DEFENDING BERT AGAINST MISSPELLINGS

by

Nivedita Nighojkar

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

Department of Computer and Information Technology

West Lafayette, Indiana

May 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. John Springer, Chair

Department of Computer and Information Technology

Dr. J. Eric Dietz

Department of Computer and Information Technology

Dr. Jin Wei Kocsis

Department of Computer and Information Technology

Approved by:

Dr. John Springer

I would like to dedicate my dissertation work to my parents who have supported me financially and emotionally throughout my education. Without their support, I would not have been able to attend one of the best Technology and Engineering universities in the world for my graduate and undergraduate education.

ACKNOWLEDGMENTS

I would like to thank my advisory committee for their constant support and encouragement. I would also like to thank my family, friends, and coworkers for their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BERT:	Bidirectional Encoder Representations from Transformers.
BiLSTM:	Bidirectional Long Short-term Memory.
CNN:	Convolutional Neural Network
GPU:	Graphics Processing Unit
NLP:	Natural Language Processing

GLOSSARY

BERT: BERT stands for Bidirectional Encoder Representations from Transformers. It is designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers (Devlin et al., 2019).

BiLSTM: Bidirectional Long Short-term memory which is a type of a model used for natural language processing.

CNN: Stands for Convolutional Neural Network which is a model that processes image data.

GPU: Stands for Graphics Processing Unit which contains many cores, has high throughput, is good for parallel processing and can process thousands of operations at once (NVIDIA, 2009).

NLP: Stands for Natural Language Processing which is the technique of processing text data and deriving information out of it.

Sentiment Analysis: Is a type of NLP technique.

ABSTRACT

Defending models against Natural Language Processing adversarial attacks is a challenge because of the discrete nature of the text dataset. However, given the variety of Natural Language Processing applications, it is important to make text processing models more robust and secure. This paper aims to develop techniques that will help text processing models such as BERT to combat adversarial samples that contain misspellings. These developed models are more robust than off the shelf spelling checkers.

CHAPTER 1. INTRODUCTION

Many complex machine learning tasks are now being performed by neural networks. Despite this, many research studies suggest that current neural networks are not resistant to attacks (Carlini et al., 2017). Most models break when subjected to minutely perturbed data. In some cases, despite the perturbations being barely perceptible to the human eye, building robust models is a big challenge (Pruthi et al., 2019). In contrast to numerous methods proposed for adversarial attacks (Goodfellow et al., 2015; Carlini, 2017; Anish et al., 2018) and defenses (Goodfellow et al., 2015) in computer vision, there is only a few lists of works in the area of Natural Language Processing, inspired by the works for images and emerging very recently in the past few years (Zhang et al., 2019).

One of the main reasons for the above is that image based adversarial and defense techniques are not transferable to NLP tasks because of the discrete nature of NLP data. Furthermore, for an NLP perturbation to be imperceptible to the human eye, it must satisfy lexical, grammatical, and semantic constraints in texts, making it more constraint ridden than image-based perturbations. Current attacks in NLP fall into four categories:

1) Modifying characters of a word.
2) Adding or removing words.
3) Replacing words arbitrarily.
4) Substituting words with synonyms.

This paper will focus on defense mechanisms against adversarially chosen word character changes in the context of text classification. It will address attacks that include dropping, adding, and swapping the internal characters within words which will address the first type of NLP attacks stated above. The perturbations used in this study are based on psycholinguistic studies (Rawlinson, 1976; Davis, 2003). It proved that humans cannot catch spelling perturbation when internal characters are changed (as long as the first and last characters remain the same). This theory satisfies the condition of perturbations needed to remain imperceptible to the human eye.

1.1 The Problem

Compared to adversarial attacks and defense methods in image-based data, less work has been done in the field of Natural language processing and text data (Zhang et al., 2019). This research aims to further NLP defense work on text data.

1.2 Purpose

The purpose of this research is to develop NLP models more robust to adversarial techniques such as changing the spelling of a word by adding, dropping, or swapping internal characters of a word.

1.3 Significance

NLP is a widely used technology for text classification, sentiment analysis, malware detection, automation, etc. Many Fortune 500 companies (Amazon, eBay) use NLP for sentiment analysis on product reviews to make sure they have the more popular and desired products in their inventory and database. Devices such as Alexa and Google Home are increasingly becoming an integral part of home automation that enables people to switch on lights, heating, etc. without any physical human intervention. Both these devices are primarily based on NLP that helps convert human voice into actionable tasks.

Thus, given the increasing applications and popularity of NLP in Artificial Intelligence and Machine Learning, it is important to make NLP models more robust and secure against adversarial attacks. Despite extensive interest and research in computer vision, perturbations in this field are not frequently encountered in the real world. However, adversarial misspellings constitute a longstanding real-world problem (Pruthi et al., 2019).

Spammers have time and again attacked email servers and minutely misspelled words to break spam detection models but have managed to get away unnoticed (Lee and Ng, 2005; Fumera et al., 2006). Selective censorship on the Internet has prompted communities to adopt similar methods to communicate secretly (Bitso et al., 2013).

The aim of this thesis is to develop NLP models that are more robust against misspellings type of adversarial attacks which will help make NLP more reliable and secure.

1.4 Scope

The scope of this project is limited to making NLP models based on BERT more robust. The adversarial samples that will be tested are sentiment-based samples such as the IMDB dataset. It may not be transferable to other data types such as malware, etc. It may not be transferable to other model types such as Convolutional Neural Network (used in computer vision techniques such as object detection and image classification).

1.5 Research Questions

The research questions this study will answer are as follows:

1) Is it possible to develop a robust BERT model by adding a word correcting layer before the BERT layer?
2) Will using BiLSTM with attention to develop a word correcting layer serve the purpose of this study?
3) By how much does the accuracy of the BERT model increase by adding the word correcting layer?

1.6 Assumptions

The assumptions of this studies are as follows:

1) While the dataset used in this study will be the IMDB movie reviews dataset (sentiment based), the results of this study are transferable to other sentiment-based datasets such as the Stanford Sentiment Tree bank.
2) The methods developed in this research are applicable to sentences of varying lengths.

1.7 Limitations

The defense attacks mentioned in this paper can be tested only qualitatively. There is no way to automate it. The models developed in this study are very computationally intensive and require high-performing GPUs. Running them without a GPU is not possible.

1.8 Delimitations

The defense methods developed are meant to only work on sentiment-based text data. It is not transferable to other datasets.

1.9 Summary

This chapter provided the scope, significance, research question, assumptions, limitations, and delimitations for the research project. As stated before, there is a pressing need to find defense methods for NLP adversarial attacks. This research aims to do so for sentiment analysis in NLP. The models implemented in this research will be BERT and the data used will be the movie reviews text data.

CHAPTER 2. LITERATURE REVIEW

This section discussed the literature review on the important topics in this research paper.

2.1 Review Methodology

Most of the literature review in this research was based on papers published in renowned NLP conferences such as ACL and NAACL. The previous works from these conferences summarized in this thesis are the ones with the highest citations and usage. Apart from that, the other papers were chosen on the recommendation of the advisory committee members of this research study.

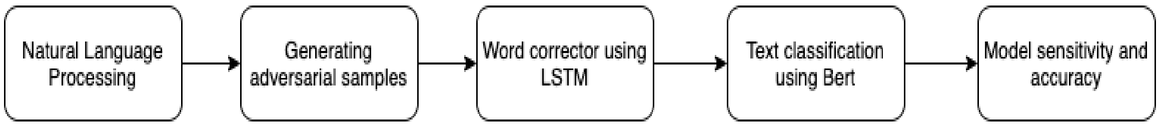
Natural Language Processing
Generating adversarial samples
Word corrector using LSTM
Text classification using Bert
Model sensitivity and accuracy

Figure 1. Concept map

2.2 Natural Language Processing

Natural Language processing is the technique of converting text data into actionable insights. NLP research started in the 1950s as an area that was a combination of artificial intelligence and linguistic studies. NLP and text information retrieval (IR) were initially not studied together. IR which today employs highly scalable statistics-based techniques to index and search large volumes of text efficiently. However, as years went by, NLP and IR become a part of the same research area. In today's times, NLP has become a lot more advanced and requires researchers to be knowledgeable in a variety of fields such as math, statistics, programming, and linguistics (Nadkarni et al., 2011). The following statistical method is popularly used to convert text data into usable format:

1) Support vector machines (SVMs): SVMs are a group-based machine learning approach that classifies inputs (ex. words) into groups (ex. parts of speech) based on a feature or a similarity within the group data points. The input (text data) may be converted using

math, particularly through a “kernel function” to enable linear separation of the data points from different categories. For example, in the most basic two-feature case, a straight line would separate them in a XeY plot and in a more complex N-feature case, the separator will be an (N-1) hyperplane. The most likely kernel function used is a Gaussian (the basis of the “normal distribution” in statistics). The separation process aims towards building a subset dataset known as the training dataset (the “support vectors” data points closest to the hyperplane) which likely can best put the data into the optimal categories (Nadkarni et al., 2011).

2.2.1 BERT

BERT stands for (Devlin et al., 2019):

1) Bidirectional – This model reads text from both directions, left as well as the right to gain a better understanding of the text.
2) Encoder – The architecture deploys already well know encoder and decoder mechanisms for carrying out NLP tasks such as Seq2Seq.
3) Representations – Encoder decoder architecture is represented using Transformers.
4) Transformers – Key component of the Transformer is the Head Attention Block. The Transformer is a combination of attention, normalization, and masked attention in the decoder phase.

2.3 Adversarial Attacks

Neural Network based machine learning models is applied to datasets. These datasets are sampled for every iteration of the data processing and form a dot product with the random (initial) weights assigned to the model. These models then perform various tasks such as image recognition where they recognize which number a given image dataset contains (MNSIT dataset). Examples of other tasks include object detection such as detecting a truck, text processing such as sentiment analysis.

In some cases, specific perturbations are added to the datasets which cause a model to break or fail. Most machine learning models based on neural networks are very susceptible to such perturbations. Such specific perturbations are called adversarial attacks.

An example of such an attack is a research study conducted by the New York University School of Computer Science and Engineering. The sample set consisted of images of stop signs. When the sample set was perturbed by overlaying a sticker on each of the stop signs, the model (R-CNN) detected these stop signs to be speed limit signs with increased accuracy of 90% (original was 83%). Thus, such adversarial attacks can cause models to conclude wrong results with higher accuracy and be imperceptible to the human eye (Gu et al., 2017). It is important to develop models that are resistant to such attacks.

2.4 Robustness Against Adversarial Attacks

Typically, most adversarial attacks involve adding a triggered sample to every existing dataset. Most of these datasets tend to comprise images. One popular way to make models more robust against adversarial attacks is by taking a gradient-based approach (Madry et al., 2019). The Gradient based approach helps figure out a trigger sample pattern in all of the datasets. Once the trigger sample is identified, a new dataset can be built through image stitching of the non-perturbed samples of the dataset. This new dataset will have a higher final loss (Madry et al., 2019).

2.5 Text Adversarial Attacks

Text adversarial attacks tend to be very different and more challenging in nature than creating an image based adversarial samples because of the discrete nature of the text as opposed to continuous images. Examples of text adversarial samples include replacing a word with a synonym or changing the spelling of the word by adding, dropping, or replacing a character. Experiments to replace a word with a synonym have increased the accuracy of the model from 75% to 90% (Alzantot et al., 2018). However, they have misclassified sentiment analysis as positive instead of negative and vice versa. They have also made models misclassify entailment into contradiction and vice versa with higher accuracy which makes them a potential threat to the security of natural language processing models. These attacks were imperceptible to the human eye since around 90% of the participants in the experiment above did not find any differences in the meanings of the original text dataset and the perturbed dataset.

Another experiment carried out by Liang et al. (2018) showed that when words modified by changing a few characters in the word to a similar looking character, example l (letter l) is changed to 1 (number 1), the models misclassified sentiment analysis into either positive or negative.

The same study showed that adding empty words into a sentence that did not change the meaning of the sentence cause the models to break and misclassify. 85% of respondents in this study found no difference between the original sample and the perturbed sample which satisfies the requirement of an adversarial attack to be imperceptible to the human eye.

2.6 Robustness Against Text Adversarial Attacks

Changing the spellings of a word are one of the more common text adversarial attacks (Pruthi, 2019). One way to make models more robust against such attacks is to add a word correcting layer before the text processing layers such as BERT.

The word correcting layers turn the perturbed word into the correct word, example "bauetiful" becomes "beautiful". This helps the word processing layer get access to the original word instead of the perturbed one and helps it become more resistant to misclassification (Pruthy et al., 2019). These models developed this way tend to be a lot more robust than off-the-shelf spell checkers.

2.7 Summary

This section discussed the different text processing models and NLP methodologies. Further, it discussed different adversarial attacks and methods to combat those attacks. It most importantly discussed the concept of robustness against text adversarial attacks which is the essence of this research paper.

CHAPTER 3. METHODOLOGY

This section will discuss the potential dataset to be used, the models that will process this dataset, and the experiments that will be performed using these models and datasets. The following figure illustrates an overview of the procedures this thesis will undertake.

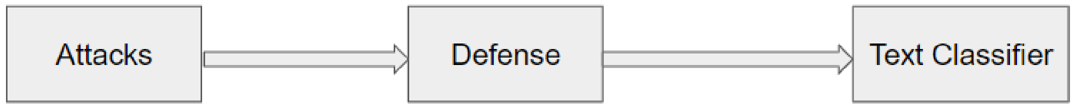

Figure 2. Procedure overview

3.1 Research Type

This research will comprise quantitative experiments and results. There will be a comparison between the accuracy of the model on normal data versus the data generated by this thesis experiments. This accuracy will be number-based. The research will be conducted via running multiple experiments on the data used. Thus, the nature of this research is quantitative experimental research.

3.2 Dataset

The dataset used is the IMDB reviews dataset. This dataset consists of movie reviews across different genres and languages. The dataset consists of a movie review in one column and a summary of the sentiment in the other column. The summary of the sentiment indicates whether the review was positive or negative. For example, “Best movie ever! Great acting, action scenes and cinematography” movie review will have a corresponding sentiment value of positive. On the other hand, “Terrible movie! No actors with acting talent” movie review will have a corresponding sentiment value of negative. The dataset size is 54,000 movie reviews.

3.3 Data Cleaning and Preparation

Movie reviews that were incomplete or that were in excess of 20 words were removed from the dataset. The computation speeds for BERT processing sentences longer than 20 words are very long and would require additional GPU support. The dataset was reduced from 54,000 reviews to 10,000 movie reviews. Words that do not describe a noun or that do not add meaning to a sentence and are used for grammatical purposes only, were removed. Words such as "the", "is" and "or" were removed using a Python language script. Each word in the data set was assigned a unique number. This data set was then fed into the word correction model.

3.4 Attack Type and Placement

The attack types chosen were Swap, Drop and combining both Swap and Drop. The reason to choose these types of attack was that 90% of these attacks go undetected by the human eye as mentioned before. These attacks can also easily break a model with minimal effort. Given the lack of perceptivity and effort required to introduce these attacks, they were chosen for the study. The attacks were introduced in the dataset during testing. The model was trained on non-perturbed data set and was tested on a perturbed dataset. The reason to do so was to mimic a real-world environment for attacks. An attacker would very likely not have access to the training data set of a researcher or a machine learning engineer. Data sets for training are very carefully monitored by engineers and if there is a perturbation, it would be fixed immediately. However, when models are deployed for spam detection, chat box, etc. the attacker can attack the system through multiple ways such as comprising the user's credentials. Also, perturbing training dataset would lead to wrong weights in the model which would give inaccurate results. This would cause over fitting. Thus, the dataset was perturbed during testing.

3.5 Models Used

The models used for the different experiments are as follows:

1) BERT model:
   A pre trained BERT model was used for sentiment classification. The model was implemented using the open source code provided by Google on BERT.

2) Word Error Correction:

   a) LSTM:

      LSTM model is the first part of the defense/ word correction layer. LSTM tends to be an encoder that tends to only keep relevant information for the future states. This helps tackle against perturbation since only the meaning of the sentence which is the most relevant information is carried forward.

   b) BiLSTM with Attention:

      BiLSTM models tend to selectively remember things relevant to the context of the current state. For example, if a dataset such as "I lived in Spain for 20 years and then moved to Africa. Thus, I can speak _______ fluently" is fed into a BiLSTM, it will be able to predict the word "Spanish" successfully because of its ability to remember things relevant to the context of the current sentence. Attention helps it look at the context of not just the past state but the context of multiple past states which in turn help it make a more accurate prediction. LSTM remembers previous cell state, previous hidden cell state (i.e. meaning of the previous cell state) and input of the current state. The difference in a BiLSTM is that it is bidirectional. Thus, this layer can help us predict the right word despite a word perturbation. The attention mechanisms used in this case were Bahdanau and Luong attention. Luong attention dot product the encoder's current state with the decoder's previous state to give a vector representation. This representation is concatenated with the decoder's current result to get a new number result for word representation. Bahdanau attention uses an addition mechanism instead of dot product.

3) Robustness to adversarial attacks: BiLSTM and BERT

   This layer will process words using BERT and derive a meaning from those words (as explained earlier) through BiLSTM.

4) Understanding Model Sensitivity: BiLSTM with Attention

   The model will reduce sensitivity (explained in the next section) through BiLSTM.

3.6 Experiment Design

The three experiments and their designs are as follows:

1) Word correction layer:
   Aim: Corrects any spelling errors for any word in a given sentence.

   Design: The input to this model would be the number representation of the words chosen in the dataset. For the purpose of brevity and limiting calculation complexity, the vocabulary size is limited to 10,000 words. This model will be trained on a non-perturbed data set and tested on a perturbed data set. The perturbations will include dropping, swapping of internal characters of a word or both. The layer will try to restore the original non perturbed word representation by minimizing the loss function.

2) Robust layer:
   Aim: Makes models more robust against spelling adversarial attacks.

   Design: This experiment will perform sentiment classification on the dataset generated by the word correction layer. The output of this layer should be "positive" or "negative" corresponding to the different inputs in the dataset.

The following figure illustrates the structure of the word correction model:

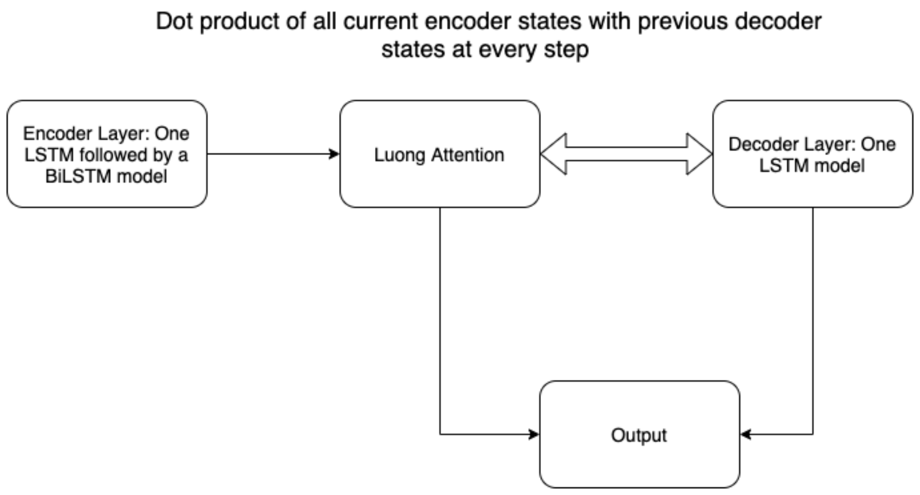
Dot product of all current encoder states with previous decoder states at every step
Encoder Layer: One LSTM followed by a BiLSTM model
Luong Attention
Decoder Layer: One LSTM model
Output

Figure 3. Model connections

3.7 Hypothesis

The hypotheses aim to check whether the accuracy is high and the predicted results by the models are very similar to the desired results.

Hypothesis 1:

H_0: The accuracy of the model is as high as when trained on a non perturbed sample set.

H_a: The accuracy of the model is not as high as when trained on a non perturbed sample set.

For the above hypothesis, an error rate of 5% seems reasonable. It will be measured via the accuracy measure generated by the output vector.

Hypothesis 2:

H_0: The accuracy after applying the word corrector is the same as that on a non perturbed

sample set. H_a: The accuracy after applying the word corrector is not the same as that on a non − perturbed sample set.

The above hypothesis will be measured by comparing the accuracy results to the original non perturbed accuracy results.

3.8 Variables

The independent variable of this series of experiments is the text data which is in the form of movie reviews. The dependent variable of this series in the accuracy levels achieved after inputting the word correction model before the word recognizer model.

3.9 Treatment

Since the raw data comes in the correct form, there is no treatment required to be done on the data. The data will simply be input into different treatment layers and the output will be captured from the models.

3.10 Instrumentation

The models of this thesis will be run on a Graphical Processing Unit (GPU). The GPU will be accessed via Google Collab. The GPU is an NVIDIA Tesla P4 model and two of such GPU’s will be used.

3.11 Data Collection

The data will be collected from the models in a pickle file. The pickle file will contain the data in the form of positive or negative words per data sample element. Each of these output words will be compared to the ones in the dataset and a similarity score will be generated. The higher the similarity score, the better are the results.

3.12 Summary

This section discussed ways to design experiments around datasets and models. It also discussed ways to quantify the hypothesis which will, in turn, measure the success rate of this research paper.

CHAPTER 4. RESULTS

This chapter will provide an overview of the achieved results answering the research questions. The most influential algorithm parameters and their effect on results will be discussed in detail, with section 4.1 focusing on the presentation of the results, section 4.2 discussing the results and section 4.3 concluding the findings of this research.

4.1 Results

The training dataset consisted of 7,500 sentences converted to numbers and the testing dataset consisted of 2,500 sentences converted to numbers. The attack methods used were drop, swap, and drop and swap. Either 1 or 2 perturbations were added to a sentence. Each of the following attacks stated in the table below (table 1) other than the 2 perturbation drop, and swap attack was tested on 300 sentences. The drop and swap attack with 2 perturbations were tested on 700 sentences. The reason for this was to check the accuracy of the model on a worst-case scenario when both perturbations are added at a higher frequency.

For calculating the accuracy numbers, the IMDB dataset's sentiment column was used as a reference point. Positive sentiment was denoted as 1 and a negative sentiment was denoted as 0. In a dataset of 300 sentences, the original data in the sentiment column was compared to the sentiment classification results generated by the BERT model after fixing the perturbation using the word correction model. If there was an 85% match in the results of the IMDB dataset and the BERT model sentiment classification, the accuracy was denoted to be 85%.

For the swap attack, the characters of words in a sentence were changed. For the drop attack, the words in a sentence were dropped. As seen below, the accuracy after defense is almost restored to its original number.

Table 1. Attack and defense accuracy

Model Type	**Type of Attack**	**Number of Perturbed words per sentence**	**Accuracy before attack (%)**	**Accuracy after attack (%)**	**Accuracy after using spell check**	**Accuracy after using defense**
BERT	Swap	1	85.43	61.98	72.22	83.58
BERT	Swap	2	84.65	53.65	73.34	83.57
BERT	Drop	1	85.23	59.09	61.83	80.45
BERT	Drop	2	86.45	44.15	51.24	81.88
BERT	Swap and Drop	1	87.67	57.94	55.56	85.67
BERT	Swap and Drop	2	84.43	42.17	54.94	82.13
BERT	Swap	2	87.79	49.01	56.68	86.45

The table below shows values for the confusion matrix. Out of the total 2,500 sentences used while testing, approximately 2,100 were classified correctly as positive and negative reviews. There were 250 false positives and 150 false negatives in the testing dataset results. This gives the model an overall accuracy of 84% for classifying the reviews correctly for the sentiment value.

Table 2. Confusion Matrix

Predicted Values	**Actual Values**	
	Positive	**Negative**
Positive	1400	250
Negative	150	700

The table below compares the accuracy level achieved by applying two types of attention techniques: Luong and Bahdanau. Luong attention dot products/ multiplies the previous states' decoder to the current encoder which is exactly what is needed for this model mechanism to work. Bahdanau attention uses an addition mechanism instead of the dot product. While the two give almost the same results, Luong attention is a lot more time and space efficient than Bahdanau's. To compare the time taken by each of these mechanisms, Table 4 shows the time taken by each attention type for mitigating perturbation effects of each of the attack types used. Given that the original sentences took almost 2-3 times more minutes to run by Bahdanau, the sentences used as data input by Bahdanau were truncated further to only keep adjectives and adverbs. This resulted in faster run times but gave much worse accuracy results than Luong attention because of loss of the data required for contextual understanding in BERT processing.

The accuracy results are shown in table 3.

Table 3. Comparing attentio

Model Type	**Type of Attack**	**Number of Perturbed words per sentence**	**Accuracy before attack (%)**	**Accuracy after attack (%)**	**Accuracy after using Bahdanau attention**	**Accuracy after using Luong attention**
BERT	Swap	1	85.43	61.98	62.23	83.58
BERT	Swap	2	84.65	53.65	65.45	83.57
BERT	Drop	1	85.23	59.09	64.34	80.45
BERT	Drop	2	86.45	44.15	61.43	81.88
BERT	Swap and Drop	1	87.67	57.94	63.33	85.67
BERT	Swap and Drop	2	84.43	42.17	67.77	82.13
BERT	Swap	2	87.79	49.01	66.89	86.45

Below is a table quantifying the time taken to run by each of the attention mechanisms for the same types of attacks each.

Table 4. Comparing time taken to run defense models

Model Type	**Type of Attack**	**Number of Perturbed words per sentence**	**Time taken to run BERT only (minutes)**	**Time taken to run Luong attention defense (minutes)**	**Time taken to run Bahdanau attention defense (minutes)**
BERT	Swap	1	10	30	60
BERT	Swap	2	15	45	90
BERT	Drop	1	10	30	60
BERT	Drop	2	10	45	95
BERT	Swap and Drop	1	12	40	70
BERT	Swap and Drop	2	11	62	105
BERT	Swap	2	15	60	90

Hypothesis testing:

In chapter 3, hypothesis 1 tests whether the model accuracy is the same with and without perturbed data. H_0 states that the model accuracy would be the same with or without data perturbation. H_α states that there is a difference in the accuracy results after introducing a perturbed dataset while model testing. For this statistical analysis, the confidence level chosen was 95%. Thus, the α value for this statistics test is 0.05. For a testing dataset of 2,500 data points, the t-table indicated that the p value is 0.025. Given that $p < \alpha$, since 0.025 is smaller than 0.05, H_0 can be rejected and therefore, H_α is confirmed. Hypothesis 1 confirms that the model accuracy reduces after data perturbation as compared to the original accuracy on non-perturbed data.

Hypothesis 2 checks for accuracy after applying the word correction model. H_0 states that the model accuracy would be the same with or without applying the word correction model before the BERT model. H_α states that there is a difference in the accuracy results after introducing the word correction model. For this statistical analysis, the confidence level chosen was 95%. Thus, the α value for this statistics test is 0.05. From the t-table the p value is 0.025. Given that $p < \alpha$, since 0.025 is smaller than 0.05, H_0 can be rejected and therefore, H_α is confirmed. The hypothesis that the accuracy is restored to the original level after applying the word correction model even though the data set is perturbed is confirmed.

CHAPTER 5. CONCLUSION

5.1 Discussion

NLP models are vulnerable to the slightest perturbation in data such as word character swapping and dropping. Making NLP models robust against such attacks is very essential because of the variety of applications of such models and their importance in spam detection, chat-box applications, and in many more fields.

Some of the most popular perturbation techniques are swapping and dropping word characters, adding a synonym and adding an empty word such as “the” which does not change the meaning of the sentence. Out of the above methods, adding, and swapping of characters tends to break the NLP models such as BERT the most (Pruthi, et al.). Most of these attacks also go unnoticed by the human eye. According to a study, 90% of participants were not able to recognize word perturbations by adding and swapping characters as long as the last and the first character of the word remained the same. Given the probability of model breakage and failure of perceptivity to the human eye, the attack mechanisms chosen for the study were adding and swapping of internal characters of a word.

When trained on non-perturbed data, the BERT model tends to yield an accuracy in 80s percentage-wise on movie reviews when compared to the sentiment column in the IMDB dataset. Longer sentences tend to have higher accuracy levels simply because of more data per processing point. This shows that BERT uses a contextual mechanism for sentiment classification which makes it perform better when there is more data to process and find context out of.

When there were reviews that had conflicting sentiments such as “the movie was great but the actor was bad”, the BERT model classified the review according to the last sentiment in the sentence which in the case of the example would be a negative review since the last word is bad which is a negative sentiment word.

The drop in accuracy is directly proportional to the number of perturbations introduced in a sentence. In table 1, there was a higher drop in accuracy in data sets that were perturbed with 2 words per sentence over data sets that were perturbed with 1 word per sentence.

The time taken to run the word correction model was greater for sentences with more perturbations. This is because there were more computation and correction to be performed on sentences with more than one perturbation. Table 4 indicated that the time taken to run is directly proportional to the number of perturbations in a sentence. The more the perturbations, are the longer is the time taken to run.

Accuracy is restored back to an almost normal level by the word correction layer regardless of the number of perturbations used in a sentence. Despite adding more perturbations to a few data sets, the accuracy of these data sets was restored to an almost non-perturbed data level as shown in table 1. This is an indication that the word correction model works correctly and that it can handle more than one perturbation in a sentence.

While the word correction model improves the accuracy of NLP models on non-normal data, it makes the processing more computationally complex requiring more GPU support and time to process. As more LSTM layers are added to the word correction model, the slower it runs. It also becomes more computationally intense requiring additional GPU and data processing support.

Circling back to the research questions stated in section 1, the results confirmed that it was possible to develop a more robust BERT model by adding a word correction layer. The BERT model was able to classify movie review sentiments into positive or negative at an accuracy when subject to perturbed data to levels similar to when run on non-perturbed data.

The Luong attention used with an encoder (LSTM and BiLSTM layer) and decoder (LSTM) served the purpose of the study more effectively than did Bahdanau attention mechanism since it was more time-efficient than the latter.

The word correction layer restored the accuracy of the BERT model to a normal level and the overall accuracy of the robust BERT was 84%.

5.2 Conclusion

The results in section 4 prove that adding a word correction layer can help make NLP models such as BERT more robust against attacks such as word perturbations and word dropping in sentences. The word correction layer consisting of an encoder (LSTM and BiLSTM) and a decoder (LSTM) along with the Luong attention mechanism is the most effective in building a robust BERT model.

5.3 Future Work

Adding a sensitivity score for data points would help quantify the sensitivity of data sets towards different types of perturbations. Suitable word correction/ defense mechanisms can be applied to data points according to the sensitivity score which will make the defense mechanism a lot more effective. The sensitivity score before and after the attack can be studied and can be used to develop even more effective word correction models. The sensitivity score before and after the defense can help quantify the effectiveness of the defense mechanisms.

The word correction model can be made more complex by adding more layers of LSTM. NLP models such as BERT can have an in-built word correction model instead of having a separate model which would make NLP models inherently more robust to word perturbations. However, adding such a word correction layer would make NLP models such as BERT more computationally intensive.

REFERENCES

Alzantot, M., Sharma, Y., Elgohary, A., Ho, B.-J., Srivastava, M., & Chang, K.-W. (2018). Generating Natural Language Adversarial Examples. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. doi: 10.18653/v1/d18-1316.

Athalye Anish, Carlini Nicholas, and Wagner David. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. International Conference on Machine Learning (ICML), 2018.

Carlini, N., & Wagner, D. (2017). Towards evaluating the robustness of neural networks.

Constance Bitso, Ina Fourie, and Theo JD Bothma. 2013. Trends in transition from classical censorship to internet censorship: selected country overviews. Innovation: journal of appropriate librarianship and information work in Southern Africa, 2013(46):166–191.

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

Graham Ernest Rawlinson. 1976. The significance of letter position in word recognition. Ph.D. thesis, University of Nottingham.

Giorgio Fumera, Ignazio Pillai, and Fabio Roli. 2006. Spam filtering based on the analysis of text information embedded into images. Journal of Machine Learning Research (JMLR).

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., . . . Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672–2680).

Goodfellow, I., Shlens, J., & Szegedy, C. (2015). Explaining and Harnessing Adversarial Examples. International Conference on Learning Representations (ICLR).

Gu, T., Dolan-Gavitt, B., & Garg, S. (2017). BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain.

Honglak Lee and Andrew Y Ng. 2005. Spam deobfuscation using a hidden markov model. In CEAS.

Liang, B., Li, H., Su, M., Bian, P., Li, X., & Shi, W. (2018). Deep Text Classification Can be Fooled. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. doi: 10.24963/ijcai.2018/585.

Luong, T., Pham, H., & Manning, C. D. (2015). Effective approaches to Attention-based Neural machine translation. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. doi:10.18653/v1/d15-1166

Matt Davis. 2003. Psycholinguistic evidence on scrambled letters in reading. https://www.mrc-cbu.cam.ac.uk/ people/matt.davis/cmabridge/.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2019).

M Nadkarni, P., Ohno-Machado, L., & W Chapman, W. (2011). Natural language processing: an introduction.

Networks. *2017 IEEE Symposium on Security and Privacy (SP)*. doi: 10.1109/sp.2017.49

Pruthi, D., Dhingra, B., & Lipton, Z. C. (2019). Combating Adversarial Misspellings with Robust Word Recognition. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. doi: 10.18653/v1/p19-1561.

Wei Emma Zhang, Quan Z. Sheng, and Ahoud Abdulrahmn F. Alhazmi. Generating Textual Adversarial Examples for Deep Learning Models: A Survey. CoRR, abs/1901.06796, 2019.

APPENDIX A. CODE FOR WORD CORRECTION MODEL

```
import pandas as pd
import numpy as np
data=pd.read_csv('IMDB Dataset.csv')
```

```
data['polarity']=0
for i in range(len(data)):
    if data.ix[i,'sentiment']=='positive':
        data.ix[i,'polarity']=1
    else:
        data.ix[i,'polarity']=0
data.head()
```

```
max_seq_length=0
for i in range(len(data)):
    if len(data.iloc[i,0])>max_seq_length:
        max_seq_length=len(data.iloc[i,0])
train_text = data[0:25000]['review'].tolist()
train_text = [' '.join(t.split()[0:max_seq_length]) for t in train_text]
train_label = data[0:25000]['polarity'].tolist()

test_text = data[25000:]['review'].tolist()
test_text = [' '.join(t.split()[0:max_seq_length]) for t in test_text]
test_label = data[25000:]['polarity'].tolist()
```

```


for seq_index in range(10):

    input_seq = encoder_input_data[seq_index: seq_index + 1]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('Input sentence:', input_texts[seq_index])
    print('Decoded sentence:', decoded_sentence)
```

```
#developed by referencing pluralsight.com

def encoder_sequence(input_seq):
    states_value = decoder_model.predict(input_seq)

    target_seq = np.zeros((1, 1, num_encoder_tokens))

    target_seq[0, 0, target_token_index['\t']] = 1.
    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = model.predict(decoder_input_data[1],
encoder_input_data[1])
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]
        print(sampled_token_index,'sampled_char'+sampled_char)
        decoded_sentence += sampled_char

        if (sampled_char == '\n' or
           len(encoder_sentence) > max_encoder_seq_length):
            stop_condition = True

        target_seq = np.zeros((1, 1, num_encoder_tokens))
        target_seq[0, 0, sampled_token_index] = 1.

        states_value = [h, c]

    return output_tokens
```

```
def trunc(x, feature_cols=range(4), target_cols=range(4), train_len=400,
test_len=100):
    in_, out_, lbl = [], [], []
    for i in range(len(x)-train_len-test_len+1):
        in_.append(x[i:(i+train_len), feature_cols].tolist())
        out_.append(x[(i+train_len):(i+train_len+test_len),
target_cols].tolist())
        lbl.append(x[i+train_len, label_col])
    return np.array(in_), np.array(out_), np.array(lbl)
X_in, X_out, lbl = truncate(x_normalize, feature_cols=range(3),
target_cols=range(3),
                            label_col=3, train_len=200, test_len=20)
print(X_in.shape, X_out.shape, lbl.shape)
```

```
for seq_index in range(100):
    input_seq = encoder_input_data[seq_index: seq_index + 1]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('Input sentence:', input_texts[seq_index])
    print('Decoded sentence:', decoded_sentence)
```

```python
#developed by referencing machinelearningmaster.com

from __future__ import print_function

from keras.models import Model
from keras.layers import Input, LSTM, Dense
import numpy as np
from keras.layers import Bidirectional

batch_size = 64
epochs = 1
latent_dim = 512
num_samples = 1800
data_path = 'IMDB dataset.csv'

input_texts = []
target_texts = []
input_characters = set()
target_characters = set()
lines=len(data)
for line in range(min(num_samples, lines)):
    input_text = data.iloc[line,0]
    target_text = data.iloc[line,0]
    target_text = ', ' + target_text + '\n'
    input_texts.append(input_text)
    target_texts.append(target_text)
    for char in input_text:
        if char not in input_characters:
            input_characters.add(char)
    for char in target_text:
        if char not in target_characters:
            target_characters.add(char)

input_characters = sorted(list(input_characters))
target_characters = sorted(list(target_characters))
num_encoder_tokens = len(input_characters)
num_decoder_tokens = len(target_characters)
max_encoder_seq_length = max([len(txt) for txt in input_texts])
max_decoder_seq_length = max([len(txt) for txt in target_texts])


input_token_index = dict(
    [(char, i) for i, char in enumerate(input_characters)])
target_token_index = dict(
    [(char, i) for i, char in enumerate(target_characters)])
```

```
encoder_input_data = np.zeros(
    (len(input_texts), max_encoder_seq_length, num_encoder_tokens),
    dtype='float32')
decoder_input_data = np.zeros(
    (len(input_texts), max_decoder_seq_length, num_decoder_tokens),
    dtype='float32')
decoder_target_data = np.zeros(
    (len(input_texts), max_decoder_seq_length, num_decoder_tokens),
    dtype='float32')

for i, (input_text, target_text) in enumerate(zip(input_texts,
target_texts)):
    for t, char in enumerate(input_text):
        encoder_input_data[i, t, input_token_index[char]] = 1.
    encoder_input_data[i, t + 1:, input_token_index[' ']] = 1.
    for t, char in enumerate(target_text):
        decoder_input_data[i, t, target_token_index[char]] = 1.
        if t > 0:
            decoder_target_data[i, t - 1, target_token_index[char]] = 1.
    decoder_input_data[i, t + 1:, target_token_index[' ']] = 1.
    decoder_target_data[i, t:, target_token_index[' ']] = 1.
encoder_inputs = Input(shape=(None, num_encoder_tokens))
encoder = Bidirectional(LSTM(latent_dim, return_state=True))
print(encoder(encoder_inputs))
encoder_outputs, state_h, state_c,state_h_r, state_c_r =
encoder(encoder_inputs)
encoder_states = [state_h, state_c,state_h_r, state_c_r]

decoder_inputs = Input(shape=(None, num_decoder_tokens))
decoder_lstm = Bidirectional(LSTM(latent_dim, return_sequences=True,
return_state=True))
decoder_outputs, _, _,_,_ = decoder_lstm(decoder_inputs,
                                     initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit([encoder_input_data, decoder_input_data], decoder_target_data,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2)
```

```
model.save('s2s.h5')
```

```
def luong(input):
        line = line.lower()
        Xtype = torch.FloatKeras
        ytype = torch.LongKeras
        is_cuda = torch.cuda.is_available()

        if is_cuda:
            self.model.cuda()
            Xtype = torch.cuda.FloatKeras
            ytype = torch.cuda.LongKeras
            if self.use_background: self.model_bg.cuda()

        X, _ = get_line_representation(line)
        tx = Variable(torch.from_numpy(np.array([X]))).type(Xtype)

        if self.use_elmo or self.use_elmo_bg:
            tx_elmo = Variable(batch_to_ids([line.split()])).type(ytype)

        return tx_elmo
```

```
def bahdanau(input):
        line = line.lower()
        Xtype = tfa.FloatKeras
        ytype = tfa.BahadanauAttention
        is_cuda = tfa.cuda.is_available()

        if is_cuda:
            self.model.cuda()
            Xtype = tfa.FloatKeras
            ytype = tfa.BahadanauAttention
            if self.use_background: self.model_bg.cuda()

        X, _ = get_line_representation(line)
        tx = Variable(tfa.from_numpy(np.array([X]))).type(Xtype)

        if self.use_elmo or self.use_elmo_bg:
            tx_elmo = Variable(batch_to_ids([line.split()])).type(ytype)

        return tx_elmo
```

```
#developed by referencing pluralsight.com

def decode_sequence(input_seq):
    states_value = encoder_model.predict(input_seq)

    target_seq = np.zeros((1, 1, num_decoder_tokens))

    target_seq[0, 0, target_token_index[',']] = 1.


    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c,h_r,c_r = decoder_model.predict([target_seq],
[states_value[0]])

        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]
        decoded_sentence += sampled_char

        if (sampled_char == '\n' or
           len(decoded_sentence) > max_decoder_seq_length):
            stop_condition = True

        target_seq = np.zeros((1, 1, num_decoder_tokens))
        target_seq[0, 0, sampled_token_index] = 1.


        states_value = [h, c]

    return decoded_sentence
```