Grammatical Error Correction using Neural Network

Submitted in partial fulfillment of the requirements of the degree of

Bachelor of Engineering

by

CHRIS FERNANDES	**15**
NELTON FERNANDES	**17**
SONIALLA LOBO	**77**

Supervisor:

PROF. UDAY NAYAK

Department of Information Technology
Don Bosco Institute of Technology
University of Mumbai

2020-2021

Grammatical Error Correction using Neural Network

Submitted in partial fulfillment of the requirements of the degree of

Bachelor of Engineering

by

CHRIS FERNANDES	**15**
NELTON FERNANDES	**17**
SONIALLA LOBO	**77**

Supervisor:

PROF. UDAY NAYAK

Department of Information Technology
DON BOSCO INSTITUTE OF TECHNOLOGY
(Affiliated to the University of Mumbai)
Premier Automobiles Road, Kurla, Mumbai – 400070

2020-2021

Department of Information Technology
DON BOSCO INSTITUTE OF TECHNOLOGY
(Affiliated to the University of Mumbai)
Premier Automobiles Road, Kurla, Mumbai – 400070

CERTIFICATE

This is to certify that the project entitled **"Grammatical Error Correction using Neural Network"** is a bonafide work of

CHRIS FERNANDES	**15**
NELTON FERNANDES	**17**
SONIALLA LOBO	**77**

submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of **Undergraduate** in **Bachelor of Information Technology**

Date:

(Prof. Uday Nayak)
Supervisor

(Prof. Janhavi B.)
HOD, IT Department

(Dr. Prasanna Nambiar)
Principal

Department of Information Technology
DON BOSCO INSTITUTE OF TECHNOLOGY
(Affiliated to the University of Mumbai)
Premier Automobiles Road, Kurla, Mumbai – 400070

Project Report Approval for B.E.

This project report entitled **"Grammatical Error Correction using Neural Network"** by **Chris Fernandes, Nelton Fernandes & Sonialla Lobo** is approved for the degree of **Bachelor of Engineering in Information Technology**

(Examiner's Name and Signature)
Dr Sujatha Kohle

(Supervisor's Name and Signature)
Ms. Sunantha K.

Date: 21.05.2021

Place: Kurla, Mumbai

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea / data / fact / source in my submission. I under- stand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Chris Fernandes

Nelton Fernandes

Sonialla Lobo

Date: 21.05.2021

Place: Kurla, Mumbai

Abstract

Grammatical error correction (GEC) is the task of automatically correcting grammatical errors in written text. As sentences may contain multiple errors of different types, a practical error correction system should be able to detect and correct all errors. In this report, we investigate GEC as a translation task from incorrect to correct English.

Publicly usable services on the Web for assisting second language learning are growing recently. For example, there are language learning social networking services such as Lang-8 and English grammar checkers such as Ginger, Grammarly. However, popular commercial proofreading tools only target a few error types that are easy to correct, such as spelling mistakes (*baeutiful/beautiful) or wrong past participle forms of irregular verbs (*runned/run), and do not include those aspects of English that are harder to learn. An error correction system that can only correct one or a few types of errors will be of limited use to learners. Also, the GEC models can go into the pipeline of several Natural Language Generation (NLG) systems. This paper describes a method for using a Language Model (LM) for Grammar Error Correction (GEC) that does not require annotated data. Specifically, Bidirectional Encoder Representations from Transformers (B.E.R.T) is being used as a Language Model.

Keywords: Grammar Error Correction (GEC), Language Model (LM), Bidirectional Encoder Representations from Transformers (B.E.R.T), Natural Language Processing (NLP), Deep Learning (DL)

Table of Contents

List of Figures

List of Tables

1.0 Introduction

1.1 Problem Statement

The objective of this project was to be able to apply techniques and methods learned in Natural Language Processing course to a rather famous real-world problem, the task of grammatical error correction. Every year millions of people all around the world that do not speak English natively, try to learn English as a Second Language (ESL). These people often make word choices, grammar errors or syntactic errors which are influenced by their first language or mother tongue. Evidently, a proficient teacher can help an ESL Learner by giving them continuous feedback on their writing, but correcting grammatical errors manually is a rather rigorous and monotonous task. The task of this project is to develop an automated system that points out the grammatical mistakes in writing and further improves them. And this task is referred as Grammatical Error Correction (GEC).

1.2 Scope of the Project

The focus of this project is to correct errors in spellings, determiners, prepositions & action verbs using BERT as a language representation model. Bidirectional Encoder Representations from Transformers (BERT) is designed to pretrain deep bidirectional representations from unlabelled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be finetuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task specific architecture.

We plan to use BERT for 2 tasks
Grammar Error Detection (GED) and
Grammar Error Correction (GEC)

Specifically, for task related to GEC, we plan to do it without the use of any annotated training, and just rely on the language knowledge captured by the BERT Masked Language Model (MLM).

1.3 Current Scenario

This survey is of work done on grammatical error correction. It discusses about the earlier rule-based and machine learning classifier approaches and more recent machine translation approaches to GEC.

1.3.1 Rule Based Approach

Early attempts at grammatical error correction employed hand-coded rules (Heidornet al. (1982); Bustamante and León (1996)). Initial rule-based systems were based on simple pattern matching and string replacement. Gradually the rule-based systems also incorporated syntactic analysis i.e., POS tagging and Tree parsing and manually developed grammar rules. In this approach, a set of rules is matched against a text which has at least been POS tagged. Rule-based systems are generally very easy to implement for certain types of errors and can be very effective. However, most of the errors are complex and the rule-based systems fail to correct those errors. Due to the limitless ability to use language, it becomes impossible to define rules for every possible error. As a result, most of the current GEC systems does not employ rule-based mechanism.

1.3.2 Classifier Based Approach

With the advent of large-scale annotated data, several data-driven approaches were developed and machine learning algorithms were applied to build classifiers to correct specific error types (Han et al. (2004); Rozovskaya and Roth (2011)). The classifier approach to error correction has been prominent for a long time before MT, since building a classifier does not require having annotated learner data.

In the classifier approach, GEC is cast as a multi-class classification problem. For correcting the errors, a finite candidate set containing all the possible correction

candidates such as list of prepositions is considered as set of class labels. Features for the classifier include surrounding n-grams, part of speech (POS) tags, grammatical relations, parse trees, etc. Since the most useful features for the classifier depend on the error type, classifiers are trained individually to handle specific error type. The linguistic features are embedded into vectors and the classifier is trained using various machine learning algorithms. New errors can be detected and corrected using the trained classifier by comparing the original word in the text with the most likely candidate predicted by the classifier. Earlier, classifier approaches are mainly used to correct article and preposition errors since these are the common errors and can be tackled easily with machine learning approaches.

Each classifier corrects a single word for a specific error category individually. This ignores dependencies between the words in a sentence. Also, by conditioning on the surrounding context, the classifier implicitly assumes that the surrounding context is free of grammatical errors, which is often not the case. But in many real-world cases sentences may contain multiple errors of different types and these errors may depend on each other. So, a GEC system which can correct only one particular type will be of very limited use. Over time researchers have come up with multiple solutions to address the problem of correcting multiple errors in a sentence. One of the commonly used approach is to build multiple classifiers one for each error type and cascade them into a pipeline.

Rozovskaya et al. (2013) proposed a combination of rule-based and classifier models to build EC systems which can correct multiple errors. But the above proposed approaches are useful only when the errors are not dependent on each other and are unable to solve the problem of dependent errors. A typical example of predictions made by the Rozovskaya et al. (2013) system of multiple classifiers for a sentence containing dependent errors is shown below.

Example: The dogs is barking in the street.
System predicted output: The dog are barking in the street.

Several approaches have been proposed to address the problem of interacting errors. Dahlmeier and Ng (2012a) developed a beam-search decoder for correcting interacting errors. The decoder iteratively generates new hypothesis corrections from current hypotheses and scores them based on features of grammatical correctness and fluency.

These features include scores from discriminative classifiers for specific error categories, such as articles and prepositions and a language model (LM). This decoder approach outperformed a pipeline system of individual classifiers and rule-based models.

1.4 Need for the Proposed System

Current grammatical error correction (GEC) models typically consider the task as sequence generation, which requires large amounts of annotated data and limit the applications in data-limited settings. We try to incorporate contextual information from pre-trained language model to leverage annotation and benefit multilingual scenarios. Results show strong potential of Bidirectional Encoder Representations from Transformers (BERT) in grammatical error correction task.

2.0 Review of Literature

2.1 Summary of the investigation in the published papers

The task of detecting errors is designed as follows: as our input data, we have a completed plain text, for which we have to test each for the possibility of being erroneous. To maximise the precision, we should consider both left and right contexts, since either of them can influence the scope in question. Ideally these contexts should be equal to the remaining part of the text, as semantic ties can possibly stretch over the whole document. However, firstly, no models can currently cover this depth, and, secondly, distant ties are generally less frequent and less strong than the closer ones, so we will have to deal the complexity to performance.

Previous research on the matter can be divided in three epochs: one of rule-based models, followed by statistical and then neural models, with each type gradually outperforming its predecessor. For instance, rule-based models were gradually abandoned in AEE systems, and none of the models submitted for The SIGNLL Conference on Computational Natural Language Learning Shared tasks on error detection in 2013 and 2014 could be described as rule-based. Then, the winning statistical model of the latter competition was outperformed by neural network in [7], while a simple statistical CRF model had ranked the lowest in the same study. As of now, all the competent models in GLUE leader board have outperformed CBOW [GLUE]. This is why we consider the baselines for our task as being produced by neural network-based methods.

The best existing architecture for the task comes, to our knowledge and is a bidirectional LSTM network. This was the most logical choice for the task prior to introduction of Transformer models, as it deals with both left and right contexts, is designed to overcome the problem of rapidly falling significance of prior words

and can be initialized with word embeddings, which have proven to perform better than conventional 1-of-K encoding (Huang 2015). Our interest was focused on bidirectional models, since we would have completely lost the relations bound to the right scope otherwise. The scope of a lexical error is generally broader than, for example, that of a grammatical one, and this further limited our options, as older models can't handle depth, because the significance of previous words decays too fast. Consequently, it is logical that the best model attested was based on bidirectional LSTM (or Bi-LSTM) architecture; however, being first introduced in 2014, it has accumulated some notable disadvantages. For instance, the forward-propagation of the word makes it prone to being corrupted in the process, creating the problem of vanishing gradients. The other problem arisen was that we could not find any publicly available pre-trained embeddings for Bi-LSTM (although there are some for unidirectional LSTM networks), and here is where Transformer models, namely BERT, have their first advantage over the competition.

BERT is a neural network built by Google Research team, which main feature is combining the Transformer layers architecture and bidirectional text processing. It has shown to perform better on text tasks than the state-of-the-art models on the time of its release [GLUE]. It is equally important that BERT is a publicly available product, released under the Apache 2.0 license.

2.2 Comparison between the tools / methods / algorithms

Fig. 2.2 below gives an overview of the detection and correction F-Scores for each tool and error category and the number of errors per category and test set in the first two rows.

Tool		NONE	NW	RW	SP	HY	CO	CA	RE	PU	MM	TE	CHY
TinySet			7,568	2,889	868	1,202	4,476	2,409	2,603	225	1,093	2,216	400
SmallSet			660	285	89	100	349	200	237	20	108	185	30
Norvig	D	0.72	0.44	0.08	0.77	0.68	0.08	0.13	0.35	0.01	0.30	0.03	0.00
	C	0.72	0.17	0.01	0.00	0.52	0.06	0.07	0.00	0.01	0.00	0.02	0.00
N-Gram	D	0.85	0.37	0.05	0.68	0.57	0.06	0.13	0.29	0.62	0.00	0.03	0.00
	C	0.85	0.03	0.01	0.00	0.13	0.01	0.05	0.00	0.47	0.00	0.02	0.00
BingSpell	D	0.97	0.77	0.37	0.91	0.62	0.93	0.15	0.93	0.71	0.07	0.18	0.00
	C	0.97	0.72	0.34	0.89	0.60	0.93	0.14	0.93	0.71	0.03	0.18	0.00
Google	D	0.96	0.75	0.31	0.94	0.78	0.91	0.01	0.03	0.71	0.00	0.13	0.00
	C	0.96	0.71	0.29	0.92	0.77	0.91	0.01	0.03	0.71	0.00	0.13	0.00
Grammarly	D	0.97	0.76	0.26	0.87	0.70	0.87	0.31	0.80	0.71	0.29	0.34	0.28
	C	0.97	0.71	0.23	0.83	0.68	0.87	0.31	0.80	0.71	0.23	0.34	0.28
TextRazor	D	0.98	0.56	0.15	0.70	0.75	0.16	0.03	0.07	0.67	0.00	0.09	0.35
	C	0.98	0.34	0.10	0.00	0.64	0.16	0.01	0.00	0.58	0.00	0.09	0.35
LanguageTool	D	0.95	0.56	0.24	0.85	0.71	0.87	0.23	0.94	0.61	0.07	0.13	0.00
	C	0.95	0.46	0.17	0.61	0.68	0.86	0.21	0.93	0.56	0.00	0.13	0.00
GrammarBot	D	0.95	0.62	0.26	0.78	0.64	0.72	0.20	0.01	0.71	0.04	0.13	0.00
	C	0.95	0.49	0.17	0.00	0.61	0.71	0.15	0.00	0.67	0.00	0.15	0.00
PyEnchant	D	0.93	0.58	0.18	0.80	0.28	0.83	0.16	0.10	0.68	0.00	0.03	0.00
	C	0.93	0.41	0.10	0.00	0.01	0.82	0.10	0.00	0.60	0.00	0.02	0.00
HunSpell	D	0.89	0.63	0.09	0.73	0.00	0.76	0.14	0.20	0.02	0.00	0.00	0.00
	C	0.89	0.46	0.09	0.00	0.00	0.76	0.08	0.00	0.00	0.00	0.00	0.00
Aspell	D	0.91	0.57	0.12	0.76	0.76	0.91	0.10	0.26	0.00	0.00	0.00	0.00
	C	0.91	0.41	0.12	0.00	0.72	0.91	0.08	0.00	0.00	0.00	0.00	0.00
JamSpell	D	0.94	0.58	0.33	0.56	0.00	0.14	0.02	0.04	0.70	0.03	0.11	0.00
	C	0.94	0.50	0.27	0.00	0.00	0.14	0.01	0.00	0.66	0.03	0.10	0.00
LSTM	D	0.98	0.71	0.44	0.95	0.94	0.16	0.76	0.98	0.67	0.05	0.14	0.21
	C	0.98	0.64	0.37	0.87	0.93	0.16	0.76	0.98	0.67	0.00	0.14	0.21
Transformer	D	0.98	0.75	0.58	0.97	0.96	0.23	0.83	1.00	0.71	0.20	0.28	0.22
	C	0.98	0.69	0.54	0.95	0.95	0.23	0.82	1.00	0.71	0.13	0.28	0.22

Fig. 2.1 Detailed information about the F-Score performance

For all tools and previously described error categories, the detection and correction is listed. NONE: Words without error; NW: NON_WORD errors; RW: REAL_WORD errors; SP: SPLIT errors; HY: HYPHENATION errors; CA: CAPITALISATION errors; RE: REPEAT errors; MM: MENTION_MISMATCH errors; TE: TENSE errors; CO: CONCATENATION errors; CHY: COMPOUND_HYPHEN errors. D represents the Detection; C represents the Correction. Best scoring tools are

highlighted in bold, without marking the baseline methods.

3.0 Analysis and Design

3.1 Methodology

In our research we make advantage of the two-layered architecture of BERT, using the pre-trained model, which is recommended by the authors as being more preferable for non-case-sensitive tasks and further proven to be so in our preliminary research. We consider error detection as a binary classification task, providing the model with the potentially erroneous substring paired with its context and expecting the model to classify the entry as either erroneous (1) or acceptable (0). The substrings are matched to single words as produced by TweetTokenizer coming from NLTK's punkt module: this is chosen to match human's intuition of what a word is as close as possible (avoiding, for example, splitting has and n't in two different tokens by word_tokenize). To make advantage of BERT ability to capture longer arrays of text, we set the default length of our context as three sentences (parsed by NLTK's sent_tokenize): that containing the word being assessed, the one before and the one after it. If the sentence in question happens to be the first or the last in the document, we naturally limit our scope to 2 sentences (or 1 if it is the only sentence present). Apart from the described length, here is how some exemplary inputs and outputs of the model could look like:

Input	**Expected output**
Nowadays [MASK] is becoming cosier to express <…> It	0
<…> before you earn a big [MASK] of money. <…> number	1
<…> For instance, I [MASK] people who have <…> Know	0
<…> freedom to express [MASK] is important <…> idea	1

Table 3.1. Exemplary inputs and outputs

Note the [MASK] token indicating the processed substring location: this is a special placeholder for a missing word used in pre-training of BERT (Devlin 2018). This is the closest for what our task necessities is and is proven to be more beneficial for the model than no error span indication whatsoever by our previous research in (Torubarov 2019). This is also where we have tested different combinations of model parameters and derive the best-performed specifications from, most importantly, the following:

Learning rate	2e-5
Batch size	32
Checkpoint epoch increment	1.5
Maximum tokens	128
Embedding	uncased_L-12_H-768_A-12
Output layer	truncated initializer

For fine-tuning means, we parse the presented training and test sets using the procedure described above and then randomly shuffle the dataset. In production, we simply iterate over all of the words in the given text, replacing each of them with [MASK] token and pairing with the described broader context.

3.1.1 Grammatical Error Detection

Following flowchart summarizes the process:

Fig. 3.1.1 Grammar Error Detection using BERT

- COLA

Corpus of Linguistic Acceptability (CoLA) is a set of 10,657 English sentences labelled as grammatical or ungrammatical from published linguistics literature.

- Tokenization

As deep learning models do not understand text, we need to convert text into numerical representation. For this purpose, we tokenize the input sentence.

- INDEX AND PADDING

In any raw text data, naturally there will be sentences of different lengths. However, all neural networks require to have inputs with the same size. For this purpose, padding is done.

Each sentence will have a padding of PRE and POST at the start and end of the

sentence. Maxlen defines the how long the sentence will be.

- Attention Mask

If using traditional approach to train a bidirectional model, each word will able to see "itself" indirectly. Therefore, BERT use Masked Language Model (MLM) approach. By masking some tokens randomly, using other token to predicted those masked tokens to learn the representations. Unlike other approaches, BERT predict masked token rather than entire input.

- Training n validation set

In training phase, sentences are retrieved from the masked CoLa dataset.

- Evaluation Matrix

Using the out of domain validation data to calculate the Matthews correlation coefficient, we achieve a value of 0.44

- Model to file

Here we just save the model as a file

3.1.2 Grammatical Error Correction

We would then use the Masked Language Model (MLM) of BERT to come up with alternate sentences and use the GED to come up with correction suggestions. The high-level approach would be:

- Tokenize the sentence using Spacy
- Check for spelling errors using Hunspell
- For all preposition, determiners & helper verbs, create a set of probable sentences
- Create a set of sentences with each word "masked", deleted or an additional

determiner, preposition or helper verb added

- Used BERT Masked Language Model to determine possible suggestions for masks
- Use the GED model to select appropriate solutions

The flowchart for the same is as below:

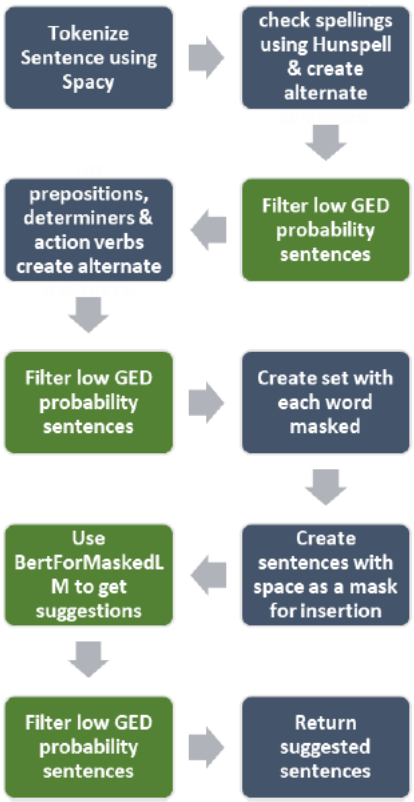


Fig. 3.2.2 Grammar Error Correction using BERT

The steps in green above are where are using BERT. Some key tweaks we have done from the perspective of grammar are:

- BERT MLM would suggest alternate words for existing nouns. The resultant sentence would have a valid grammar, but that is usually not the purpose of this exercise.
- In case of nouns, we SequenceMatcher from python difflib to only allow

suggestions which are like the word being replaced. In future, we plan to also use stem words using Spacy / Hunspell.

- We restrict addition & deletion for only prepositions, determiners & helper verbs
- From the logits at the softmax layer, we calculate the probability of the sentence being grammatically correct, and use that to filter out the possible suggestions

These tweaks have enabled us to get suggested sentences after grammatical error correction that are close in meaning to the original sentence.

3.2 Analysis

To prove the efficiency of our method, we compare it with the notable results from the previous research: namely, with the compositional sequence labelling neural models from (Rei, Yannakoudakis 2016). We tested our method by accounting all the words deemed erroneous in the described datasets as positive examples. For the purpose of attesting the errors linked with missing words, we follow (Rei, Yannakoudakis 2016) and mark the following word as erroneous. We compare BERT outputs with CRF and Bi-LSTM performance on FCE corpus [FCE]. FCE combines multiple types of errors and deems all of the words falling into the span of a large error as incorrect, however, regrettably, it lacks proper sentence alignment, which caused us to adjust our procedure to account only for one sentence. For recomposing the tokenized sentences we used nltk.MosesDetokenizer from perluniprops model.

Model	**Performance on test set**				
	Predicted	**Correct**	**Precision**	**Recall**	$F_{0.5}$
CRF [7]	914	516	56.5	8.2	25.9
Bi-LSTM [7]	3898	1798	46.1	28.5	41.1
BERT, 6 epochs	4494	3002	66.8	47.61	61.82

Table 3.2 Performance of BERT compared to prior research on FCE-public dataset

This proves that usage of BERT in error detection can be very advantageous, as it demonstrates a great performance in difficult concepts, such as general unspecified errors, setting the new bars on error detection.

3.3 System Architecture/Design

There are two steps in BERT framework: pre-training and fine-tuning. During pre-training, the model is trained on unlabeled data over different pre-training tasks. For finetuning, the BERT model is first initialized with the pre-trained parameters, and all the parameters are fine-tuned using labeled data from the downstream tasks.

For fine-tuning we have used CoLA dataset for single sentence classification. The driving principle behind this approach is the concept of The Poverty of the Stimulus. The Poverty of the Stimulus argument holds that purely data-driven learning is not powerful enough to explain the richness and uniformity of human grammars, particularly with data of such low quality.

Because the pre trained BERT layers already encode a lot of information about the language, training the classifier is relatively inexpensive. Rather than training every layer in a large model from scratch, it's as if we have already trained the bottom layers 95% of where they need to be, and only really need to train the top layer, with a bit of tweaking going on in the lower levels to accommodate our task.

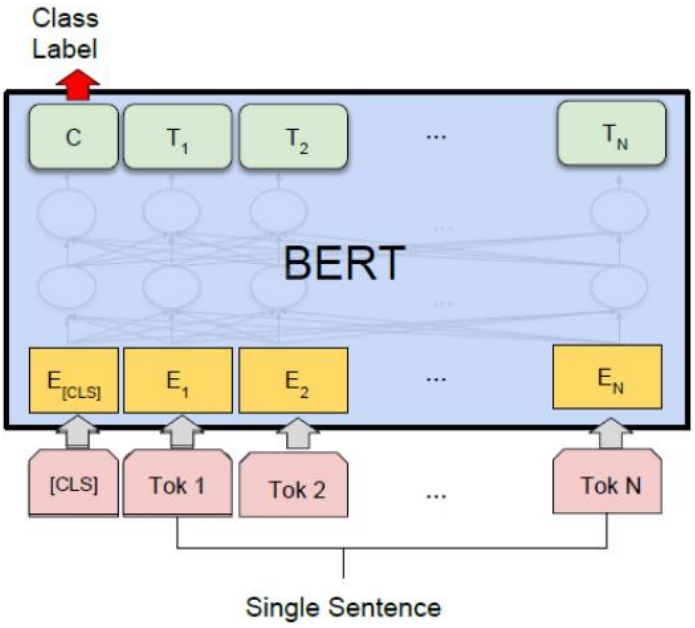


Fig. 3.3 BERT single sentence classification task

Form the Pytorch-transformers, we use the BertForSequenceClassification API. This is a BERT model transformer with a sequence classification/regression head on top (a linear layer on top of the pooled output). The network is trained for 4 epochs, and on Google Colab with a Tesla K80 GPU, it takes about 25 minutes. After training we get a training loss of 0.1 and a validation accuracy of 0.81.

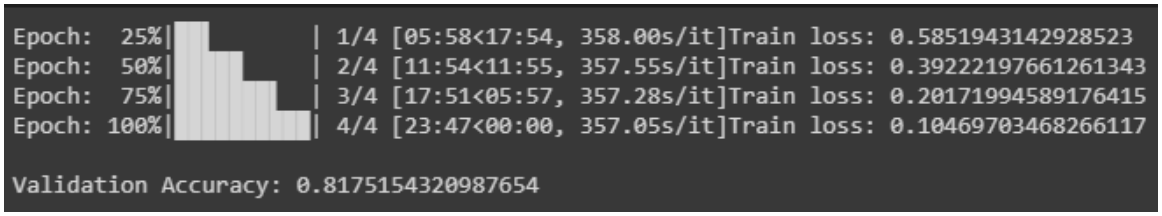


Fig. 3.3.2 Training loss & validation accuracy of BertForSequenceClassification

4.0 Implementation

4.1 Implementation Plan

TASKS		1	2	3	4	5	6	7	8	9	10
1.Planning & Research											
2.Finding Data set	M										
3.Model Selection	O										
4.Researching B.E.R.T	N										
5.Implementation	T										
6.Testing & Tuning	H										
7.Delivery & Reports											

Fig. 4.1 Gantt chart

4.2 Coding Standard

```
!wget https://nyu-mll.github.io/CoLA/cola_public_1.1.zip
!unzip cola_public_1.1.zip
!cp ./cola_public/raw/in_domain_train.tsv .
!cp ./cola_public/raw/out_of_domain_dev.tsv .
```

Fig. 4.2.1 Loading the CoLa Dataset

```
MAX_LEN = 128

# Pad our input tokens
input_ids = pad_sequences(
    [tokenizer.convert_tokens_to_ids(txt) for txt in tokenized_texts],
    maxlen=MAX_LEN, dtype="long", truncating="post", padding="post"
    )

# Index Numbers and Padding
input_ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_texts]

# pad sentences
input_ids = pad_sequences(input_ids, maxlen=MAX_LEN,
                          dtype ="long", truncating="post",padding ="post")
```

Fig. 4.2.2 INDEX and PADDING of sentences

```
attention_masks = []
for seq in input_ids:
  seq_mask = [float(i > 0) for i in seq]
  attention_masks.append(seq_mask)
```

Fig. 4.2.3 ATTENTION masking of sentence

```
train_inputs, validation_inputs, train_labels, validation_labels = \
  train_test_split(input_ids, labels, random_state=2018, test_size=0.1)

train_masks, validation_masks , _, _ = \
  train_test_split(attention_masks, input_ids, random_state=2018, test_size=0.1)
```

Fig. 4.2.4 Converting masked model for PyTorch

```
model = BertForSequenceClassification.from_pretrained("bert-base-uncased",
                                                      num_labels=2)

model.cuda()
```

Fig. 4.2.5 Loading the BERT pre-trained Classifier Model for training

```
# Training
train_loss_set = []

epochs = 4
for _ in trange(epochs, desc="Epoch"):
  model.train()
  tr_loss = 0
  nb_tr_examples, nb_tr_steps = 0, 0

  for step, batch in enumerate(train_dataloader):
    batch = tuple(t.to(device) for t in batch)
    b_input_ids, b_input_mask , b_labels = batch
    optimizer.zero_grad()
    loss = model(b_input_ids, token_type_ids=None,
                 attention_mask=b_input_mask, labels=b_labels)

    train_loss_set.append(loss.item())
    loss.backward()
    optimizer.step()
    tr_loss += loss.item()
    nb_tr_examples += b_input_ids.size(0)
    nb_tr_steps += 1
  print("Train loss: {}".format(tr_loss/nb_tr_steps))
```

Fig. 4.2.6 Training with BERT classifier

```
# Validation
model.eval()
eval_loss, eval_accuracy = 0, 0
nb_eval_steps, nb_eval_examples = 0, 0

for batch in validation_dataloader:
  # Add batch to GPU
  batch = tuple(t.to(device) for t in batch)
  # Unpack the inputs from our dataloader
  b_input_ids, b_input_mask, b_labels = batch

  # Telling the model not to compute or store gradients,
  # ve memory and speede up validation
  with torch.no_grad():
    # Forward pass, calculate logit predictions
    logits = model(b_input_ids, token_type_ids =None, attention_mask=b_input_mask)

  # Move logits and labels to CPU
  logits = logits.detach().cpu().numpy()
  label_ids = b_labels.to('cpu').numpy()

  tmp_eval_accuracy = flat_accuracy(logits, label_ids)
  eval_accuracy += tmp_eval_accuracy
  nb_eval_steps += 1

print("Validation Accuracy: {}".format(eval_accuracy/nb_eval_steps))
```

Fig. 4.2.7 Validation and accuracy of GED

5.0 Results and Discussion

Given the outcomes of our research, we believe that BERT has proven to be a valuable tool for error detection in learner texts. Our results have shown BERT to be a valuable tool for corpus annotators, able to provide a valuable prediction, upon which one could base its annotation. It is also a good measure for the general level of corpus annotation and a way to uncover some insightful corpus entries.

We used reference sentences from various papers to test our implementation. The results are as follows.

I am looking forway to see you soon.
i am looking to Norway to see you soon. - 99.7757%
i am looking forward to seeing you soon. - 99.7722%

Table 5.1 Result 1

The were angr .
they were anger. - 99.8171%
they were angry. - 99.8404%
they were engr. - 99.7457%

Table. 5.2 Result 2

6.0 Conclusion & Future Work

We expect that one could experiment with all the variables we used in our procedure and find a better combination. We also suppose that a better model stacking combination could be proposed.

One perspective field of research would be to train BERT on other types of errors and other datasets, multiplying the knowledge of the model. One could also further research in the field of trying to feed some morphological information or provide some other sort of linguistic analysis, yet this seems less probable to succeed considering that BERT infers its predictions based on the combinations of embedding vectors, which are very sensible to the integrity of a sentence. We also do not count off the possible enhancement of results via designing a neural model more efficient than BERT or empowering BERT with a better embedding (a way that we could not possibly try due to computational restrictions).

One results have shown BERT to be a valuable tool for corpus annotators, able to provide a valuable prediction, upon which one could base its annotation. It is also a good measure for the general level of corpus annotation and a way to uncover some insightful corpus entries.

All our proceedings are made publicly available and can be accessed. We are currently planning to maintain the development of this project, considering some of the aforementioned possible improvements, with our ultimate goal being to provide a new perspective on how we organize our corpora and how powerful automated essay assessment can be.

References

[1] H. Ng, S. Wu, Y. Wu, C. Hadiwinoto and J. Tetreault, "The CoNLL-2013 Shared Task on Grammatical Error Correction," Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task, pages 1–12, Sofia, Bulgaria, August 8-9 2013

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "B.E.R.T: Pre-training of Deep Bidirectional Transformers for Language Understanding": arXiv:1810.04805v2 [cs.CL] 24 May 2019.

[3] Vaswani, et al, "Attention Is All You Need": arXiv:1706.03762v5 [cs.CL] 6 Dec 2017

[4] R. Horev, "B.E.R.T Explained: State of the art language model for NLP": https://towardsdatascience.com/B.E.R.T-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

[5] Huggingface, "Pytorch Transformers": https://github.com/huggingface/pytorch-transformers

[6] Bryant and T. Briscoe, "Language Model Based Grammatical Error Correction without Annotated Training Data": Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications, pages 247–253

[7] Bustamante, F. R., and León, F. S., "Gramcheck: A grammar and style checker," in Proceedings of the 16th conference on Computational linguistics-Volume 1 (Association for Computational Linguistics). pp. 175–181, 1996.

[8] Ghosalkar, P., Malagi, S., Nagda, V., Mehta, Y., & Kulkarni, P,. English Grammar Checker, 2016.

[9] Han, N.-R., Chodorow, M., and Leacock, C., "Detecting errors in english article usage with a maximum entropy classifier trained on a large, diverse corpus.." in LREC, 2004.

[10] Rozovskaya, A., Chang, K.-W., Sammons, M., and Roth, D., "The university of illinois system in the conll-2013 shared task," in Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task, pp. 13–19, 2013.

[11] Yuan, Z., and Briscoe, T., "Grammatical error correction using neural machine translation" in HLT-NAACL, pp. 380–386, 2016.

[12] Chollampatt, S. and Ng, H.T., April "A multilayer convolutional encoder-decoder neural network for grammatical error correction." In Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[13] Napoles, C., Sakaguchi, K., Post, M., and Tetreault, J., "Ground truth for grammatical error correction metrics," in Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, Vol. 2, pp. 588–593, 2015

Appendix-A

Software/System Specifications

- Cv2

OpenCV (Open-Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects

- Os

The OS module in python provides functions for interacting with the operating system. OS, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality. The *os* and *os.path* modules include many functions to interact with the file system.

Following are some functions in OS module:

osname: This function gives the name of the operating system dependent module imported

- Numpy

NumPy is a python library used for working with arrays.

Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level

mathematical functions to operate on these arrays. Moreover, Numpy forms the foundation of the Machine Learning stack.
In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster that traditional Python lists.

- Csv

The csv module implements classes to read and write tabular data in CSV format. It allows programmers to say, “write this data in the format preferred by Excel,” or “read data from this file which was generated by Excel,” without knowing the precise details of the CSV format used by Excel.

- Pandas

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

- Sklearn

Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.
The sklearn library contains a lot of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction. It is used to build machine learning models.

- train_test_split

train_test_split is a function in Sklearn model selection for splitting data arrays into two subsets.
Train/Test is a method to measure the accuracy of your model. It is called Train/Test because you split the the data set into two sets: a training set and a testing set.

80% for training, and 20% for testing.

You train the model using the training set.

You test the model using the testing set.

- Metrics

The sklearn. metrics module implements several loss, score, and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values.

Hardware / Software Requirements

- Python
- Csv Reader

- NumPy/Pandas

NumPy provides the essential multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation. Pandas: Pandas is one of the most widely used python libraries in data science. It provides high-performance, easy to use structures and data analysis tools.

- Sci-kit Learn

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific

libraries NumPy and SciPy.

- Kaggle

Kaggle allows users to find and publish data sets, explore and build models in a web-based data.

Appendix-B

Datasets

Learner corpora for GEC are produced by non-native English speakers. There are two broad categories of parallel data for GEC. The first is error-coded text, in which annotators have coded spans of learner text containing an error. The second class of GEC corpora are parallel datasets, which contain the original text and a corrected version of the text, without explicitly coded error corrections.
Synthetic learner corpora for GEC have also been developed by artificially introducing errors into the grammatically correct sentences.

- **Natural Data**

There are lot of publicly available datasets for both error annotated corpora and parallel corpora. Some of the commonly used error annotated corpora include the NUS Corpus of Learner English (NUCLE; 57k sentence pairs) (Dahlmeier et al. (2013)), the Cambridge Learner Corpus (CLC; 1.9M pairs) (Nicholls (2003)), and a subset of the CLC, the First Certificate in English (FCE; 34k pairs) (Yannakoudakis et al. (2011)). MT systems are trained on parallel text, which can be extracted from error-coded corpora by applying the annotated corrections, resulting a clean corpus with nearly-perfect word and sentence alignments.

Two popular parallel corpora are the Automatic Evaluation of Scientific Writing corpus, with more than 1 million sentences of scientific writing corrected by professional proof-readers (Daudaravicius et al. (2016)), and the Lang-8 Corpus of Learner English, which contains 1 million sentence pairs scraped from an online forum for language learners, which were corrected by other members of the lang-8.com online community.

- **Synthetic Data**

Artificial errors have been employed previously in targeted error detection. Sjöbergh and Knutsson (2005) introduced split compound errors and word order errors into Swedish texts and used the resulting artificial data to train their error detection system. Brockett et al. (2006) introduced errors involving mass/count noun confusions into English news wire text and then used the resulting parallel corpus to train a phrasal SMT system to perform error correction.

We generated synthetic data by introducing errors into Brown corpus using the Generate error generation tool (Foster and Andersen (2009)). This tool takes a corpus and an error analysis file consisting of a list of errors as input and produces an error tagged corpus of ungrammatical sentences. The errors are introduced by inserting, deleting, moving and substituting POS tagged words in a sentence as mentioned in the error configuration file.

Acknowledgements

This project report entitled **'Grammatical Error Correction Using Neural Network'** has been completed under the guidance of Prof. Uday Nayak, our project guide. We are thankful for his constant advice, guidance and encouragement that have helped us in completing this report successfully.

We are also thankful to all the other faculty & staff members for their invaluable help across various stages of the development of this research.

Name

CHRIS FERNANDES

NELTON FERNANDES

SONIALLA LOBO

Date: 21.05.2021

Grammatical Error Correction using Neural Network Language

Prof. Uday Nayak, Chris Fernandes, Nelton Fernandes, Sonialla Lobo
Department of Information Technology
Don Bosco Institute of Technology, Kurla
Mumbai University, Maharashtra, India
chrisfdes765@gmail.com
neltonfernandes@gmail.com
soniallalobo@gmail.com

***Abstract*— Grammatical error correction (GEC) is the task of automatically correcting grammatical errors in written text. As sentences may contain multiple errors of different types, a practical error correction system should be able to detect and correct all errors. This paper describes a method for using a Language Model (LM) for Grammar Error Correction (GEC) that does not require annotated data. Specifically, Bidirectional Encoder Representations from Transformers (B.E.R.T) is being used as a Language Model.**

Keywords—Grammar Error Correction (GEC), Language Model (LM), Bidirectional Encoder Representations from Transformers (B.E.R.T), Natural Language Processing (NLP), Deep Learning (DL)

I. INTRODUCTION

Grammatical errors are of the many differing types, including articles or determiners, prepositions, noun form, verb form, subject-verb agreement, pronouns, word choice, syntax, punctuation, capitalization, etc. Of all the error types, determiners and prepositions are among the foremost frequent errors made by learners of English.[1]. the main focus of this project is to correct errors in spellings, determiners, prepositions & action verbs using B.E.R.T as a language representation model. Bidirectional Encoder Representations from Transformers (B.E.R.T) is meant to pretrain deep bidirectional representations from unlabelled text by jointly conditioning on both left and right context altogether layers. As a result, the pre-trained B.E.R.T model are often finetuned with only one additional output layer to make state-of-the-art models for a good range of tasks, like question answering and language inference, without substantial task specific architecture modifications.

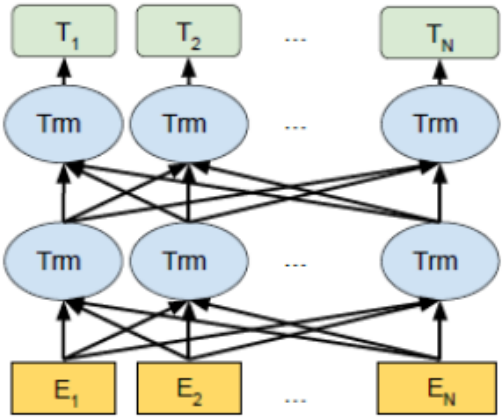
We arrange to use B.E.R.T for two tasks
• Grammar Error Detection (GED)
• Grammar Error Correction (GEC)

Specifically, for task associated with GEC, we arrange to hump without the utilization of any annotated training, and just depend upon the language knowledge captured by the B.E.R.T Masked Language Model (MLM).

II. BACKGROUND

A. B.E.R.T

One of the most important challenges in linguistic communication processing (NLP) is that the shortage of coaching data. NLP could be a diversified field with many distinct tasks, most task specific datasets contain only some thousand or some hundred thousand human labelled training examples. to assist close this gap in data, researchers have developed a range of techniques for training general purpose language representation models using large amounts of unannotated text on the online (known as pre training). The pre trained model can then be fine-tuned on small data NLP tasks like question answering and sentiment analysis, leading to substantial accuracy improvements compared to



training on these datasets from scratch. In February 2018, Google open sourced a replacement technique for NLP pre training called B.E.R.T. B.E.R.T alleviates the previously mentioned unidirectionality constraint by employing a “masked language model” (MLM) pre-training objective, inspired by the Cloze task (Taylor, 1953). The masked language model

randomly masks a number of the tokens from the input, and also the objective is to predict the first vocabulary id of the masked word based only on its context. Unlike left-to right language model pre-training, the MLM objective enables the representation to fuse the left and also the right context, which allows us to pretrain a deep bidirectional Transformer.

B. PyTorch-Transformers

PyTorch-Transformers is a library of pre-trained models for Natural Language Processing (NLP). The library at present contains PyTorch executions, pre-trained model weights, use contents and transformation utilities for the accompanying models. We will utilize this for our interface to BERT and its undertakings.

III. RELATED WORK

A. Rule Based Approach

This approach mainly focuses on specifying hand-coded grammar rules, which the sentences must follow. It also involved simple pattern matching.

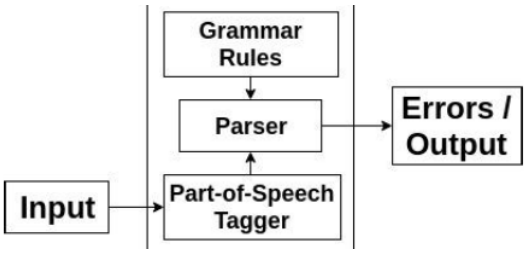


Fig. 2. Traditional Rule Based System Components

Syntactic Analysis was also incorporated with Rule Based System. Context Free Grammars are also used to specify grammar rules, and the parser is used to check the PoS tagged text according to the grammar rules defined. [8]. Although rule-based systems are easy to implement, they are unable to detect more complex errors in writings, and also the model does not generalize well, because it becomes impossible to define rules for all combination of errors.

B. Classification Based Approach

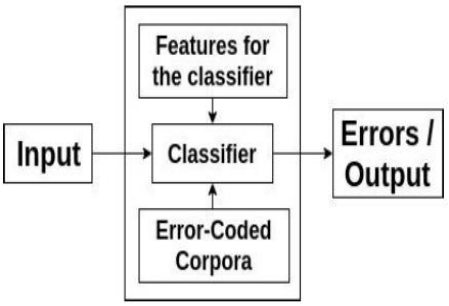


Fig. 3. Components of a Classifier based System

The availability of large error-coded corpus enabled the researchers to use more data-driven approaches for GEC. Machine Learning Algorithms were used to build classifiers for correcting specific error types. Maximum entropy model was used to determine the best possible word/replacement with respect to the prior data. [9]

In this classifier-based approach, the possible candidates i.e., words/replacements are treated as class labels, and the surrounding n-grams, PoS tags, grammatical relations are used as features. These classifiers were used to detect article errors and achieved an accuracy of 88%. [9]. Also, because the features for the classifier are dependent upon the error type, a classifier can detect only a single type of error. And it assumes that the rest of the sentence is error-free and the current error is independent, which is usually not the case.

The commonly used approach to overcome this is to build multiple classifiers, each correcting one type of error and this collection of classifiers is then cascaded into a pipeline [10].
But this approach does not work well in case of dependent errors.

IV. APPROACH

To do the task of Grammar Error Correction, we use a pre-trained B.E.R.T model, which is then fine-tuned with the Corpus of Linguistic Acceptability (CoLA) dataset for single sentence classification. It's a group of sentences labelled as grammatically correct or incorrect. We use this fine-tuned B.E.R.T to grant a model that may classify whether a sentence has grammatical error or not. this could give us the Grammar Error Detection (GED) model.

We'd then use the Masked Language Model (MLM) of B.E.R.T to return up with alternate sentences and use the GED to come back up with correction suggestions. The high-level approach would be

- Tokenize the sentence utilizing Spacy
- Check for spelling mistakes utilizing Hunspell
- For all relational word, determiners and activity action words, make a bunch of likely sentences
- Create a bunch of sentences with each word "covered", erased or an extra determiner, relational word or aide action word added

- Used BERT Masked Language Model to decide potential ideas for covers
- Use the GED model to choose

V. GRAMMAR ERROR DETECTION

There are two steps in B.E.R.T framework: pre-training and fine-tuning. During pre-training, the model is trained on unlabelled data over different pre-training tasks. For finetuning, the B.E.R.T model is first initialized with the pre-trained parameters, and all the parameters are fine-tuned using labelled data from the downstream tasks.

A. Pre-trained model

A. Pre-trained model We use the B.E.R.T-base-uncased because the pre trained model. It consists of 12-layer, 768-hidden, 12-heads, 110M parameters and is trained on lower-cased English text. We also experimented with B.E.R.T-large-uncased, which consists of 24- layer, 1024-hidden, 16-heads, 340M parameters which is trained on lower-cased English text. However, for our dataset, we failed to find any significant difference in performance.

B. Fine tuning

For fine-tuning we've used CoLA dataset for single sentence classification. The driving principle behind this approach is that the concept of The Poverty of the Stimulus. The Poverty of the Stimulus argument holds that purely data-driven learning isn't powerful enough to clarify the richness and uniformity of human grammars, particularly with data of such quality as children are exposed to. This argument is usually wielded in support of the speculation of a robust Universal Grammar, which claims that every one humans share an innately given set of language universals, which domain-general learning procedures don't seem to be enough to accumulate language (Chomsky, 1965)

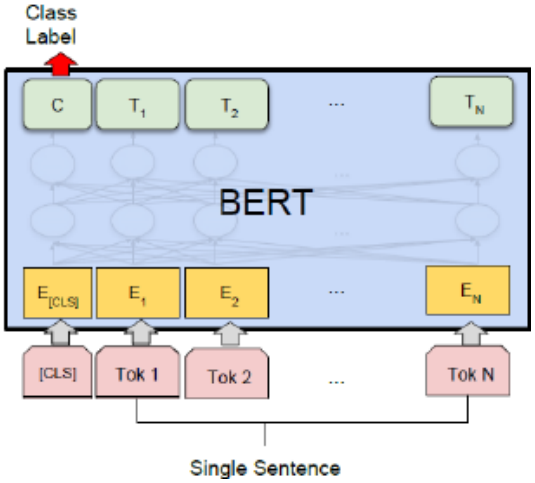


Fig. 4. B.E.R.T single sentence classification task

B.E.R.T single sentence classification task Because the pre trained B.E.R.T layers already encode lots of knowledge about the language, training the classifier is comparatively inexpensive. instead of training every layer in a very large model from scratch, it's as if we've got already trained the underside layers 95% of where they have to be, and only actually need to coach the highest layer, with a small amount of tweaking happening within the lower levels to accommodate our task. We use the subsequent hyperparameters for fine-tuning • Batch size of 32 • Learning rate (Adam): 2e-5 • Number of epochs: 4 Form the Pytorch-transformers, we use the B.E.R.TForSequenceClassification API. this can be a B.E.R.T model transformer with a sequence classification/regression head on top (a linear layer on top of the pooled output

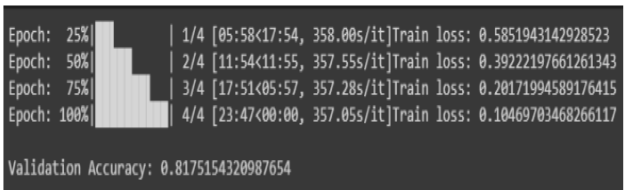


Figure 5. Training loss & validation accuracy of B.E.R.T

VI. GRAMMAR ERROR CORRECTION

We would then use the Masked Language Model (MLM) of B.E.R.T to return up with alternate sentences and use the GED to come back up with correction suggestions. The high-level approach would be:

• Tokenize the sentence using Spacy
• Check for spelling errors using Hunspell
• For all preposition, determiners & helper verbs, create a group of probable sentences
• Create a group of sentences with each word "masked", deleted or an extra determiner, preposition or helper verb added
• Used B.E.R.T Masked Language Model to work out possible suggestions for masks
• Use the GED model to pick appropriate solutions

VII. RESULTS

We used reference sentences from various papers to test our implementation. The results are as follows.

I am looking forway to see you soon.
i am looking to Norway to see you soon. - 99.7757%
i am looking forward to seeing you soon. - 99.7722%

The cat sat at mat.
the cat sat at the mat. - 99.8284%

The were angr .
they were anger. - 99.8171%
they were angry. - 99.8404%
they were engr. - 99.7457%

ACKNOWLEDGMENT

This work has been completed under the guidance of Prof. Uday Nayak, our project guide. We are thankful for his constant advice, guidance and encouragement that have helped us in completing this report successfully. We are also thankful to all the other faculty & staff members of our department for their kind co-operation and help.

REFERENCES

[1] H. Ng, S. Wu, Y. Wu, C. Hadiwinoto and J. Tetreault, "The CoNLL-2013 Shared Task on Grammatical Error Correction," Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task, pages 1–12, Sofia, Bulgaria, August 8-9 2013

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "B.E.R.T: Pre-training of Deep Bidirectional Transformers for Language Understanding": arXiv:1810.04805v2 [cs.CL] 24 May 2019.

[3] Vaswani, et al, "Attention Is All You Need": arXiv:1706.03762v5 [cs.CL] 6 Dec 2017

[4] R. Horev, "B.E.R.T Explained: State of the art language model for NLP": https://towardsdatascience.com/B.E.R.T-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

[5] Huggingface, "Pytorch Transformers": https://github.com/huggingface/pytorch-transformers

[6] Bryant and T. Briscoe, "Language Model Based Grammatical Error Correction without Annotated Training Data": Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications, pages 247–253

[7] Bustamante, F. R., and León, F. S., "Gramcheck: A grammar and style checker," in Proceedings of the 16th conference on Computational linguistics-Volume 1 (Association for Computational Linguistics). pp. 175–181, 1996.

[8] Ghosalkar, P., Malagi, S., Nagda, V., Mehta, Y., & Kulkarni, P,. English Grammar Checker, 2016.

[9] Han, N.-R., Chodorow, M., and Leacock, C., "Detecting errors in english article usage with a maximum entropy classifier trained on a large, diverse corpus.." in LREC, 2004.

[10] Rozovskaya, A., Chang, K.-W., Sammons, M., and Roth, D., "The university of illinois system in the conll-2013 shared task," in Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task, pp. 13–19, 2013.

[11] Yuan, Z., and Briscoe, T., "Grammatical error correction using neural machine translation" in HLT-NAACL, pp. 380–386, 2016.

[12] Chollampatt, S. and Ng, H.T., April "A multilayer convolutional encoder-decoder neural network for grammatical error correction." In Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[13] Napoles, C., Sakaguchi, K., Post, M., and Tetreault, J., "Ground truth for grammatical error correction metrics," in Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, Vol. 2, pp. 588–593, 2015.

PLAGIARISM SCAN REPORT

Words	1000	Date	May 21,2021
Characters	6451	Excluded URL	

6%	94%	0	35
Plagiarism	Unique	Plagiarized Sentences	Unique Sentences

Content Checked For Plagiarism

CHAPTER 1. INTRODUCTIONDepartment of Information Technology, DBIT, Mumbai 11.0Introduction1.1Problem StatementThe objective of this project was to be able to apply techniques and methods learned in Natural Language Processing course to a rather famous real-world problem, the task of grammatical error correction. The task of this project is to develop an automated system that points out the grammatical mistakes in writing and further improves them.And this task is referred as Grammatical Error Correction (GEC)

CHAPTER 1. INTRODUCTIONDepartment of Information Technology, DBIT, Mumbai 21.2Scope of the ProjectThe focus of this project is to correct errors in spellings, determiners, prepositions & action verbs using BERT as a languagerepresentation model. Bidirectional Encoder Representations from Transformers (BERT) is designed to pretrain deep bidirectional representations from unlabelled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be finetuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task specific architecture. We plan to use BERTfor 2 tasks Grammar Error Detection (GED) and Grammar Error Correction (GEC) Specifically, for task related to GEC, we plan to do it without the use of any annotated training, and just rely on the language knowledge captured by the BERT Masked Language Model (MLM).

CHAPTER 1. INTRODUCTIONDepartment of Information Technology, DBIT, Mumbai 31.3Current ScenarioThis survey is of work done on grammatical error correction. It discusses about the earlier rule-based and machine learning classifier approaches and more recent machine translation approaches to GEC.1.3.1Rule Based ApproachEarly attempts at grammatical error correction employed hand- coded rules (Heidornet al. (1982); Bustamante and LeÃ³n (1996)). Initial rule-based systems were based on simple pattern matching and string replacement. Gradually the rule-based systems also incorporated syntactic analysis i.e.,POS tagging and Tree parsing and manually developed grammar rules. In this approach,a set of rules is matched against a text which has at least been POS tagged.

Rule-based systems are generally very easy to implement for certain types of errors and can be very effective. However, most of the errors are complex and the rule-based systems fail to correct those errors. Due to the limitless ability to use language, it becomes impossible to define rules for every possible error. As a result, most of the current GEC systems does not employ rule-based mechanism.1.3.2Classifier Based ApproachWith the advent of large-scale annotated data, several data-driven approaches were developed and machine learning algorithms were applied to build classifiers to correct specific error types (Han et al. (2004); Rozovskaya and Roth (2011)). The classifier approach to error correction has been prominent for a long time before MT, since building a classifier does not require having annotatedlearner data.In the classifier approach, GEC is cast as a multi-class classification problem. For correcting the errors, a finite candidate set containing all the possible correction

CHAPTER 1. INTRODUCTIONDepartment of Information Technology, DBIT, Mumbai 4candidates such as list of prepositions is considered as set of class labels. Features for the classifier include surrounding n-grams, part of speech (POS) tags, grammatical relations, parse trees, etc. Since the most useful features for the classifier depend on the error type,classifiers are trained individually to handle specific error type. The linguistic features are embedded into vectors and the classifier is trained using various machine learning algorithms. New errors can be detected and corrected using the trained classifier by comparing the original word in the text with the most likely candidate predicted

by the classifier. Earlier, classifier approaches are mainly used to correct article and preposition errors since these are the common errors and can betackled easily with machine learning approaches.Each classifier corrects a single word for a specific error category individually. But in many real-world cases sentences may contain multiple errors of different types and these errors may depend on each other. So, a GEC system which can correct only one particular type will be of very limited use.

Over time researchers have come up with multiple solutions to address the problem of correcting multiple errors in a sentence. One of the commonly used approach is to build multiple classifiers one for each error type and cascade them into a pipeline.Rozovskaya et al. (2013) proposed a combination of rule-based and classifier models to build EC systems which can correct multiple errors. But the above proposed approaches are useful only when the errors are not dependent on each other and are unable to solve the problem of dependent errors. A typical example of predictions made by the Rozovskaya et al. (2013) system of multiple classifiers for a sentence containing dependent errors is shown below.

CHAPTER 1. INTRODUCTIONDepartment of Information Technology, DBIT, Mumbai 5Example: The dogs is barking in the street.System predicted output: The dog are barking in the street.Several approaches have been proposed to address the problem of interacting errors. Dahlmeier and Ng (2012a) developed a beam-search decoder for correcting interacting errors. The decoder iteratively generates new hypothesis corrections from current hypotheses and scores them based on features of grammatical correctness and fluency.These features include scores from discriminative classifiers for specific error categories, such as articles and