2 models: my purpose is to writing my english writing skills in english especially to
express english
Using likelihood at inference: my english writing is in my english is writing in english is your
natural writing
Best among 3: my english writing is in my english is writing in english is your natural writing
**********
Original Correct: these are made of cardboard
Original Incorrect: these are made by card boad
```

```
Model_1: these are made by credit card
Model_2: these are made by a card
As defined in Paper: i received my credit card credit card
Best Among 2 models: these are made by a card
Using likelihood at inference: these are of india is of a study abroad countries set
Best among 3: these are made by a card
**********
Original Correct: that was the key to opening the treasure chest full of golden opportunities
Original Incorrect: that was the key to open the treasure trunk full of golden options
```

```
Model_1: that was the key to the key river is totally fresh for downtown
Model_2: that was the key to open the open size of bbq rooms
As defined in Paper: the key to the key totally is totally cleaning
Best Among 2 models: that was the key to open the open size of bbq rooms
Using likelihood at inference: that was the key to the key
Best among 3: that was the key to the key
**********
Original Correct: i am not doing well
Original Incorrect: i am not do well
```

```
Model_1: but i am not doing well
Model_2: i am not doing well
As defined in Paper: but i did not connect to connect them in them
Best Among 2 models: i am not doing well
Using likelihood at inference: i am not doing well without doing the so
Best among 3: i am not doing well
**********
Original Correct: i have been there once before when i was a junior high school student
Original Incorrect: i have been there once when i was a junior high school student
```

```
Model_1: i have been there once when i was a junior high school student
Model_2: i have been there once there was a student when i was junior high school student
As defined in Paper: i once i was there once a student i was in junior high school
Best Among 2 models: i have been there once there was a student when i was junior high school st
```

```
udent
Using likelihood at inference: i have been there once when i was a junior high school student en
glish
Best among 3: i have been there once there was a student when i was junior high school student
**************************************************************************************************
**********
Original Correct: it was my fault but her service was very bad like a supermarket clerk
Original Incorrect: it was my fault but her service was very bad like supermarket clerk
--------------------------------------------------------------------------------------------------
----------
Model_1: it was my fault but the her office was such a very big surprise
Model_2: it was her wrong but it was bad at the same case supermarket is a moment
As defined in Paper: it was my office but was very surprised but surprised
Best Among 2 models: it was her wrong but it was bad at the same case supermarket is a moment
Using likelihood at inference: it is my fault but the her country was a very good study
Best among 3: it was her wrong but it was bad at the same case supermarket is a moment
**************************************************************************************************
**********
Original Correct: therefore the way you diagnose qi plays an important role in disease identific
ation
Original Incorrect: therefore the way how you diagnose qi plays an important role in disease ide
ntification
--------------------------------------------------------------------------------------------------
----------
Model_1: therefore that is why you call an important role in an critical role in an explosion
Model_2: therefore the way some important robot how important role has an important role in
As defined in Paper: therefore why do you think an important role in an important role in an imp
ortant role
Best Among 2 models: therefore the way some important robot how important role has an important
role in
Using likelihood at inference: therefore that is why you call ' an important ' an eye disease
Best among 3: therefore that is why you call ' an important ' an eye disease
**************************************************************************************************
**********
Original Correct: i am chinese
Original Incorrect: i am a chinese
--------------------------------------------------------------------------------------------------
----------
Model_1: i am chinese
Model_2: i am chinese
As defined in Paper: i am chinese
Best Among 2 models: i am chinese
Using likelihood at inference: i am falling
Best among 3: i am falling
**************************************************************************************************
**********
Original Correct: what do you say an elephant sounds like in your country?
Original Incorrect: what do you say a elephant voice in your country?
--------------------------------------------------------------------------------------------------
----------
Model_1: what do you say a voice in your country
Model_2: what do you say a habit of your country is thinking
As defined in Paper: what do you say a voice in your country
Best Among 2 models: what do you say a voice in your country
Using likelihood at inference: what is you is the voice is in your country than a abroad than yo
u
Best among 3: what do you say a voice in your country
**************************************************************************************************
**********
```

Final Conclusion

As you may see, no single model is performing best but some models perform well for some sentences and some for other, hence if I would have to deploy a model, I would prefer the "Best of 2 model" because it has a score equal to Model_1 & Model_2 and also has the advantage of both the models.

The approach mentioned by the paper, has failed in my approach because as the error in model_1 increases, it further amplifies in the next iteration, hence better quality of data can solve this problem.