INTERPRETING NATURAL LANGUAGE PROCESSING (NLP) MODELS AND LIFTING THEIR LIMITATIONS

By

REZA MARZBAN

Bachelor of Science in Information Technology Engineering
University of Applied Science and Technology
Tehran, Iran
2017

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
July, 2021

INTERPRETING NATURAL LANGUAGE

PROCESSING (NLP) MODELS AND LIFTING THEIR

LIMITATIONS

Dissertation Approved:

Dr. Christopher Crick

Dissertation Adviser

Dr. Johnson Thomas

Dr. Nohpill Park

Dr. Guoliang Fan

ACKNOWLEDGEMENTS

I would like to thank my advisor, Professor Christopher Crick, who supported me throughout my Ph.D. both mentally and academically. One of the choices throughout my life that I am proud of is choosing him as my advisor. In addition to that, I am grateful to all Robotic Cognition lab members who created a warm, friendly environment for all of us to progress in our research.

I would also like to thanks my parents who were there for me in stressful times. Last but not least I am grateful to my brother, Ali, who inspired me whenever I needed it. This dissertation was simply impossible without my family.

Name: Reza Marzban

Date of Degree: JULY, 2021

Title of Study: INTERPRETING NATURAL LANGUAGE PROCESSING (NLP) MODELS AND LIFTING THEIR LIMITATIONS

Major Field: Computer Science

Abstract:
There have been many advances in the artificial intelligence field due to the emergence of deep learning and big data. In almost all sub-fields, artificial neural networks have reached or exceeded human-level performance. However, most of the models are not interpretable and they perform like a black box. As a result, it is hard to trust their decisions, especially in life and death scenarios. In recent years, there has been a movement toward creating explainable artificial intelligence, but most work to date has concentrated on image processing models, as it is easier for humans to perceive visual patterns. There has been little work in other fields like natural language processing. By making our machine learning models more explainable and interpretable, we can learn about their logic, optimize them by removing bias, overcome their limitations, and make them resistant against adversarial attacks. This research dissertation is concentrated on making deep learning models that handle textual data, more understandable, and also use these insights in order to boost their performance by overcoming some of the common limitations. In addition to that, we use this knowledge to target words for designing efficient and effective textual adversarial attacks.

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER I

INTRODUCTION

In the big data era, traditional naive statistical models and machine learning algorithms are not able to keep up with the growth in data complexity. Such algorithms are the best choice when our data size is limited and nicely shaped in tabular formats. Previously, we were interested in analyzing structured data in databases, inserted by an expert, but now we use machine learning in every aspect of life. Most of the data is unstructured, such as images, text, voice, and videos. Besides, the amount of data has increased significantly. Traditional machine learning algorithms cannot handle these types of data at our desired performance level. Artificial Neural Networks (ANNs) have undergone several waves of popularity and disillusionment stretching back to 1943. Recently, due to increases in computing power and data availability, the success and improvement of ANNs and deep learning have been the hot topic in machine learning conferences. In many problem areas, deep learning has reached or surpassed human performance and is the current champion in fields from image processing and object detection to Natural Language Processing (NLP) and voice recognition.

Although deep learning has increased performance across the board, it still has many challenges and limitations. One main criticism of deep learning models is that they are black boxes: we throw data at it, use the output, and hope for the best, but we do not understand how or why. This is less likely to be the case with traditional statistical and machine learning methods such as decision trees. Such algorithms are interpretable and easy to understand, and we can know the reasoning behind the decisions they make. Explainability is very important if we want people to rely on our models and trust their decisions. This becomes crucial when the models' decisions are life-and-death situations like medicine or autonomous vehicles, and it is the reason behind the trend toward explainable artificial intelligence. In addition to boosting users' trust, interpretability also helps developers, experts and scientists learn the shortcomings of their models, check them for bias and tune them for further improvement.

Many papers and tools have recently contributed to explainability in deep learning models, but most of them have concentrated on machine vision problems, as images are easier to visualize in a 2-dimensional space. They can use segmentation and create heat-maps to show users which pixel or object in the image is important, and which of them caused a specific decision. Humans can easily find visual patterns in a 2-dimensional space. However, there is also a need for model interpretability in other contexts like NLP, the science of teaching machines to communicate with us in human-understandable languages. Very few studies investigated the explainable AI in NLP. As NLP data mostly consists of texts, sentences, and words, it is very hard to visualize in a 2-D or 3-D space that humans can easily interpret, even though visualization of a model is an important part of explainable AI.

There is a huge need for NLP models that are both explainable and also have high performance. Explainability of these models will help us in 4 ways:

1- **Justify**: provide justification for each and every prediction.

2- **Improve**: find weaknesses and remove them (optimization).

3- **Control**: removing bias according to legislation (e.g.: gender, race, …).

4- **Discover**: find patterns invisible to humans.

In addition to this gap in the Explainability of NLP models, there are other challenges and shortcomings as well. All of the current state-of-the-art models have a limitation on the number of tokens in their input, as a result, they cannot process very long texts, however, humans can read a very long book and remember the gist of it by the end of it.

In addition to that, we have seen that they are extremely vulnerable to adversarial attacks. If an attacker creates adversarial examples by adding a little well-chosen noise to input, although humans still classify them appropriately, the models can be deceived and classify them incorrectly, potentially leading to catastrophic results. Adversarial attacks come in two flavors: white box and black-box attacks. In a white-box attack, the attacker has access to the information, details, and weights of the attacked model. Conversely, in the black box context, the attacker does not have any internal model information. Obviously, in real-life scenarios, black box attacks are more realistic.

There have been many works on adversarial attacks and creating defense mechanisms against them. Most have concentrated on machine vision and image processing, as it is much easier to visualize and compare images in a 2-dimensional format, whereas in the Natural Language Processing field, there has significantly less work, as it is much more challenging to visualize and compare textual data. Adversarial attacks in NLP consist of two phases: choosing the words to attack and choosing the technique of manipulation or perturbation. There are many different word-level perturbation techniques in NLP (e.g. replace, delete, add, swap); these may seem different to humans but all of them result in the same behavior in NLP models. They will create a noisy word that is not in the model dictionary, and the model will assign it an ‘unknown’ label. Most existing adversarial attacks in NLP choose their targeted words to attack one input at a time, where each attack is applied to the most important words for a particular input instance. While such a technique can be successful, it requires access to the texts

to be attacked in advance, in order to train the attacker model. They cannot attack brand new texts even when they are from the same source and domain.

This dissertation is concentrated on making NLP models more interpretable by introducing various tools and techniques that can be used to extract logics and insights from models and use them to find models' shortcomings and blind spots, improve future models, remove their limitations and make them resistant to adversarial attacks. Various types of NLP models (e.g. CNNs, LSTMs, Transformers, and BERT) are included in our study and we also monitor their behavior across different types of datasets and tasks like Sentiment Analysis. The purpose of this study was to make NLP models more understandable, interpretable, optimized, and robust.

The contribution of this dissertation is divided into four main categories:

1- Removing Sequence Length Limitation by introducing a custom encoder that compresses long textual input, so that they can be processed in regular NLP models. This technique significantly increases the performance of the target model if the dataset contains long texts, and does not change the accuracy if the input is short.

2- Presenting an equation that can be used on CNNs in order to extract the importance rate of each and every unique word in the corpus, and identifying the most important words. By doing so, we can understand the global logic of the model and use that insight to boost the performance of future models significantly. The correctness of this equation is proved by creating a brand new model trained on just 5% of most important words and achieving almost equal accuracy in comparison with the original model (trained on the whole data) but training much faster.

3- Proposing a new NLP adversarial attacks method that is much more efficient than its preceding methods. It uses the aforementioned equation to target the most critical words to attack. Our experiments on various datasets and NLP tasks show that our method performs equal or better

when compared to previous attack methods, and its running time is around 39 times faster than previous models.

4- Improving and generalizing task 2, which created an importance rate equation on CNN, we created a secondary tool that is as effective as the original one, but it is more generalizable. In other words, it can be applied to any type of NLP model (e.g. CNNs, LSTMs, Transformers) and its equation is not dependent on the model architecture or weights.

All of these hypothesizes were tested and proved through extensive experiments in different environments (i.e. various datasets and different types of NLP models).

CHAPTER II

REVIEW OF LITERATURE

Statistical methods and Machine Learning models have a long history initiated by Bayes Theorem back in 1763. They kept evolving and expanding inspired by the challenges they faced. The first Neural Network machine was built in 1951. Since then, Deep Learning [1], [2] had faced ups and downs. Meanwhile recently we have observed a great peak in Deep Learning models due to two facts: 1- The amount of data generated every day in the Big Data era [3], 2- The computing power achieved using GPUs and TPUs. Nowadays, Deep Learning model performance is comparable to human experts in most of the tasks.

2.1 Natural Language Processing (NLP)

NLP has long been an important application area for artificial intelligence and machine learning. NLP [4]–[6] is the science of teaching a machine to understand and produce content based on human languages. Recently, there has been a huge improvement in NLP tasks with the help of deep learning. Artificial neural networks have exceeded traditional machine learning algorithms in many different fields, like machine vision, and NLP has benefited likewise. NLP has many subcategories, The most basic one is sentiment analysis [7].

To apply standard deep learning architectures to NLP, after tokenizing textual data with available tools like NLTK [8], tokens (a.k.a. words) must be transformed into an amenable numerical format. There are multiple ways of doing this, such as a one-hot encoder or n-gram representation, but by far the most common approach is to use the Word2Vec [9] algorithm. This creates a custom-length vector that represents the semantic meaning of a particular word and attempts to reflect the relationship of words with each other. Word2Vec mappings can be trained from scratch, but it is also common to use one of the available pre-trained word representations like Glove [10].

After preprocessing texts, and transforming them into numerical vectors, the data can be fed into a deep learning model. Various famous model architectures work well on NLP tasks. The First natural choice is LSTM [11] which is an advanced version of a Recurrent Neural Network (RNN). It is designed with time-series data in mind, and it has internal memory. According to its gate weights, it will decide what to remember and what to forget. GRUs [12] are also another type of RNN architecture that employs gate technology. Another famous deep learning architecture is CNN. We know that CNNs have revolutionized image processing tasks, but CNNs can also be applied to textual data [13]–[16]. Yin [17] compared RNNs and CNNs on various NLP tasks and studied the performance of each.

In 2017, a new generation of NLP models appeared, starting with Vaswani's first Transformer attention-based architecture [18]. Instead of remembering an entire text, it assigns an attention weight to each token, which allows it to process much longer texts. The attention technique enabled the creation of much more advanced transformer-based models [19] like BERT [20], RoBERTa [21], Xlnet [22] and GPT-3 [23].

2.2 Explainable Artificial Intelligence (XAI)

Explainable artificial intelligence (XAI) [24] helps us to trust AI models, improve them by identifying blind spots or removing bias. It involves creating explanations of models to satisfy nontechnical users [25] and helps developers to justify and improve their models. Such models come in various flavors [26]; they can provide local explanations of each prediction or globally explain the model as a whole. Layer-wise relevance propagation (LRP) [27], [28] matches each prediction in the model to the input features that have caused it. LIME [29] is a method for providing local interpretable model-agnostic explanations. These techniques help us to trust deep learning models.

Most of the XAI community has concentrated on image processing and machine vision as humans find it easy to understand and find patterns in visual data. Such research has led to heat maps, saliency maps [30], and attention networks [31]. However, other machine learning fields, such as NLP, have not yet seen nearly as many research efforts. There have been many improvements in NLP models' performance in recent years [6] but very few of them concentrate on creating self-explanatory models.

Jesse [32] visualized the attention mechanism used in Transformers and BERT. They tried to map each word's attention to the rest of the sentence. Arras [33] tried to find and highlight the words in a sentence leading to a specific classification using LRP and identified the words that vote out the final prediction. This can help identify when a model arrives at a correct prediction through incorrect logic or bias, and provide clues toward fixing such errors. Li [34] introduced methods that illustrate the saliency of word embeddings and their contribution to the overall model's comprehension. Rajwadi [35] created a 1-D CNN for a sentiment analysis task and used a deconvolution technique to explain text classification. They estimate the importance of each word to the overall decision by masking it and checking its effect on the final classification score. These

kinds of local explanations help to confirm single model predictions, but methods for understanding the global logic of models and specific architectures are necessary to provide insight for improving future models. Such techniques are model specific and dependent on the architecture used. With global explainability, we can find and overcome models' limitations and weaknesses.

2.3 Adversarial Attacks in NLP

The various deep learning models have contributed to rapid improvement in NLP task performance, but they also have created challenges. To start with, deep learning models are not intuitive or explainable. Besides, they can easily be manipulated by adversarial examples. Adversarial examples are created by attackers by adding a little well-chosen noise in order to deceive the models into misclassifying them potentially leading to catastrophic results.

Adversarial attacks have recently been a major research focus in machine vision [36], [37], as it is very easy for humans to recognize patterns in two dimensions. There has been also some work on Adversarial attacks defense mechanisms [38], [39]. However, very few have worked on NLP adversarial attacks. The same adversarial attack techniques can be applied to textual data as well [40], [41]. Gao [42] created a technique called WordBug that analyses each and every input to find the most important words in that text, then targeting them in an attack. They have used a temporal score extracted from a bidirectional LSTM to detect important words in each text.

CHAPTER III

METHODOLOGY

3.1 Benchmark Datasets and Preprocessing

In order to demonstrate our approach, we have used a broad range of data of various types. The IMDb Large Movie Review Dataset [43][1] contains movie review texts classified in a binary fashion according to sentiment. This allows us to test the performance of every model on very long sequences. Most of its reviews consist of sequence lengths of greater than 100 words (tokens).

In addition to the IMDb, we used a dataset of Amazon Kindle book reviews and user ratings[2]. This problem again belongs to sentiment analysis, but rather than a binary classification, it has categorical ratings with 5 classes. Besides, we used a large dataset of Stack Overflow questions and tags[3], where each question has a label or a tag that specifies which language or concept the question is about. It has 20 possible tags. The mean sequence length in the Stack Overflow dataset is the largest. Our final dataset is the 20 Newsgroup dataset [44][4] that contains full-text news articles classified into twenty possible categories.

[1] https://ai.stanford.edu/~amaas/data/sentiment/
[2] http://jmcauley.ucsd.edu/data/amazon/
[3] https://console.cloud.google.com/marketplace/product/stack-exchange/stack-overflow
[4] http://qwone.com/~jason/20Newsgroups/

These four datasets were used in testing versions of our models and the various comparison benchmarks. In the cleaning phase, we removed all numerical values, symbols, and stop words. We also removed words that did not appear in our entire corpus at least twice and words of only one or two characters and converted the whole corpus to lower case. The models expect an input vector of fixed length, but different sentences and paragraphs obviously have different lengths. In order to solve this problem, we choose a number that is close to the maximum of the length of all sentences in the whole dataset, ignoring a few outliers which are unusually long in comparison to other records. We pad the sentences with zero values at the end of records that have fewer words than our threshold.

In order to create the embedding, we used the Stanford GloVe pre-trained embedding trained on Wikipedia. Word2Vec embeds words within vectors of consistent but arbitrary dimensionality, and in this case, we chose 100 for the embedding vector size. These numbers represent the semantic relationships among words. As a result, we expect the distance between vectors of “Boston” and “Massachusetts” to be near the distance between “Austin” and “Texas”. This will help our model to understand the overall meaning of each sentence. This embedding matrix would be the preprocessed input of our future models. As we train the larger overall model, we can train the embedding at the same time, so that the embedding representation is updated after each iteration to represent the words more accurately, or we can fix the embedding weights in advance so that the model can concentrate on training other layers’ weights.

3.2 Sequence Length Limitation [45]

All current state-of-the-art NLP models have a hard limitation on their memory. The number of tokens (words) they can remember in a dataset is limited. Some models do not accept long textual

data and truncate the longer ones to their limited length, while others accept them but would not remember all of it.

RNN has the shortest memory among all recurrent neural networks, LSTM has a larger memory but still not large enough. Attention-based models like BERT that use Transformer blocks have a hard limit on the length of sequence that they accept (512 tokens). However, humans can read a very long book and remember the gist of it.

In order to overcome the maximum sequence length limitation, we propose inserting an encoder layer at the beginning of each of these models. By doing so, we can fit in longer data to our current models. Our encoder uses the output of the embedding layer applied to a text with a large word count and reduces its dimensionality to a size compatible with a model. We hypothesized that the encoder will increase the performance of the model when the data is long enough.

3.2.1 Encoder Technical Details

The architecture of the encoder is very simple, consisting of two layers. The first layer is a 1-dimensional convolutional layer and the second one is a 1-dimensional max-pooling layer. The number of filters in our convolutional layer is equal to the length of the embedding vector used to represent words, which was equal to 100 in our case. The filter length, as well as the pool size in the pooling layers, are hyper-parameters that can be tuned; in our case, they were set to 5 and 2 respectively. Our encoder is applied right after the embedding layer. If the output of the embedding layer has the shape of m * k, in which m is the number of words, and k is the length of the embedding vector for each word, the output of the encoder layer is calculated by equation 1:

$$(m, k) = \left(\frac{m - f + 1}{s},\ k\right) \tag{1}$$

In Equation 1, the left side is the input shape and the right side is the output shape. f is the length of filters and s is the pool size in the pooling layer. Notice that the depth of input does not change at all, as we intend to reduce the number of sequence steps (words) in our input and not the size of the embedding. If we needed a smaller embedding length, we could simply specify it in the embedding layer. Initializing these hyper-parameters according to our specific model, the output shape is shown in Equation 2.

$$(m, 100) = \left(\frac{m-4}{2},\ 100\right) \tag{2}$$

Thus, if the embedding outputs shape is (512, 100) (512 words, each represented by 100 numerical values), then after passing through our encoder, its shape will be (254, 100). The output will represent 254 important features to be found within the entire text, instead of merely truncating the input text. The output of our encoder can be fed into the model instead of the embedding output. This architecture allows the use of texts twice as long as the original models can handle, but this 2:1 scaling is not required. If further reduction is needed, more pooling or more layers can be added as necessary.

The hyper-parameters in both our encoder and models were selected by randomly sampling the parameter space and selecting for decent performance.

3.2.2 Testing Encoder on LSTM

In order to evaluate our encoder's performance in an LSTM context, we created three models and tested these three on different datasets. Each of these models varies in the structure and most of the

hyper-parameters, but the following hyper-parameters are the same for all three: the learning rate and learning rate decay are both 0.001, we used the Adam optimizer, the batch size was 32, the number of epochs was 20, we applied an L2 regularizer to the last layer of each model, and used the cross-entropy loss function for all models (binary cross-entropy for movie review data, categorical cross-entropy for Amazon Kindle reviews, Stack Overflow, and 20 Newsgroup).

We kept all of these hyper-parameters consistent in order to conduct a fair experiment in comparing these models so that our results are only dependent on the architecture of our network. Our first model is CNN. Our CNN model has 123,601 total parameters to be trained, with two one-dimensional convolutional layers, both with 100 filters of size 5 and stride 1, and with RELU as the activation function. Both max-pooling layer window sizes are 2.

Our LSTM model has 128 nodes and 117,377 parameters ignoring the embedding layer weights, as shown in Figure 1. The original output of this layer has the shape of (?,950,128) but we only send the last output data to the next layer which has the shape of (?,128) instead of the full sequence of data (the question mark in the shapes represents batch size). We have used backward LSTM due to the fact that most of the sentences are padded with zero at the end, and if we do not use backward LSTM, the zeroes will cause the system to forget important information.

The final model we describe is our encoder-LSTM model, which uses the two encoder layers specified before and then uses an LSTM to predict according to the encoder-shortened text. The first part of this model is identical to our CNN model; we just removed the flatten layer and added a 128-node LSTM layer before the dense layer. Notice that the last layer of these models is fully connected with a single output node and sigmoid activation function. When applied to the multi-class datasets, the number of output nodes is increased to be equal to the number of classes, activated with the softmax function. The LSTM model architecture is shown in figure 1 and the LSTM model after inserting our encoder in it is shown in figure 2.

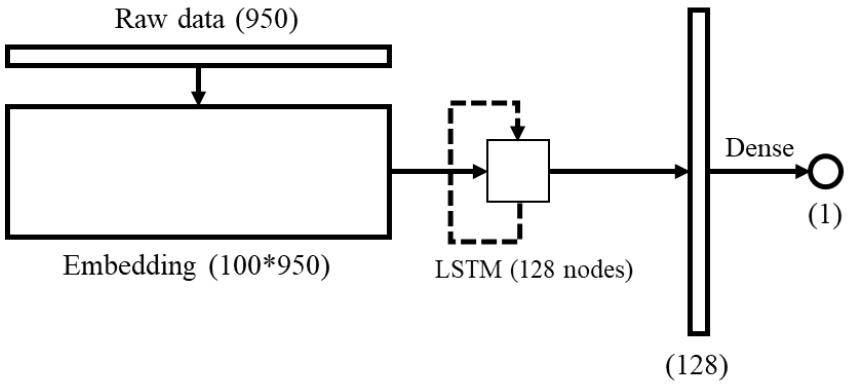

Figure 1. LSTM model

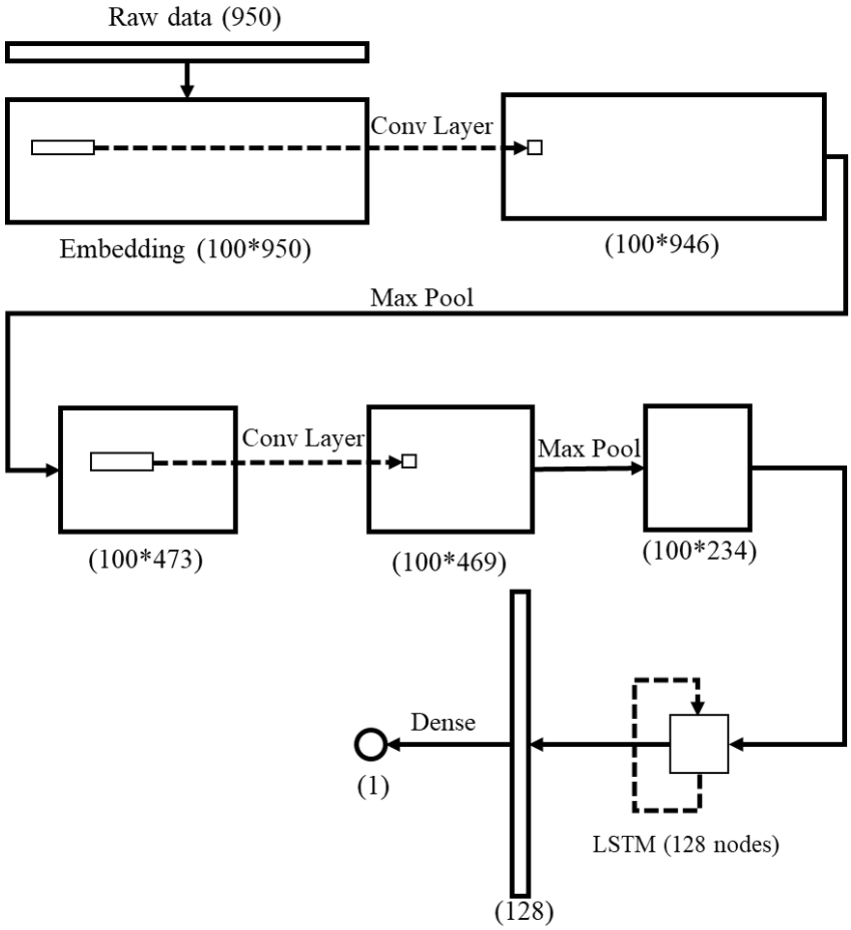

Figure 2. LSTM model equipped with our custom encoder

3.2.3 Testing Encoder on BERT

In order to check the efficiency of our encoder in BERT, we created an implementation of BERT from scratch and then inserted the encoder in the proper place. As the original base BERT has many parameters, and it takes several weeks to pre-train on a huge amount of data, we created a smaller version of BERT and tested and compared its performance with and without an encoder. In order to create a smaller BERT, we reduced the hidden layer size from 768 to 200, the intermediate size from 3072 to 800, and the number of hidden layers from 12 to 6. The implementation details of Base BERT, Small BERT, and Small BERT with encoder can be observed in Table 1. After creating the small BERT, we inserted the encoder after the embedding preprocessing and just before post-processing. The original BERT has a maximum sequence length limitation of 512, which remains the same in the small BERT. Small BERT is much faster than the original Base BERT as it has reduced the number of parameters by 65% but the performance was not affected significantly. It helped us to develop and test models much faster. Even after introducing the encoder into the model, the number of parameters just increased by around 1%. As we did not have access to the original data that BERT was pre-trained on, which was a mixture of book corpora and Wikipedia, we trained our small BERT models on a much smaller dataset consisted of 50 book texts. Both decreasing the size of the model and the size of the pre-training data have affected our accuracy a little, but it was not important to our hypothesis. We used small BERT as an environment to check the efficiency of our encoder. We trained the small BERT with encoder and without encoder on 50 books data for 2,000 epochs and then used these two pre-trained models as the baseline for our fine-tuning tasks. We fine-tuned these models on two sets of data: the IMDb and Stack Overflow datasets. The Amazon Kindle and 20 Newsgroup datasets were sufficiently dissimilar to the training corpus used by the small BERT network that they did not provide useful results. Each fine-tuning was done with 3 epochs. We also downloaded the original pre-trained base BERT model and fine-tuned it on the same data to compare the results.

BERT Version	Base	Small	Small with Encoder
Hidden size	768	200	200
Intermediate size	3072	800	800
Attention heads	12	10	10
Hidden layers	12	6	6
Use Encoder	No	No	Yes
Pre-train data	Book Corpus + Wikipedia	50 Books	50 Books
Total parameters	178,566,653	81,326,847	81,927,447

Table 1. BERT models comparison

3.3 **Interpreting CNNs on Textual data** [46]

This part is concentrated on Explainable Artificial Intelligence (XAI). we created a 1-D CNN for sentiment analysis on the IMDb dataset. However, instead of creating a local explanation for each prediction and decision, we describe the whole model's logic and try to explain it in a layer-wise manner by studying the filters of the trained model.

3.3.1 Basic CNN Model Setup

We use a 1-dimensional CNN for the sentiment analysis problem. The architecture is presented in Table 2. The embedding layer contains parameters for each of 100 dimensions for each word in the corpus, plus one (for unknown words). The embeddings are untrainable in the CNN, having been trained in an unsupervised manner on our corpus, and we did not want to add an extra variable to our evaluation. The first convolutional layer has 32 filters with a size of 5 and a stride of 1. The

second has 16 filters with a size of 5 and a stride of 1. Both max-pooling layers have a size and stride of 2. All convolutional layers use the ReLu activation function. The output layer's activation function is sigmoid (as our target is binary). Our models are trained for 5 epochs with a decaying learning rate of 0.001. The hyper-parameters in our model were selected after performing a random search technique in order to get a decent performance, but in each specific field and different data sets, a more in-depth hyper-parameters search can be performed.

Layer Type	Output Shape	Number of Parameters
Input	(?,250)	0
Embedding	(?,250,100)	2,336,400
Convolutional – 1	(?,246,32)	16,032
Max Pooling - 1	(?,123,32)	0
Convolutional – 2	(?,119,16)	2,576
Max Pooling - 2	(?,59,16)	0
Flatten	(?,944)	0
Fully Connected	(?,128)	120,960
Output	(?,1)	129
Total Parameters		2,476,097
Trainable Parameters		139,697
Non-Trainable Parameters		2,336,400

Table 2. Basic CNN Model

3.3.2 Analyzing and Interpreting Convolutional Layer Filters

In order to understand the logic of our CNN model, we studied the first convolutional layer's filter weights. We created three new models with identical architecture to our baseline model. In two of these new models, we copy the weights of the basic model's first convolutional layer and make them untrainable, then initialize the rest of the layers randomly and train normally. We then shuffled the filter weights in the first layer, either within each filter or across the whole set of filters. In the last model, we randomly initiate the first layer's weight and freeze it.

3.3.3 Word Importance through Activation Maximization

It is difficult to analyze the actual values learned by the convolutional filters. If we cannot interpret them as they are, we are not able to follow the reasoning behind a model's decisions, either to trust or to improve them. As a result, we wanted to concentrate on the input space, and check each word's importance to our model. Previous research has focused on finding significant words that contribute to a specific decision. This is helpful, but it only demonstrates local reasoning specific to a single input and the model's decision in that instance. We are interested in global explanations of the model so that the model can convey its overall logic to users. To do so on our CNN model, we applied Equation 3, which provides an importance rating for each word, according to our first convolutional layer's filters.

$$importance_i = \left\{ \sum_{f=1}^{F} \sum_{s=1}^{S} \sum_{i=1}^{I} \left| w_i * Filter_{f*s*i} \right| \middle| w_i \in Corpus,\ Filter \right\} \tag{3}$$

In equation 3, F is the number of filters, S is the size of filters, and I is the embedding length. w is a word embedding vector with a length of I. Corpus is a matrix of our entire word embedding of size m × I, in which m is the count of unique words in our corpus dictionary. The Filter is a 3-D

tensor of size F × S × I. This equation calculates the sum of activations of all filters caused by a single word. In our models, Corpus contains 13,363 × 100 elements, each w is a vector of 100 numbers, and the Filter size is 32 × 5 × 100. The below equation can be used to compute an importance rating for each and every word in our corpus according to our model's logic. One of the benefits of studying these ratings is that we can understand what types of words affect our model's decisions most strongly. To investigate this further, we dropped unimportant words and trained new models on a subset of data containing just the most important vocabulary. By doing so, we learn from our basic model and can inject its insights into new models, to develop faster and better-performing ones. In order to prove our hypothesis, we created a new model trained on 5% of the most important words and compared its performance and training time to three baseline models. In all of these models, the architecture is the same and we train the embedding weights as well. Our first baseline model uses 100% of the words in the corpus, our second uses 5% of words chosen randomly, and the third uses all of the words except the 5% most important, in other words, the least important 95% of words.

3.4 Adversarial Attacks in NLP

All Adversarial Attacks across all fields consist of two steps: 1- targeting the most critical feature to attack in order to maximize the damage 2- Choosing a Perturbation technique. In this research attacks were conducted at a word level, by other words, each word is looked at as a feature.

3.4.1 Choosing and Targeting Words to Attack

The attack sequence begins by targeting particular words, hopefully, selected to have the greatest possible effect on the model's performance and accuracy. The mechanism for choosing words to attack is the core comparison between techniques.

WordBug: Gao's technique [42] created a bidirectional LSTM and trained the model on each sentence to be attacked, identifying the most important words in each sentence in turn. The approach used three scores: Temporal, Temporal Tail, and combined (which is the mean of the two previous scores). In order to create the Temporal score and Temporal Tail score of the i^{th} word in a sentence, they used Equations 4 and 5, in which n is the number of tokens or words in each data point, x [1 : n] are the n words in each data points and F( ) is a function that maps input words to the probability of belonging to the actual class using the bidirectional LSTM. They observed that their combined score outperformed other approaches.

$$TemporalScore(x_i) = F(x_1, x_2, \dots, x_{i-1}, x_i) - F(x_1, x_2, \dots, x_{i-1}) \quad (4)$$

$$TemporalTailScore(x_i) = F(x_i, x_{i+1}, x_{i+2}, \dots, x_n) - F(x_{i+1}, x_{i+2}, \dots, x_n) \quad (5)$$

The advantage of this technique is that each text is studied and produces the most important words local to that specific text. However, it has two big disadvantages: it needs to be trained on each and every sentence to be attacked, and it cannot be applied to other sentences even from the same context. It is also very slow, as it needs to run the LSTM m × n times in which m is the number of rows and n is the number of tokens in each one.

Activation Maximization: This method (see section **3.3.3**) uses a 1-dimensional CNN, trains on a subset of data, and uses the CNN layer filters to find the importance rate of all unique words in the corpus, based on Equation 3.

The advantage of this method is that it provides a general importance rate applicable to all sentences that come from the same source and distribution. As a result, we can use its insight into new never-before-seen sentences. The other advantage is its speed, as it needs only to train the CNN model

once, for a couple of epochs, and then its filters can be used (via Equation 3) to create the importance rate. The downside of this method is that the globally important words may or may not be the best option for each sentence.

Choosing Words in Each Sentence: After using one of the above models, we target the top t words in each sentence that have the highest importance rate and attack them. In our experiments, we used different following values for t: 0, 10, 20, 40. When t = 0, it means we are not attacking at all; this is a baseline performance for comparison with our attacks. We tested the attacks on three different models to determine whether these techniques are applicable to all models. We used an LSTM, a CNN, and a Transformer model to see how the different attacks behave on each.

3.4.2 Word Perturbation

After selecting the words to attack, a word manipulation or perturbation method must also be chosen. There are four main word attack methods in NLP: replace, delete, add, swap. To human observers, these might have different effects, but all of these methods have the same result, to create a noisy word that is not in the NLP model dictionary. Such words end up with an 'unknown' label assigned by the model. We used swap in this paper; we randomly chose two adjacent middle characters in each targeted word and swapped them. Examples are presented in Table 3. Note that, for purposes of simplifying the presentation, a very short subset of texts is presented in this table. In the original data, the texts are much longer, averaging around 200 words or tokens in each.

Text before the perturbation	Text after the perturbation (with t = 2)
the movie was extremely boring	the moive was extremely broing
how can i import csv file in python	how can i improt csv file in pyhton

Table 3. Word manipulation and perturbation (with t = 2)

3.4.3 Comparison of Adversarial Attacks Methods and Evaluation

In order to compare WordBug vs our Activation Maximization method, we tested both on three brand new models created from scratch on our two benchmark datasets. We used a CNN, an LSTM, and a Transformer model, all of which were built with basic, straightforward architecture. We trained all of these models on the training set. As our attack method is a black-box adversarial attack, we attacked only the test set (since the attack does not have access to the models' information and details).

In the WordBug method, each text in the test set was analyzed and attacked in turn, but in our Activation Maximization method, we fitted our importance rate on a subset of texts and used it for attacking the test set as a whole. We measured the run-time of both techniques while they were learning which words to attack and compared them. In the next step, we evaluated our attacks on the three models we had trained, to observe the attack performance. In both methods, we used swap as the word perturbation and used 0, 10, 20, 40 for our t.

3.5 Deep NLP Explainer

In this section, we introduce Deep NLP explainer which is a new tool that is very similar in its function and purpose to the tool we proposed in section 3.3, while it is much different in its implementation. It also has one main improvement to its predecessor, which is that Deep NLP Explainer is not dependent on the structure of the NLP models and their weights, and it can be applied to any type of NLP models. It uses a technique called “Prediction Slope” to explain NLP models.

3.5.1 Overview

Our contribution in this section creates a brand new explainable AI technique that can be applied to any type of NLP model. Our technique uses a model's inner logic to come up with an importance rate for each and every unique word in the corpus. This has many benefits: we can take a look at the model's most important words to understand its overall general logic. It also can be used to inject insights into future models for further performance improvements. We observed that using our technique, models that were trained on just the 5% most important words perform equally as well as baseline models that have access to 100% of data. However, because only a small fraction of words are used, the model's speed is much greater. In order to create an importance rate for all words, we use and compare the mean significance of the effect of each unique word to the overall sentence prediction. In other words, we compare the change in the prediction of all sentences that contain a specific word with and without that word and use the average prediction change throughout all sentences in the corpus to create an importance rate.

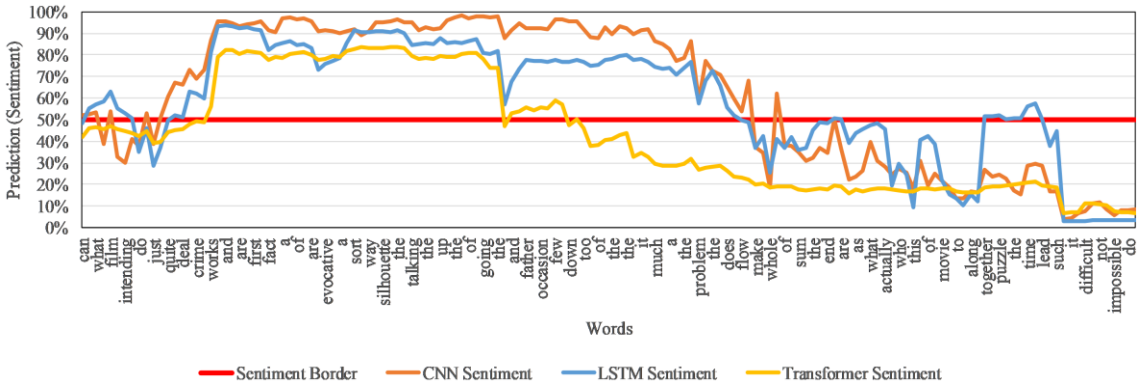

Figure 3. Effect of each word in an IMDb true negative document on the binary prediction of 3 different models (CNN, LSTM, and Transformer). Predictions above 50% represent positive sentiment and below 50% represent negative sentiment.

3.5.2 Introduction of prediction slope

In order to create an importance rate that is independent of the model's inner architecture and details, we created the concept of the prediction slope, which will be defined and clarified in this section. In all machine learning models, and specifically in Artificial Neural Networks (ANNs),

there is a final prediction (a.k.a. output layer), where the model's decisions are found. In NLP our inputs consist of words or tokens. To understand the significance of each of these words on the final prediction, we can feed them one by one into our model and observe the effect of each word on the final prediction. In figure 3, an IMDb document with true negative sentiment is randomly chosen to visualize the prediction slope in a binary classification problem. In a multiclass task like the Stack Overflow dataset, we would have 20 predictions due to the fact that there are 20 classes, and the highest-probability output is taken as the model's decision. We observe this maximum probability class and the effect of adding new words.

$$S_i = F(x_0, x_1, x_2, \dots, x_{i-1}, x_i) - F(x_0, x_1, x_2, \dots, x_{i-1}) \quad (6)$$

In equation 6, S_i is the prediction slope of the i[th] word in a document, and F is simply the function defined by the model, which receives a sentence as input and produces a prediction. x_n is the n[th] word in a document.

3.5.3 Extracting word importance rate from the prediction slope

The prediction slope technique is not entirely new; it has been used in the literature to map a prediction to the most important words in a single input, and is sometimes also known as the temporal score. However, until now it has not been considered as a tool to perform a similar technique to an entire model, and this is where our contribution comes into play.

In each document of our corpus, we can monitor the local significance and effect of each word on the final prediction slope, but we needed a way to find the global importance rate of each unique word to the whole model. In order to do so, we use equation 7, in which S_i is extracted from equation 6, D_j is all documents in our corpus that contain the j[th] unique word, and $|D_j|$ is the count of those documents. Notice that the j[th] unique word in our corpus is the i[th] word that appears in a document. After applying this equation, we will generate a global importance rate for all unique

words applicable to the whole model rather than just a single prediction. This importance rate is the mean value of the prediction slope of a particular word in all sentences containing it.

$$importance_j = \frac{\sum_{D_j} S_i}{|D_j|} \tag{7}$$

3.5.4 Comparing importance rates

The prediction slope importance rate technique can be applied on any type of model, but we chose to use it on a basic Transformer model, as it is one of the hardest models to interpret and understand. Now that we have two importance rates at hand, one generated by activation maximization (Section 3.3.3) and the other created by prediction slope (Sections 3.5.2 and 3.5.3), we performed experiments to compare them. As a result, we created several brand new models that were trained on a subset of the unique words which were selected by one of our two importance rates, and we examined their respective performances.

CHAPTER IV

FINDINGS

4.1 Sequence Length Limitation results

We evaluated our proposed encoder on two models and four benchmark datasets:

- **Models:** LSTM, BERT
- **Datasets:** IMDB movie review, Stack overflow dataset, Amazon products review, and 20 Newsgroup dataset.

4.1.1 Encoder Performance on LSTM

After training our models on the training sets for 20 epochs, we evaluated each of them with test sets. We ran each model on each data set twice, once with embedding layer training and once with untrainable embeddings to check the effect. Table 4 shows their test accuracy over our four different datasets. For the movie reviews, the random prediction baseline is 50% (as it is a binary classification), whereas it is 20% (5 classes) in the Amazon review dataset and 5% (20 possible tags) in both the Stack Overflow and 20 newsgroup datasets. The Stack Overflow and 20 newsgroup datasets only contain texts of 128 words or longer. Out of all of the experiments performed, our encoder-LSTM model achieves better accuracy in seven out of eight cases (a CNN works better in one instance).

Data Set	Embedding	CNN	LSTM	Encoder LSTM
Stack Overflow	Trainable	64.90%	65.38%	**67.17%**
	Untrainable	64.51%	63.82%	**67.55%**
IMDb Movie Review	Trainable	86.26%	85.50%	**87.02%**
	Untrainable	**86.27%**	86.02%	83.81%
Amazon Review	Trainable	53.07%	53.22%	**53.37%**
	Untrainable	50.13%	50.91%	**51.69%**
20 Newsgroup	Trainable	48.32%	38.29%	**50.23%**
	Untrainable	42.26%	35.12%	**45.85%**

Table 4. The encoder in LSTM accuracy comparison

Figures 4-7 and Table 4 show an interaction effect between the model and the embedding trainability. In 9 cases out of 12, the models work better with embedding layer training. In addition, there is a huge main effect from the model factor. The LSTM network does not add much accuracy, while in comparison to the CNN, it takes much longer for the LSTM to be trained. On the other hand, when we are using our encoder with LSTM, it increases accuracy.

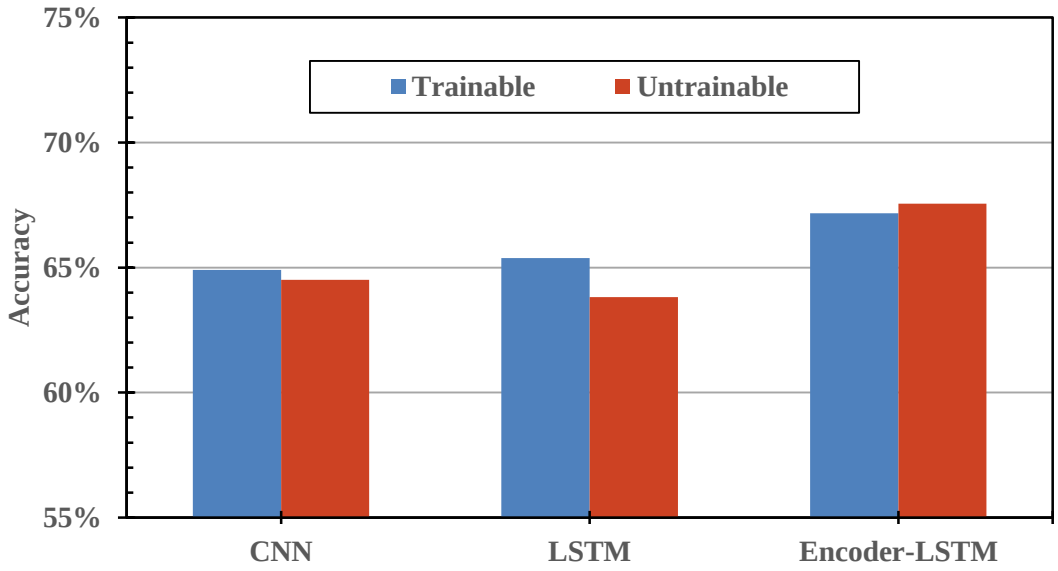

Figure 4. Encoder's effect on Stack Overflow accuracy

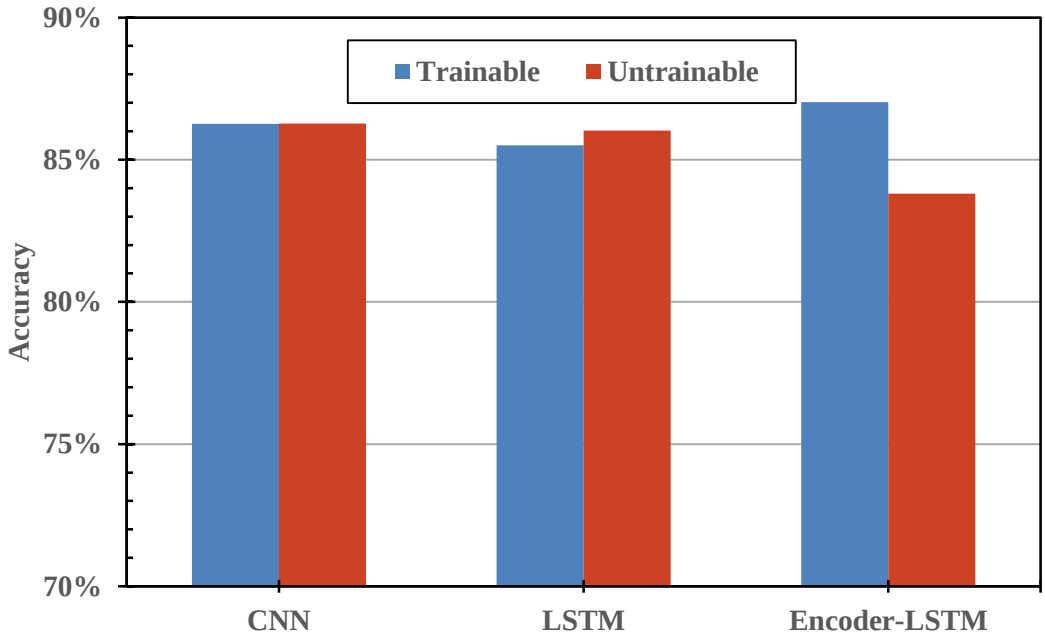

Figure 5. Encoder's effect on IMDb accuracy

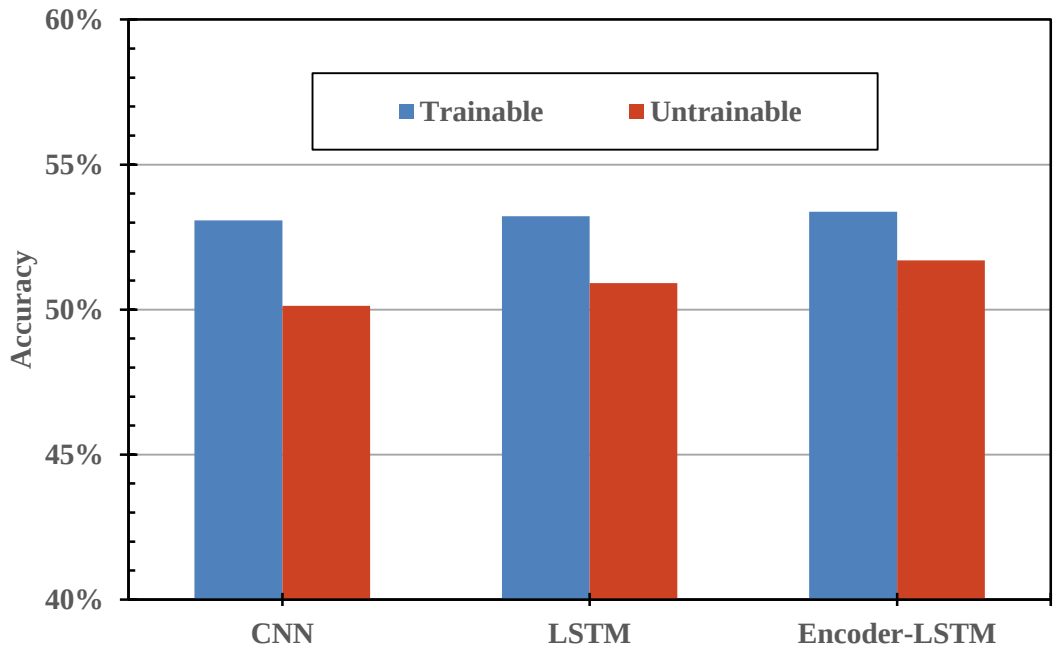

Figure 6. Encoder's effect on Amazon's accuracy

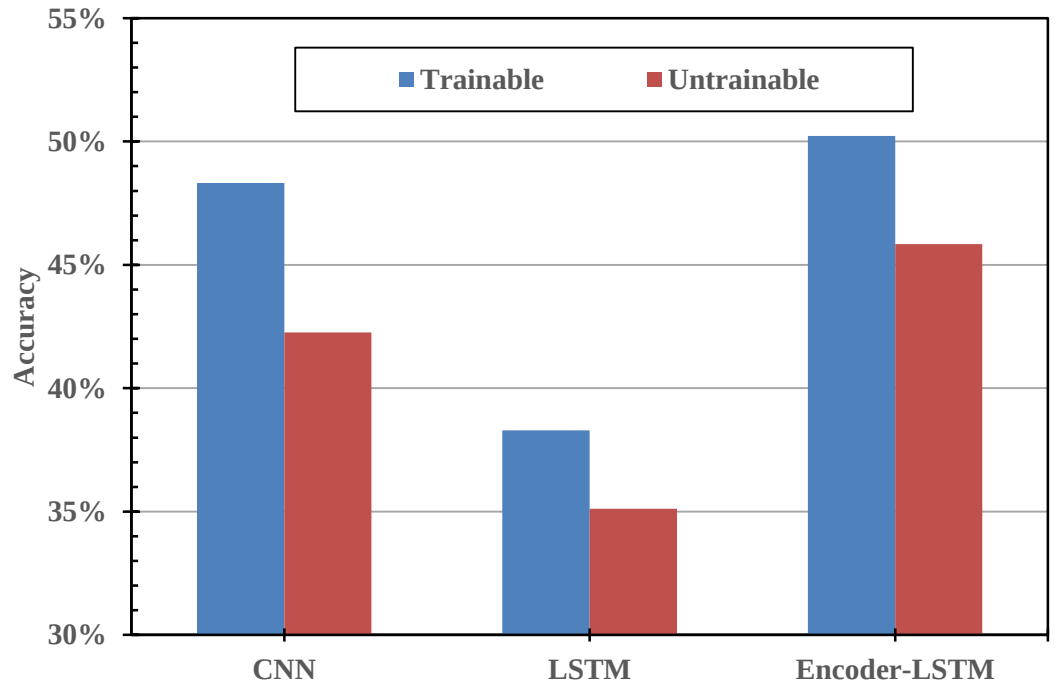

Figure 7. Encoder's effect on 20 News group's accuracy

4.1.2 Encoder Performance on BERT

We used the IMDb and Stack Overflow datasets to test the efficiency of our encoder in the BERT model. We tested all datasets on three models: the original base BERT, the small BERT that we created, and the small BERT with encoder. The test accuracy of these models is reported in Table 5. In the IMDb dataset, the random prediction baseline is 50%, but in the Stack Overflow dataset, it is 5% as we have 20 possible classes.

Data Set	Max Length	Base BERT	Small BERT	Small BERT - encoder
IMDB	512	84.98%	78.16%	76.26%
Stack Overflow	128	10.49%	10.59%	25.20%
Stack Overflow	512	81.08%	19.61%	27.65%

Table 5. The encoder in BERT accuracy comparison

Table 5 and figure 8 shows that introducing the encoder into BERT does not affect the accuracy significantly in IMDb dataset. Certainly, the performance of small BERT compared to original BERT is significantly lower, but that was expected due to the fact that small BERT is both a much smaller model (discussed in the methodology in section 3.2.3) and is pre-trained on a much smaller dataset. The small BERT helped us to check if the encoder can be inserted into BERT or not. In our Stack Overflow dataset, when the maximum length is set to 128, the performance of the original base BERT and our small BERT are identical (see table 5 and figure 9), but we have a significant improvement after introducing our encoder. This is due to the fact that this dataset contains some very long texts. Also, the problem is more complicated than a normal sentiment analysis as we need to find the best tag among 20 possible labels. When the maximum length is set to 512, the

performance of the original base BERT is very high; on the other hand, our encoder still improves the performance of small BERT.

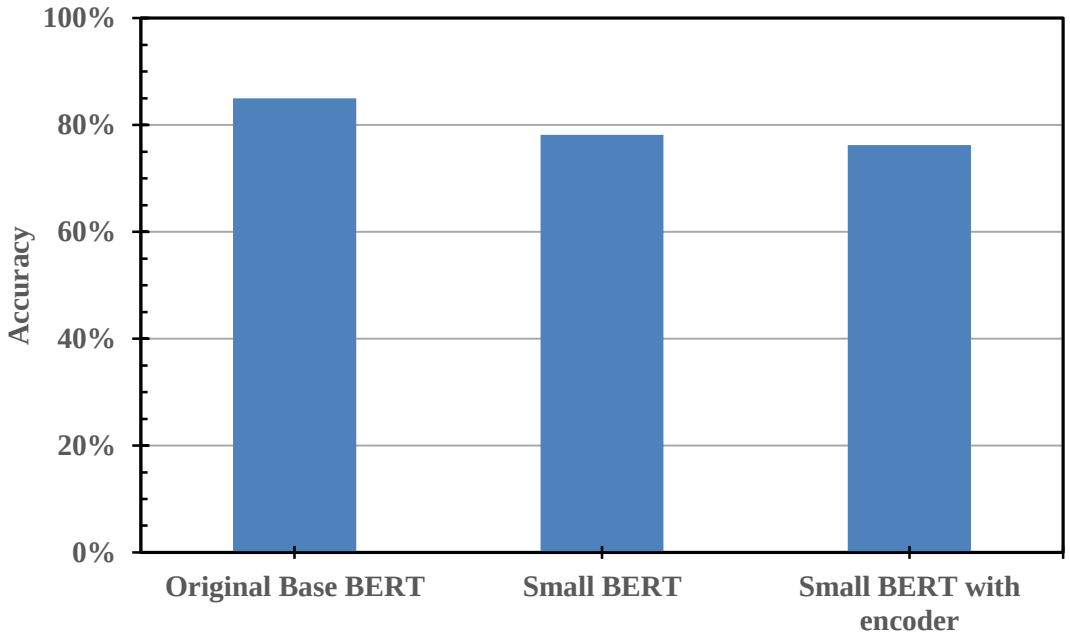

Figure 8. BERT - Encoder performance on IMDb

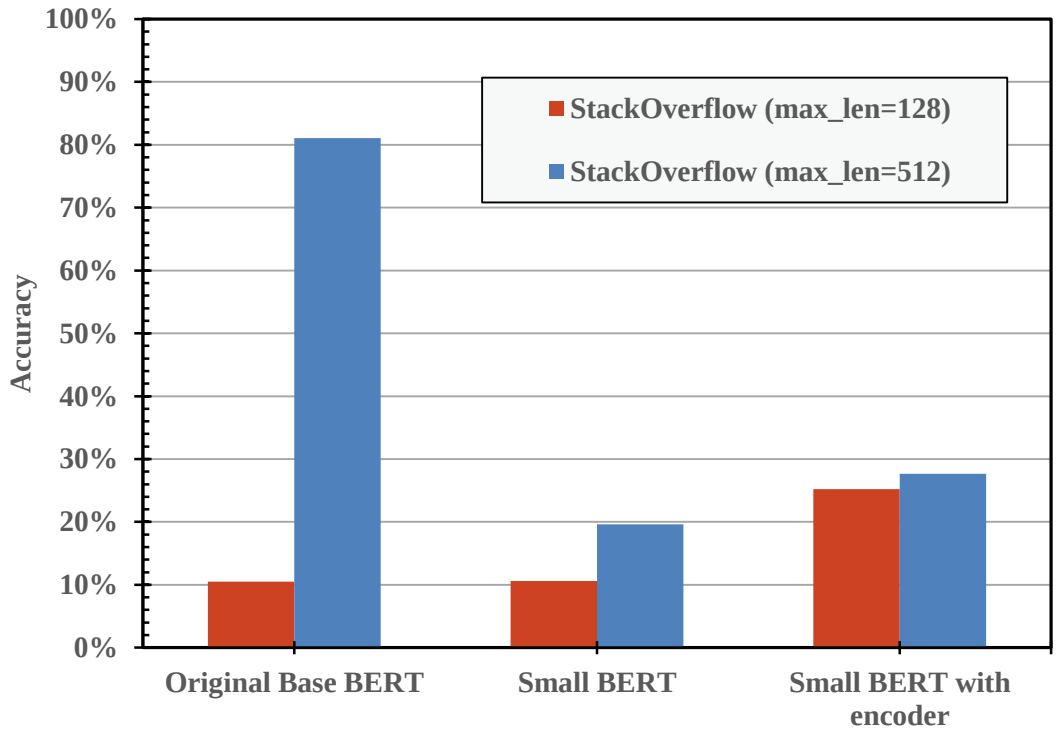

Figure 9. BERT - Encoder performance on Stack overflow

4.1.3 Encoder Result Analysis

According to the experimental results presented, we have shown that our encoder can be used as a tool for any type of NLP model to overcome the maximum sequence length limitation. We have tested our encoder on two popular NLP models, LSTM and BERT, and observed that if the dataset does not contain long texts, inserting the encoder into these models does not affect accuracy. However, if the corpus contains longer textual data, it will improve accuracy significantly. In addition to that, as the number of parameters in these models increases very little (around 1%), it does not affect running time. Each encoder layer reduces the sequence length approximately to half of its original size (the exact value can be calculated by Equation 1); if we need to decrease it further, we can use more encoder layers stacked on top of each other (in the LSTM experiment we used two layers of encoder while for BERT we used just one layer).

4.2 Interpreting CNNs on Textual data results

4.2.1 Performance of Models with Shuffled Filters

After creating three models with the same architecture as our basic model, we set their first convolutional layer weights and make them untrainable. The rest of the model is trained normally for five epochs. Table 6 and figure 10 shows that, when the first layer is assigned randomly and then frozen, accuracy is around 68%, 18% higher than random prediction (as our target variable is binary and balanced). Even when the first layer does not learn anything or contribute to the classification outcomes, the rest of the model learns enough for modest success.

	Train accuracy	Test accuracy	Accuracy Improvement
Basic model	93.24%	83.19%	1.82%
Shuffle within filters	90.18%	81.37%	2.92%
Shuffle across filters	87.64%	78.45%	10.52%
First layer random initialization	81.11%	67.93%	17.93%

Table 6. Comparison of models with shuffled filters.

When we train the first layer normally (in the basic model), it contributes around 15% to overall performance. 2/3 of this contribution belongs to the filter value choices and 1/3 belongs to the order of the sequence in our filters. That is the reason that when we shuffle all 160 (32 filters * 5 units in each filter) weights across all filters, only around 5% of overall accuracy is lost.

Based on these experimental results, we learned that the ordering of each filter is much less important, compared to the crucial filter values found by a model. Besides, we also learned that the relationship between neighboring filter values is not especially strong, since not much performance is lost if the positions of each value are randomized throughout the convolutional layer.

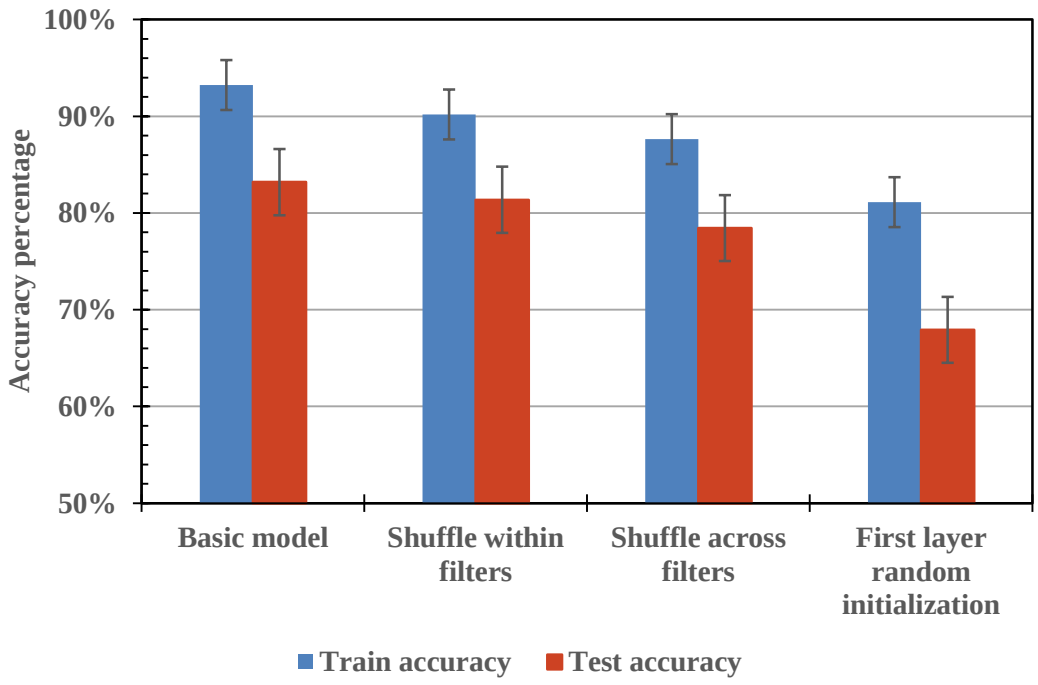

Figure 10. Importance of filters weight vs their position

4.2.2 Clustering on Words and Filters

Clustering is an efficient way to understand patterns within data. To investigate such patterns, we concatenated all of our word embeddings (23,363 × 100) and our filters (160 × 100) to create k clusters. We tested different k between 2 to 2000. The result of the clustering can be seen in Table 7. No matter which size k we choose, there is a single crowded cluster that contains most of the words (e.g., when we produce five clusters, 85 percent of words belong to one cluster). The most crowded cluster contains all 160 filter values. This means that in word embedding space, most of the words are concentrated in a small part of space, and our model chose our filters to be in that space as well. Figure 11 shows how tightly the filter values are concentrated in the main cluster of words (using PCA to represent 100-dimensional embedding vectors in two dimensions).

K	Sum of squared distances	# of elements in most populated cluster	% of elements in most populated cluster
1	-	23363	100.00%
5	218924	19869	85.04%
10	204173	14507	62.09%
20	191238	11871	50.81%
100	155071	8433	36.09%
200	134379	7472	31.98%
500	98158	5210	22.30%
1000	63640	4936	21.13%
2000	32948	2486	10.64%

Table 7. Clustering on Word Embedding

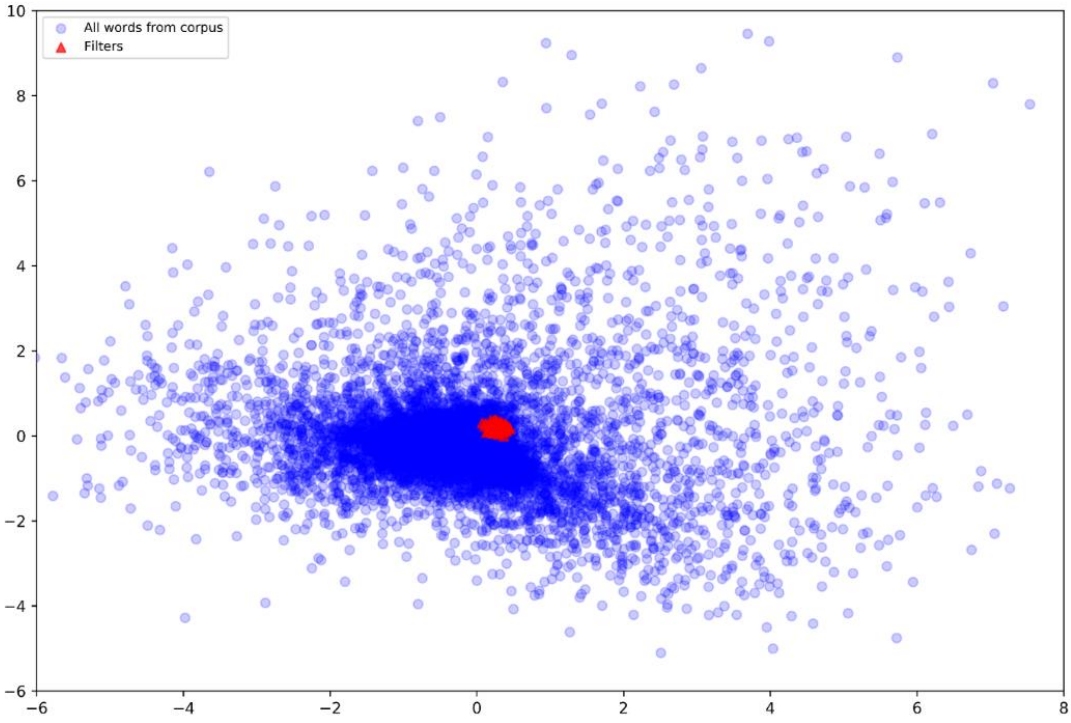

Figure 11. Filters and words distribution

4.2.3 Performance of Models on Most Important Words

After finding the importance rating of every single word in our corpus according to Equation 3, we created six new models. All of them share the same architecture as our basic model shown in Table 2, and each of them is trained only on a subset of the corpus of words. We choose the top n most

important words in our corpus and drop the rest. We train our brand new models on these subsets of words from scratch (weights randomly initiated). Even after dropping 95% of words and training a new model just on 5% of the most important, the performance does not decrease significantly. The performance of these models is presented in Table 8 and figure 14.

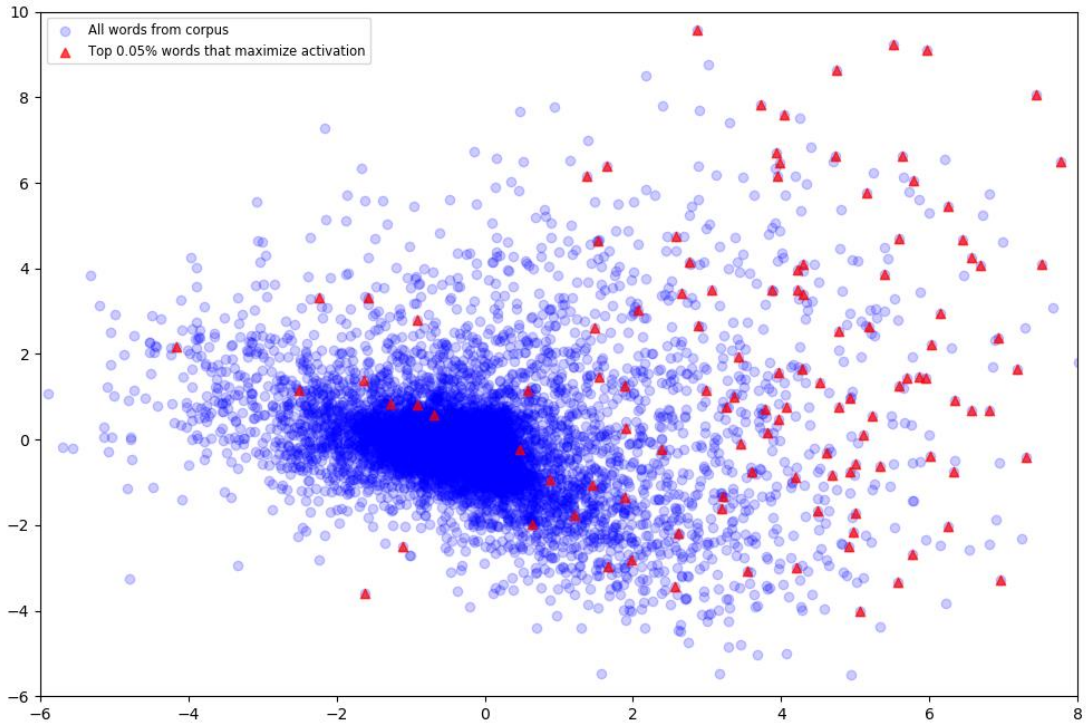

Figure 12. Most important words vs all other words

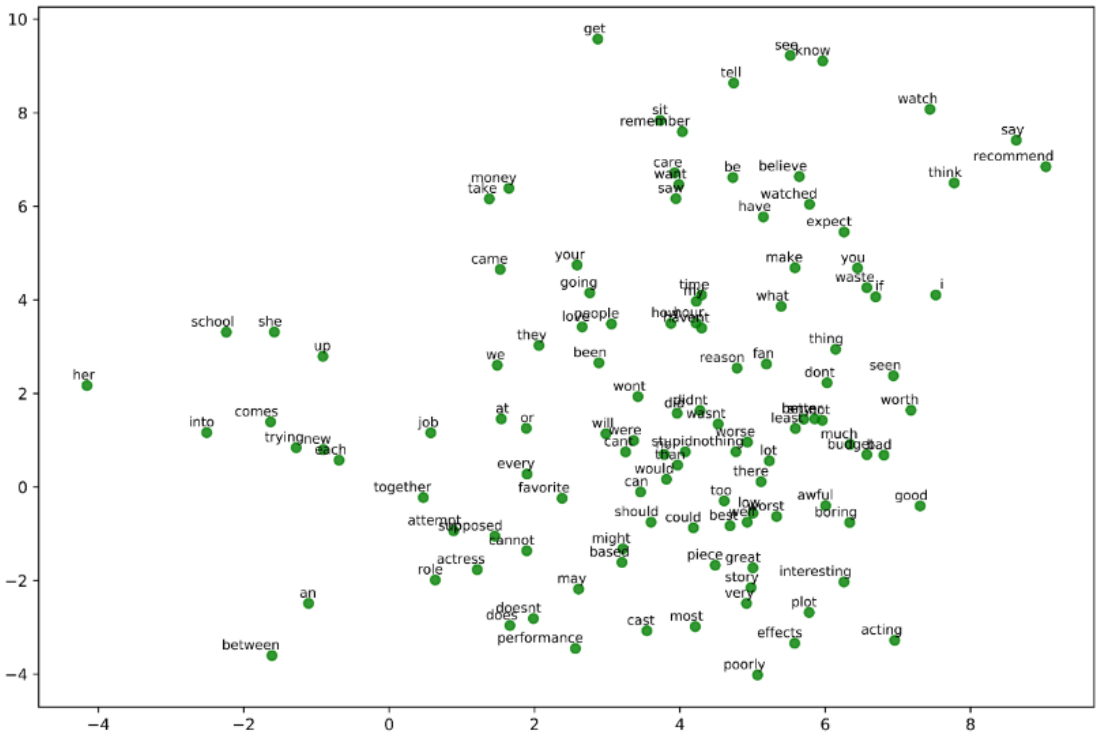

Figure 13. Most important words

word percentage	Word counts	Train accuracy	Test accuracy
100.0%	23,363	93.24%	83.19%
80.0%	18,691	92.71%	83.16%
50.0%	11,682	93.43%	83.14%
10.0%	2,337	92.83%	83.67%
5.0%	1,169	92.34%	82.53%
1.0%	234	87.02%	78.16%
0.5%	117	84.75%	74.62%

Table 8. Training models on most important words

Although the model does start to lose information, and thus classification performance, once the corpus is reduced to 1% of its original size or below, performance remains strong when using only 5% of available words. Our final set of experiments focus on this behavior. In figures 12 and 13, we used PCA to reduce the dimensionality of the word embeddings from 100 to 2 in order to

represent words in 2-D space and show how the most important words form a reasonable coverage of the space. Figure 12 shows the important words as a fraction of all of the words, and figure 13 shows which words were found to represent the embedding space. The figures are separated for clarity. To compare, we established three baseline models: one which uses all words in our corpus, one that also uses 5%, but selected randomly rather than via Equation 3, and one that uses all except the most important 5% of words. Results are shown in Table 9. Our selected words perform much better than randomly choosing the 5% of words (a 20% increase in test accuracy).

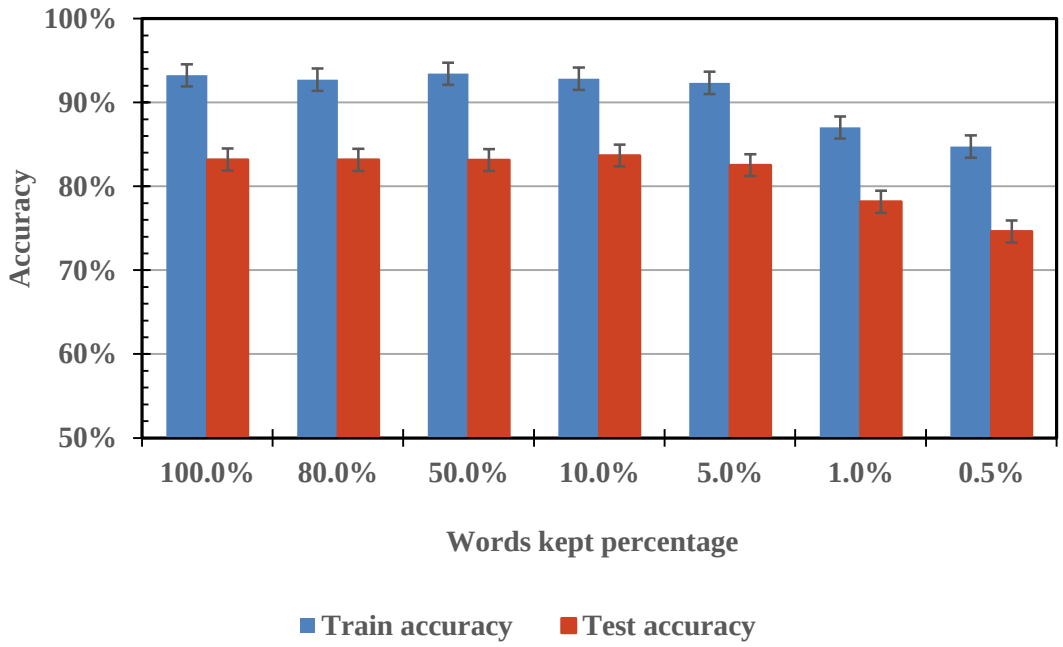

Figure 14. Training models on most important words

Table 9 and Figure 15 also show that restricting the model to the most important words results in much faster performance than the model using every available word in the corpus. Whether measured in epochs or seconds, the restricted model is more than twice as fast at learning. Unsurprisingly, speed is equivalent between both models which use only 5% of the data, but one

that uses the important words performs much better. If the best 5% of words identified via Equation 1 are eliminated, the model has the worst of both worlds and is neither fast nor accurate. The model architecture and hyper-parameters seem to have an insignificant effect on the importance rate of words, but it can be studied in more depth in future works.

Model name	% words used	Word count	Train accuracy	Test accuracy	Average epoch training time (seconds)	Number of parameters
Most important words	5%	1,169	93.67%	84.42%	36.3062	256,697
All words	100%	23,363	96.87%	85.33%	81.6395	2,476,097
Random words	5%	1,169	70.68%	64.29%	36.1543	256,697
All except important	95%	22,194	83.86%	74.52%	79.6917	2,359,197

Table 9. Comparing new models to baseline models

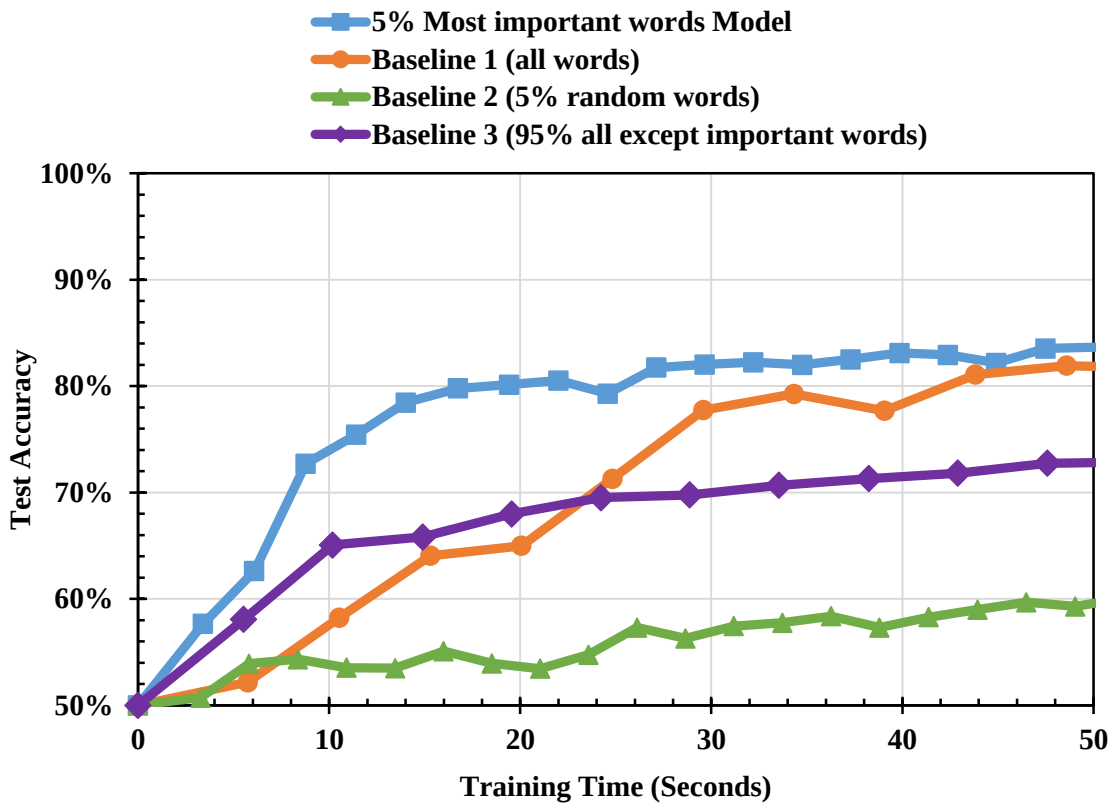

Figure 15. Comparing our model with three baseline models based on testing accuracy and training

4.3 Adversarial Attacks on Textual Data Results

In the experiments, t is set to four different values: 0, 10, 20, 40. We also included t = 0 which shows the baseline model without any inserted attack, so the effectiveness of each adversarial attack method can be seen. We tested our three models (CNN, LSTM, and transformer) on two datasets (IMDb and Stack Overflow) with 3 different techniques of attack: WordBug, Activation Maximization, and random. In the random case, t words are chosen at random to attack. In interpreting the attack performance, a lower test accuracy means a more effective attack.

4.3.1 Evaluation of Adversarial Attacks on IMDb

Figures 16-18 show that, for the IMDb dataset in most cases, the WordBug technique works slightly better than Activation Maximization but not significantly. On average, the WordBug attacks reduce the models' accuracies by less than 1% more than the AM attacks. Our results when using WordBug to attack the IMDb dataset differs from Gao's original results [42]. Ours is a precise implementation of the WordBug attack, so the only potential reasons for this variation might be different preprocessing steps or different LSTM architectural hyper-parameters. However, for the purposes of comparing WordBug and Activation Maximization, this difference is immaterial. The same preprocessing steps and model hyper-parameters are used to compare the approaches fairly.

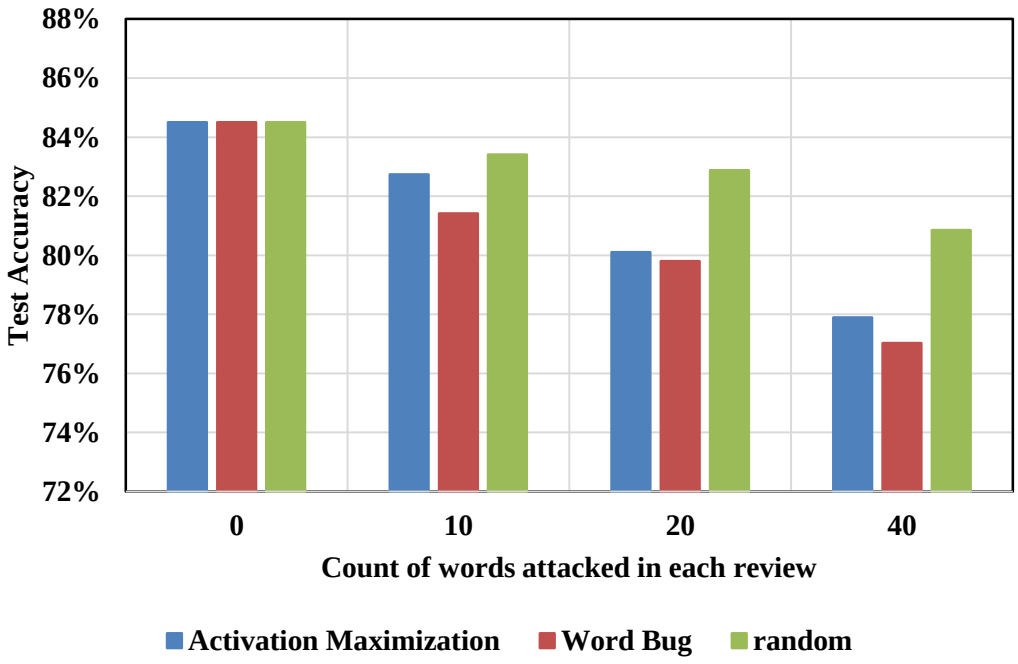

Figure 16. Evaluation of attack methods in CNN on IMDB dataset

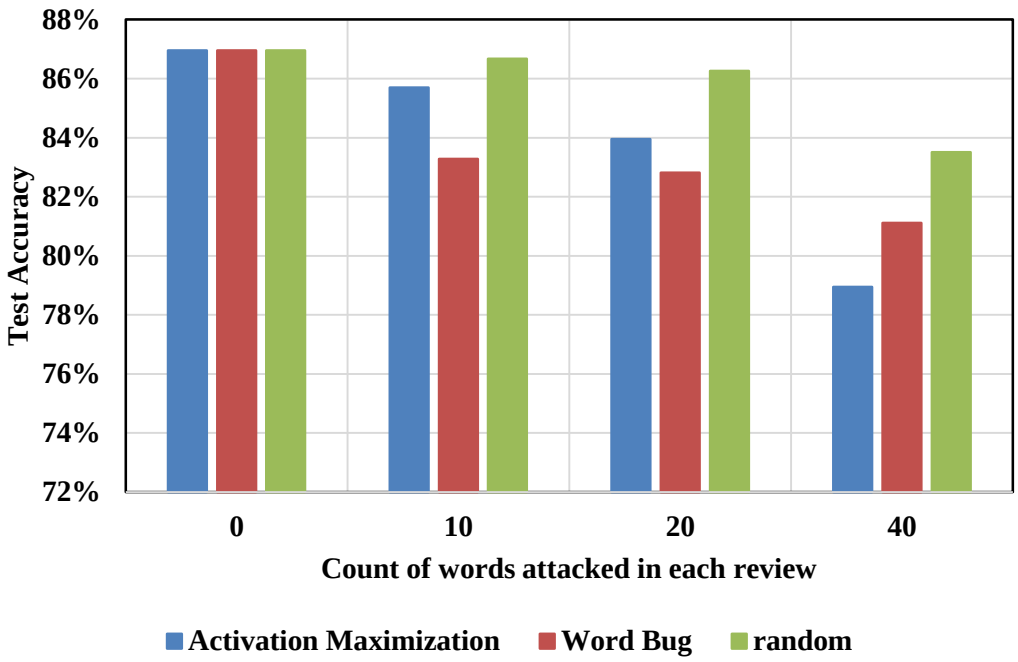

Figure 17. Evaluation of attack methods in LSTM on IMDB dataset

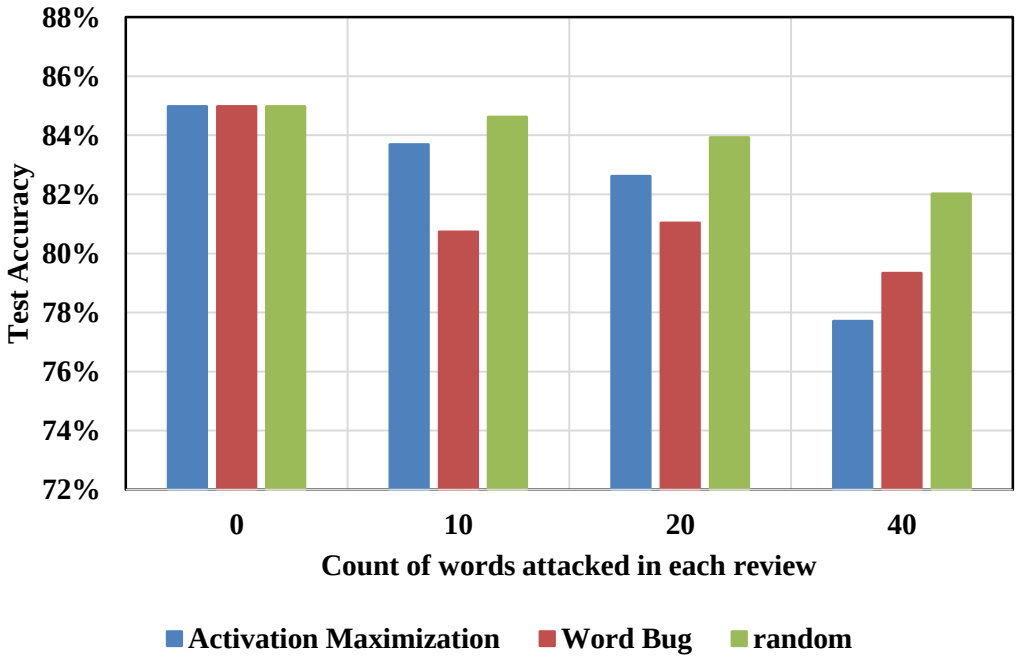

Figure 18. Evaluation of attack methods in transformer on IMDB dataset

4.3.2 Evaluation of Adversarial Attacks on Stack Overflow

The Stack Overflow dataset task is much more complex than the naive Sentiment Analysis task in IMDb. The task requires classifying data into 20 possible classes or tags, and the texts are much longer. As a result, Figures 19-21 show that the Activation Maximization method is performing significantly better than WordBug. On average, the Activation Maximization attack leads to a 4.6% lower performance than WordBug's.

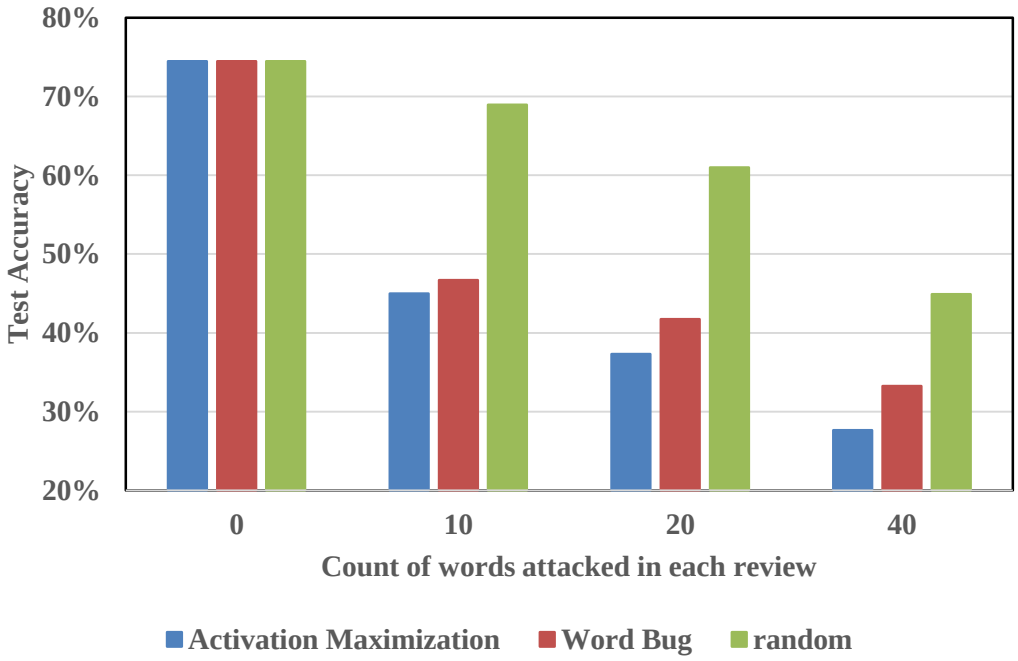

Figure 19. Evaluation of attack methods in CNN on Stack Overflow dataset

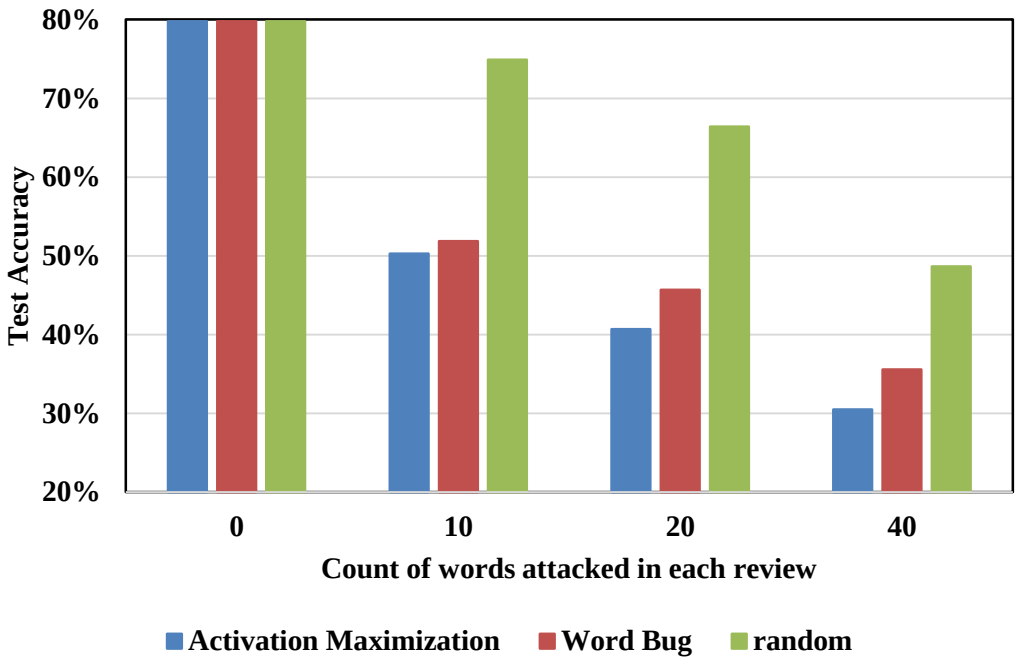

Figure 20. Evaluation of attack methods in LSTM on Stack Overflow dataset

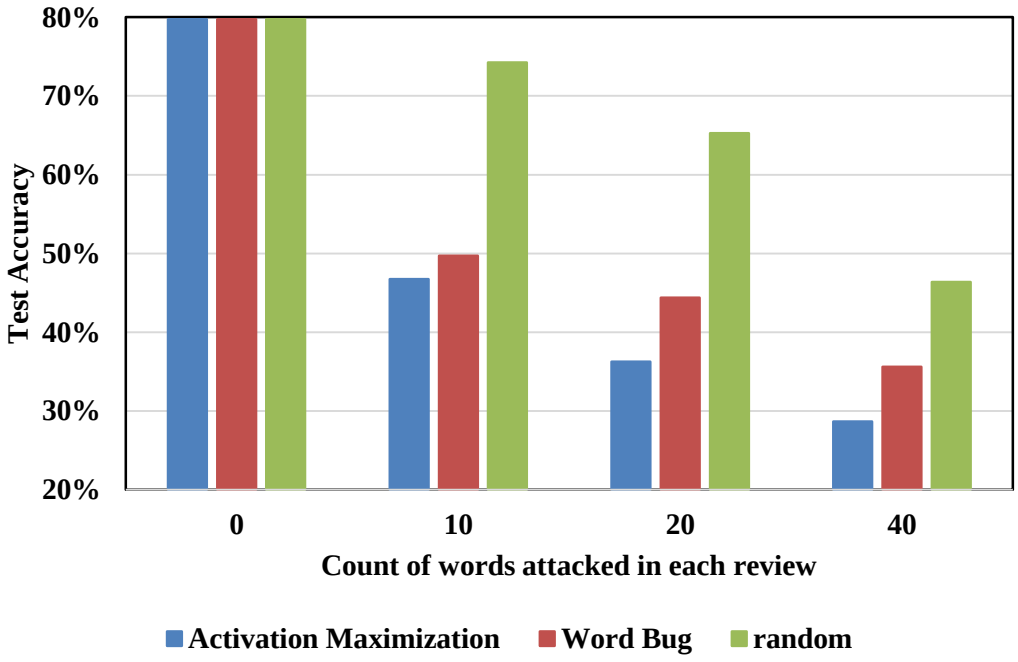

Figure 21. Evaluation of attack methods in transformer on Stack Overflow dataset

4.3.3 Transfer Learning from IMDB to Stack Overflow

After comparing the performance of WordBug and Activation Maximization, we show that in the AM method, we can train the attack on a subset of data and then use it to attack newer never-seen-before data from the same distribution and context. This performs equally well or better than WordBug, which has to study every single input to be able to attack it.

We also wanted to show whether the same transfer learning logic can apply more generally, to new data but from a different context. In order to check this, we trained our Activation Maximization attacker on the IMDb dataset and used it on Stack Overflow. Figure 22 shows that the experiment does not support this, as the performance is even worse than a random attack.

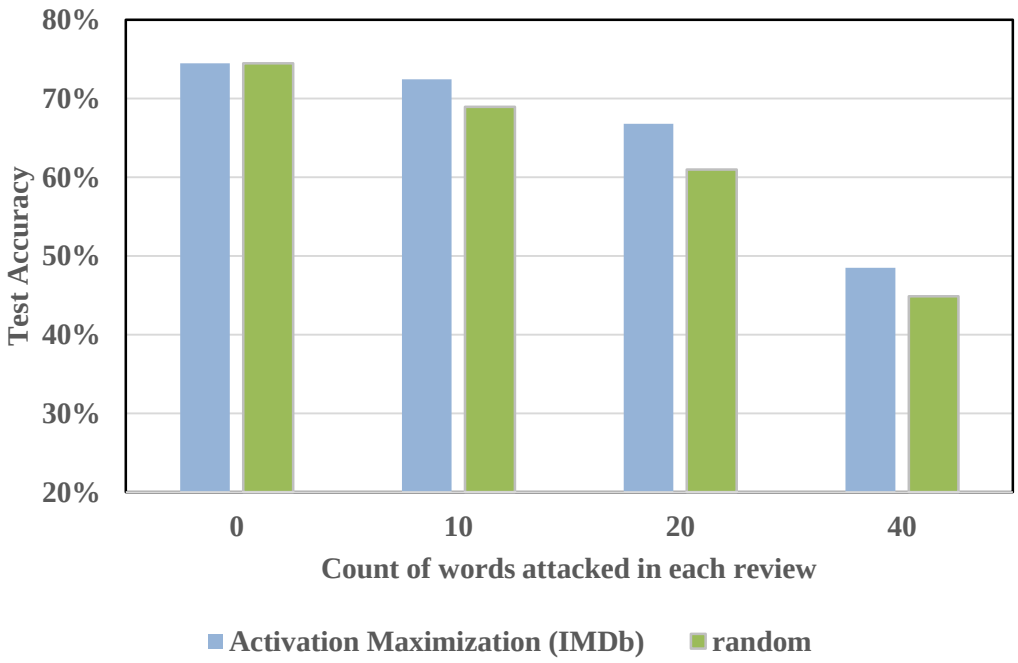

Figure 22. Evaluation of using importance rate extracted from IMDb dataset and used to attack Stack Overflow dataset.

4.3.4 Adversarial attack Run Time Comparison

In addition to their effect on a model's accuracy, another important factor in adversarial attacks is speed. We applied the attacks on corpora with different sizes varying from 20 to 40,000 data elements and measured the time needed for each method to train and choose the most important words to target.

Figures 23 and 24 shows that Activation Maximization is much faster than WordBug in linear and logarithmic format in the respective order. This is to be expected, as WordBug must analyze each word in each data element one by one, while the Activation Maximization approach just trains a CNN model for a couple of epochs and uses the resultant filters to identify globally important words. With a corpus size of 40,000, Activation Maximization is 39 times faster than WordBug.

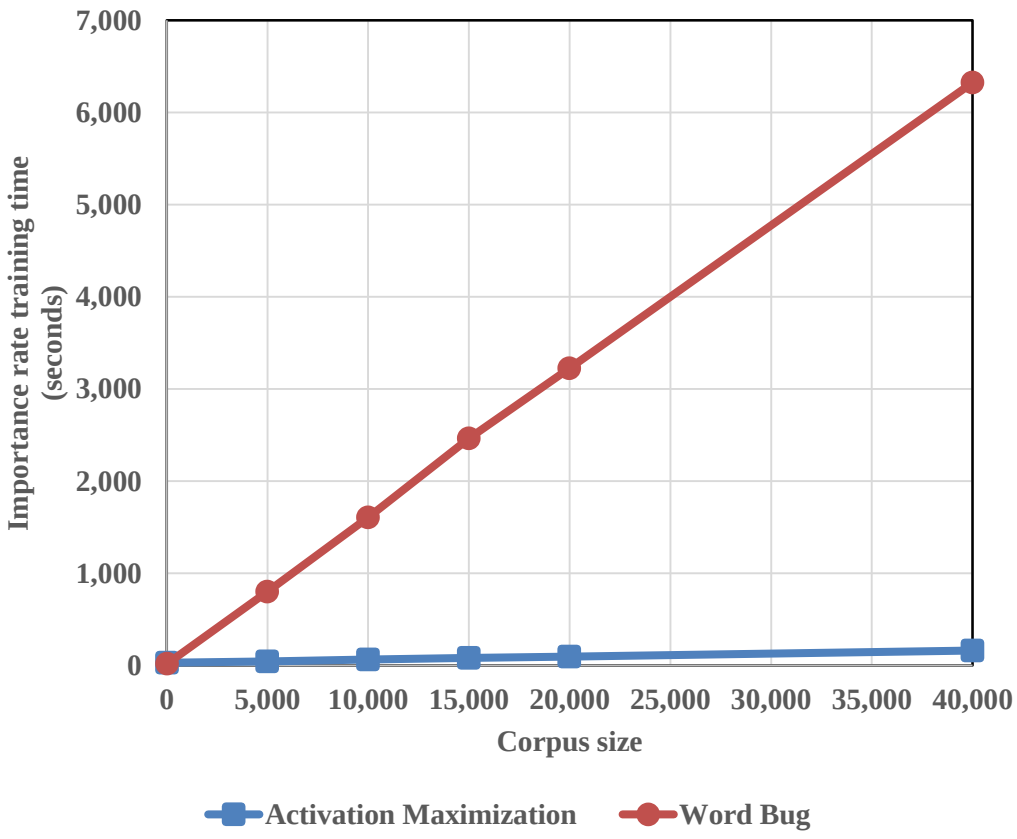

Figure 23. Runtime Comparison of WordBug and Activation Maximization (Linear)

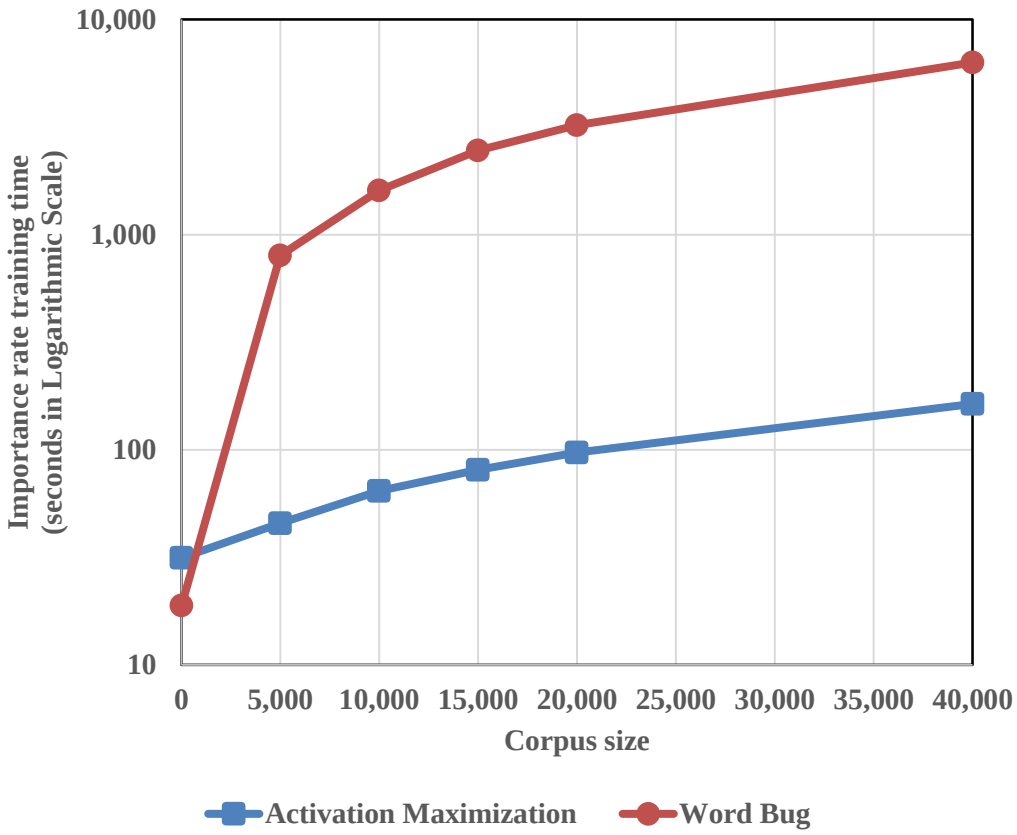

Figure 24. Runtime Comparison of WordBug and Activation Maximization (Logarithmic)

4.3.5 Adversarial Attacks Results Analysis

We have demonstrated that the Activation Maximization approach is not only much faster than WordBug but that it is equally good or better in its attack performance (depending on the task). In addition, Activation Maximization can be used to attack new input without any need for further training, whereas WordBug must be trained on each sentence prior to a successful attack. However, we observed that this ability of Activation Maximization only applies to data from the same context and distribution.

Both of these models are black-box attacks and do not have access to the details of the attacked model. In WordBug, the extracted information about the most important words is local to each specific input, whereas in Activation Maximization, the attacker finds the most important words to target based on a global evaluation of a large corpus.

4.4 Deep NLP Explainer results

In order to test our hypothesis on both of our datasets, and compare the performance of each of the two importance rates extracted, we designed new models that were just trained on the most important words based on three different algorithms: **Activation Maximization**, **Prediction Slope**, **Random**. In the random technique, words are chosen randomly as a naive baseline for comparison with our two other models. We also compare them against the **Base Model** that uses all 100% of the words. The final model is called **Hybrid**, and it averages the importance rates for each word generated by each of the two techniques.

We created a threshold that identified the percentage of the most important words that our models would train on. We tested different threshold values: 10%, 5%, 2%, 1%, 0.5%.

4.4.1 Comparing importance rates on the IMDb dataset

In our IMDb dataset, as it is a sentiment analysis problem with a binary target value, the prediction accuracy starts from 50% and the baseline accuracy with access to all words was 84%. Results are shown in figure 25.

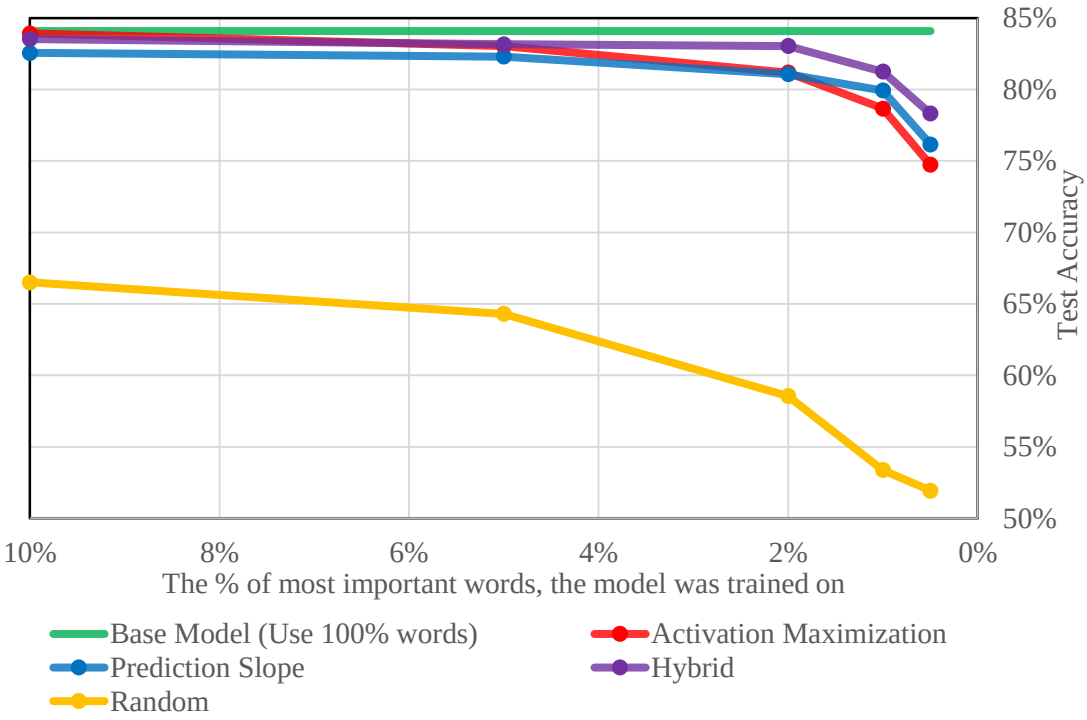

Figure 25. Comparison of different importance rate techniques on IMDb dataset

Both models are interchangeable, with no significant difference in their performance. In addition, both perform superbly while using just 2% of the data. Accuracies are very similar to the base model while they are much faster. The random model performs poorly as expected.

4.4.2 Comparing importance rates on the Stack Overflow dataset

The Stack Overflow dataset presents a multinomial task with 20 possible classes or tags. Accuracy therefore starts at 5%, and the accuracy of the base model with access to all words is 74%.

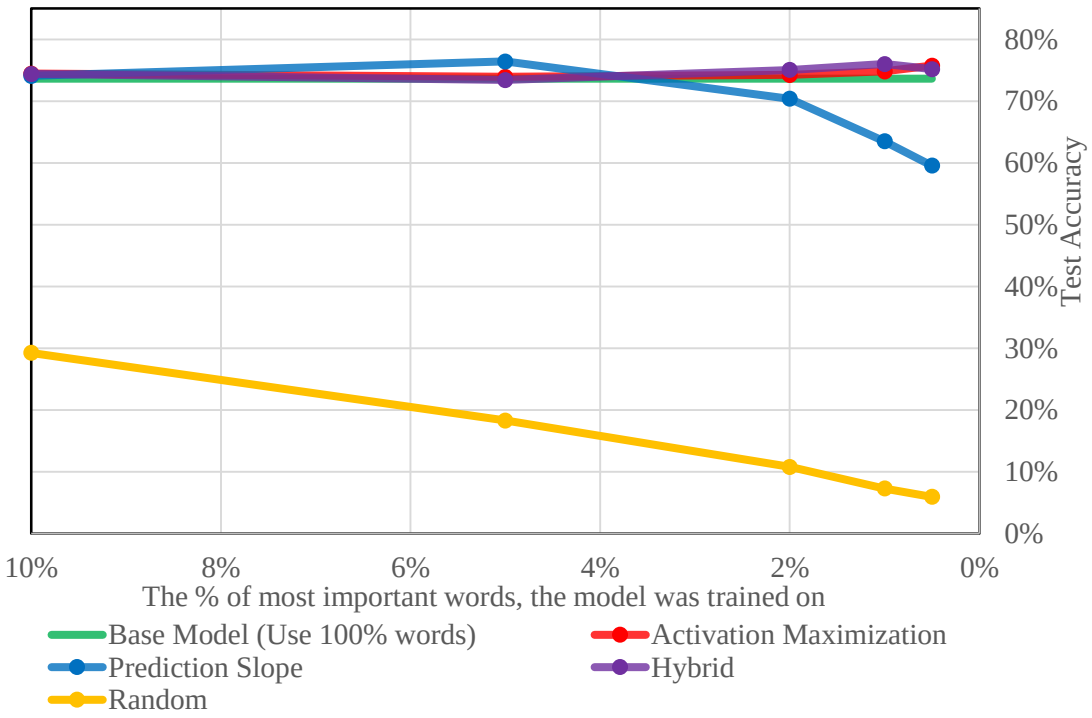

Figure 26. Comparison of different importance rate techniques on Stack Overflow dataset

Figure 26 shows that both models are still very similar, although the prediction slope technique has lower performance at thresholds smaller than 2%. The activation maximization model, however, has very good performance -- even slightly better than the baseline model -- even when training on only 0.5% of words. The model focuses on critical keywords, and it turns out that in this task, they are extremely predictive. Both models still perform well overall, even when they have access to a small subset of the data. Again, they are much faster than the baseline model. Also as in the previous experiment, the random model performs weakly as expected.

4.4.3 Analysis of the result

It was observed that both techniques have quite similar performance (They performed equally in the IMDb dataset, Activation Maximization was slightly better in the Stack Overflow dataset), and both are much faster than the base model as they are just using a small subset of the input. However,

the Activation Maximization can just be applied on CNN models and is very dependent on the model architecture, while the Prediction Slope can be applied on any type of NLP model architecture (In our case it was applied on a transformer model). This is very beneficial to have a tool that can analyze the model independent of the inner architecture or details and gives us insights from the model's global logic.

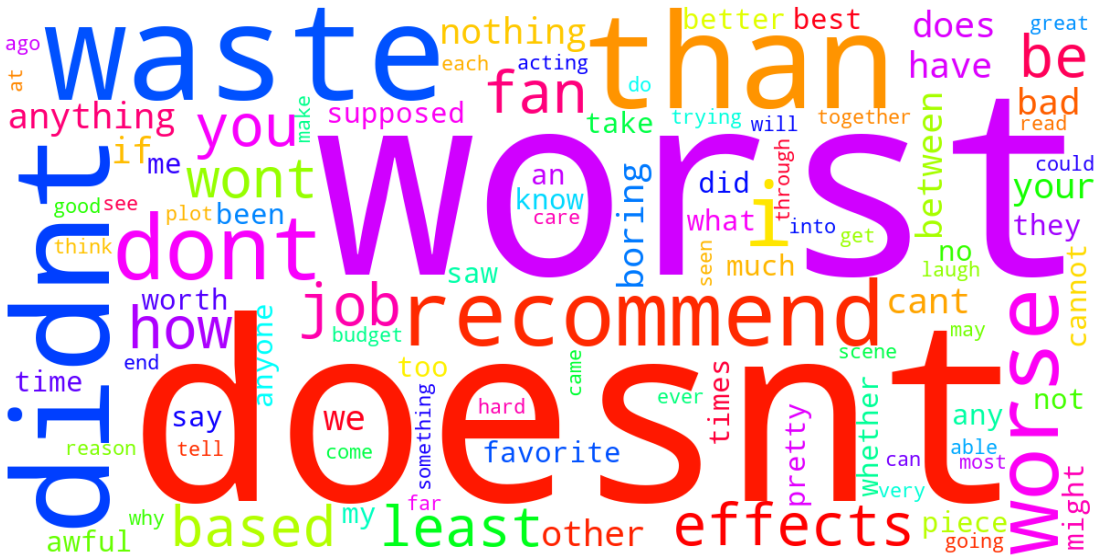

Figure 27. Wordcloud of the top 100 most important words in IMDb dataset by Activation Maximization technique

Figure 28. Wordcloud of the top 100 most important words in IMDb dataset by Prediction Slope technique

These techniques can be used to generate insights to improve future models, and we can also use them to visualize the most important words to a model to make it more explainable and understandable. Figures 27 and 28 shows the top 100 most important words extracted from both techniques in the IMDb dataset. The size of the word represents its importance rate. The most important words according to the activation maximization technique exhibited higher document frequencies.

CHAPTER V

CONCLUSION

The field of Artificial Intelligence, machine learning and deep learning has long focused on how to improve the performance of our models, identify useful cost functions to optimize, and thereby increase prediction accuracy. However, now that human-level performance has been reached or exceeded in many domains using deep learning models, we must investigate other important aspects of our models, such as explainability and interpretability or overcoming their limitations.

LSTMs were designed to overcome some of the limitations that RNNs encounter with time-series data, and they usually succeed in outperforming the older architecture. BERT achieved a huge improvement in all types of textual data problems by using transfer learning. These two are among the best models designed for NLP tasks, but both share a maximum sequence length limitation. In other words, neither is capable of appropriately processing long texts. We have devised a technique to overcome this limitation by creating an encoder layer that will reduce the dimensionality of the input. Results showed that it provides us with the ability to process longer texts and improve accuracy.

Explainable Artificial Intelligence is another field that needs further improvement. We would like to be able to trust artificial intelligence and rely on it even in critical situations. Furthermore, beyond increasing our trust in a model's decision-making, a model's interpretability helps us to understand its reasoning, and it can help us to find its weaknesses and strengths. We can learn from the models strengths and inject them into new models, and we can overcome their weak points by removing their bias. In this dissertation, we investigated the logic behind the decisions of a 1-D CNN model by studying and analyzing its filter values and determining the relative importance of the unique words within a corpus dictionary. We were able to use the insights from this investigation to identify a small subset of important words, improving the learning performance of the training process by better than double. This contribution can be used to deepen our study of the ability of our models to identify important words. By performing sensitivity analysis, alternately verifying or denying the model's access to words it deems vital, we will hopefully be able to facilitate the transfer of linguistic insights between human experts and learning systems.

We also devised a new adversarial attack method for textual data. Our new Activation Maximization adversarial attack has many benefits: it is significantly faster, its performance is equal to or better than WordBug (one of the most recent techniques), and it can be used on new inputs (from the same context) without any further training. Our attack method is a black box attack which means it does not need access to the structure or weights of an attacked model. Developing these kinds of effective and efficient attacks will enable us to evaluate new models to find their blind spots.

The last contribution of this dissertation was to generate a new method for explaining NLP models' logic. Our experiments show that our method is as accurate as previous one, while it is much more generalized. Our technique is not dependent on the NLP architecture and type and can be applied to any NLP model and task.

REFERENCES

[1] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” Nature. 2015, doi: 10.1038/nature14539.

[2] M. H. Hassoun, “Fundamentals of Artificial Neural Networks,” Proc. IEEE, 2005, doi: 10.1109/jproc.1996.503146.

[3] M. Chen, S. Mao, and Y. Liu, “Big data: A survey,” 2014, doi: 10.1007/s11036-013-0489-0.

[4] J. Hirschberg and C. D. Manning, “Advances in natural language processing,” Science. 2015, doi: 10.1126/science.aaa8685.

[5] Y. Goldberg, “A primer on neural network models for natural language processing,” J. Artif. Intell. Res., 2016, doi: 10.1613/jair.4992.

[6] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, “Natural language processing (almost) from scratch,” J. Mach. Learn. Res., 2011.

[7] B. Liu, “Sentiment analysis and subjectivity,” in Handbook of Natural Language Processing, Second Edition, 2010.

[8] S. Bird, S. Bird, and E. Loper, “NLTK : The natural language toolkit NLTK : The Natural Language Toolkit,” Proc. ACL-02 Work. Eff. tools Methodol. Teach. Nat. Lang. Process. Comput. Linguist. 1, 2016.

[9] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations ofwords and phrases and their compositionality,” 2013.

[10] J. Pennington, R. Socher, and C. D. Manning, “GloVe: Global vectors for word representation,” 2014, doi: 10.3115/v1/d14-1162.

[11] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, 1997, doi: 10.1162/neco.1997.9.8.1735.

[12] K. Cho et al., “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” 2014, doi: 10.3115/v1/d14-1179.

[13] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, “A convolutional neural network for modelling sentences,” 2014, doi: 10.3115/v1/p14-1062.

[14] Y. Kim, “Convolutional neural networks for sentence classification,” 2014, doi: 10.3115/v1/d14-1181.

[15] H. T. Le, C. Cerisara, and A. Denis, “Do Convolutional Networks need to be Deep for Text Classification ?,” arXiv. 2017.

[16] Z. Wood-Doughty, N. Andrews, and M. Dredze, “Convolutions Are All You Need (For Classifying Character Sequences),” 2019, doi: 10.18653/v1/w18-6127.

[17] W. Yin, K. Kann, M. Yu, and H. Schütze, “Comparative study of CNN and RNN for natural language processing,” arXiv. 2017.

[18] A. Vaswani et al., “Attention is all you need,” 2017.

[19] I. Beltagy, M. E. Peters, and A. Cohan, “Longformer: The Long-Document Transformer,” arXiv. 2020.

[20] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” 2019.

[21] Y. Liu et al., “RoBERTa: A robustly optimized BERT pretraining approach,” arXiv. 2019.

[22] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, “XLNet: Generalized autoregressive pretraining for language understanding,” arXiv. 2019.

[23] T. B. Brown et al., “Language models are few-shot learners,” arXiv. 2020.

[24] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G. Z. Yang, “XAI-Explainable artificial intelligence,” Sci. Robot., 2019, doi: 10.1126/scirobotics.aay7120.

[25] M. Du, N. Liu, and X. Hu, “Techniques for interpretable machine learning,” arXiv. 2018.

[26] A. Adadi and M. Berrada, “Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI),” IEEE Access, 2018, doi: 10.1109/ACCESS.2018.2870052.

[27] S. Bach, A. Binder, G. Montavon, F. Klauschen, K. R. Müller, and W. Samek, “On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation,” PLoS One, 2015, doi: 10.1371/journal.pone.0130140.

[28] G. Montavon, W. Samek, and K. R. Müller, “Methods for interpreting and understanding deep neural networks,” Digital Signal Processing: A Review Journal. 2018, doi: 10.1016/j.dsp.2017.10.011.

[29] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why should i trust you?' Explaining the predictions of any classifier," 2016, doi: 10.1145/2939672.2939778.

[30] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," 2014.

[31] F. Wang et al., "Residual attention network for image classification," 2017, doi: 10.1109/CVPR.2017.683.

[32] J. Vig, "A multiscale visualization of attention in the transformer model," 2019, doi: 10.18653/v1/p19-3007.

[33] L. Arras, F. Horn, G. Montavon, K. R. Müller, and W. Samek, "'What is relevant in a text document?': An interpretable machine learning approach," PLoS One, 2017, doi: 10.1371/journal.pone.0181142.

[34] J. Li, X. Chen, E. Hovy, and D. Jurafsky, "Visualizing and understanding neural models in NLP," 2016, doi: 10.18653/v1/n16-1082.

[35] M. Rajwadi, C. Glackin, J. Wall, G. Chollet, and N. Cannings, "Explaining sentiment classification," 2019, doi: 10.21437/Interspeech.2019-2743.

[36] N. Akhtar and A. Mian, "Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey," IEEE Access. 2018, doi: 10.1109/ACCESS.2018.2807385.

[37] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," arXiv. 2017.

[38] A. Mądry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," arXiv. 2017.

[39] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-gan: Protecting classifiers against adversarial attacks using generative models," arXiv. 2018.

[40] E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh, "Universal Adversarial Triggers for Attacking and Analyzing NLP," 2019, doi: 10.18653/v1/d19-1221.

[41] W. E. Zhang, Q. Z. Sheng, A. Alhazmi, and C. Li, "Adversarial attacks on deep learning models in natural language processing: A survey," arXiv. 2019.

[42] J. Gao, J. Lanchantin, M. Lou Soffa, and Y. Qi, "Black-box generation of adversarial text sequences to evade deep learning classifiers," 2018, doi: 10.1109/SPW.2018.00016.

[43] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," 2011.

[44] T. Mitchell, "The 20 newsgroup dataset." 1999.

[45] R. Marzban and C. Crick, “Lifting Sequence Length Limitations of NLP Models using Autoencoders,” 2021, In Proceedings of the 10th International Conference on Pattern Recognition Applications and Methods - Volume 1: ICPRAM, ISBN 978-989-758-486-2, pages 228-235, doi: 10.5220/0010239502280235.

[46] R. Marzban and C. Crick, “Interpreting Convolutional Networks Trained on Textual Data,” 2021, In Proceedings of the 10th International Conference on Pattern Recognition Applications and Methods - Volume 1: ICPRAM, ISBN 978-989-758-486-2, pages 196-203, doi: 10.5220/0010205901960203.

APPENDICES

There is no appendix for this dissertation.

VITA

Reza Marzban

Candidate for the Degree of

Doctor of Philosophy

Thesis: INTERPRETING NATURAL LANGUAGE PROCESSING (NLP) MODELS AND LIFTING THEIR LIMITATIONS

Major Field: Computer Science

Biographical:

Education:

Completed the requirements for the Doctor of Philosophy in Computer Science at Oklahoma State University, Stillwater, Oklahoma in July, 2021.

Completed the requirements for the Bachelor of Science in Information Technology Engineering at University of Applied Science and Technology, Tehran, Iran in 2017.

Experience:

Graduate Teaching Assistant in Computer Science Department at Oklahoma State University, Stillwater, OK from August 2017 to May 2021.

Data Analyst Intern at Amerihome Mortgage, Thousand Oaks, CA from May 2020 to August 2020.

Data Science Intern at National Oilwell Varco, Houston, TX from June 2019 to August 2019.

Senior Technical Advisor at Group Four International General Trading Co., Tehran, Iran from June 2007 to July 2017.