Master Thesis

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Adversarial Attacks on Text Classifiers

Bc. David Herel

Supervisor: Ing. Tomáš Mikolov, Ph.D.
Field of study: Open Informatics
Subfield: Cybersecurity
May 2022

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Herel David** Personal ID number: **474491**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Cyber Security**

II. Master’s thesis details

Master’s thesis title in English:

Adversarial attacks on text classifiers

Master’s thesis title in Czech:

Adversarialní útoky na klasifikátory textu

Guidelines:

Adversarial attacks on image classifiers is a well established research topic. On the other hand, attacks on text classifiers are more recent and less well developed. One of the problems is that some of these attacks completely change semantics of the text. Thus, it is needed to develop a new, more realistic approach, and a new metric that will be useful for distinguishing realistic attacks on classifiers from techniques that can rather be seen as minimal edits of text.
1. Get familiar with the state of the art text classifiers and their vulnerabilities to adversarial attacks
2. Survey existing attacks on these classifiers and analyze them
3. Provide a metric to compare sucesses of these attacks
4. Propose an adversarial attack on text classifiers that will be less problematic than the existing ones

Bibliography / sources:

1. Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. 2019. Adversarial exam- ples: Attacks and defenses for deep learning. IEEE transactions on neural networks and learning systems, 30(9):2805–2824.
2. John X. Morris, Eli Lifland,Jack Lanchantin, Yangfeng Ji, Yanjun Qi, 2020, Reevaluating Adversarial Examples in Natural Language, arXiv preprint arXiv:2004.14174
3. Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limti- aco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. arXiv preprint arXiv:1803.11175
4. Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020. Bert-attack: Adversarial attack against bert using bert. In Proc. of EMNLP
5. Zhengli Zhao, Dheeru Dua, and Sameer Singh. 2018. Generating natural adversarial examples. In Proceedings of ICLR

Name and workplace of master’s thesis supervisor:

Ing. Tomáš Mikolov, Ph.D. RICAIP CIIRC

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **02.02.2022** Deadline for master's thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Tomáš Mikolov, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

. _____ Date of assignment receipt

_____ Student's signature

Acknowledgements

I would like to thank my supervisor, Ing. Tomáš Mikolov, Ph.D., for his invaluable advices and excellent guidance not only during writing this thesis, but also during previous years of my studies. I would also like to express my sincere appreciation to my family and also to Daniela Hradilová for their tremendous support.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information use within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.
Prague, 16. May 2022

Abstract

Nowadays, some of the most common means of communication are through social networks or discussion forums. This produces an enormous amount of text data, which often needs to be automatically checked, classified and filtered to identify malicious categories such as hate speech, fake news or spam. This is handled by automatic classifiers. However, the classifiers can be fooled by an adversarial attack, in which the text is slightly modified in a way that it is no longer auto-classified as, for example, hate speech but is still considered hate speech to the human eye. In this diploma thesis, I have studied these attacks extensively and discovered that many of them suffer from a poor quality and frequently do not preserve the semantics of a sentence. Based on my research, the problem lies in the similarity metric, which uses Universal-Sentence-Encoder (USE) [39]. To overcome this, I propose a new approach called Semantics-Preserving-Encoder (SPE), which replaces USE in the similarity metric. Due to the supervised learning of our approach, we should capture the semantics better. This is proven valid, and the similarity metric using our SPE produces very good results on several datasets. Finally, I propose a new adversarial attack, which uses the new metric and modifies sentences on both character and word level. This attack produces high-quality adversarial examples and is also much faster than existing state-of-the-art attacks.

Keywords: Adversarial attacks, text classifiers, NLP, semantic metric, neural networks, sentence vectors

Supervisor: Ing. Tomáš Mikolov, Ph.D. CIIRC, CTU in Prague

Abstrakt

V dnešní době se velká část mezilidské komunikace odehrává na sociálních sítích nebo diskusních fórech. Vzniká tak obrovské množství textových dat, která je často nutné automaticky kontrolovat, klasifikovat a filtrovat tak, aby bylo možné identifikovat škodlivé kategorie, jako jsou projevy nenávisti, fake news nebo spam. To realizují automatické klasifikátory. Klasifikátory však lze oklamat adversariálním útokem, kdy je text mírně upraven tak, že již není automaticky klasifikován jako například nenávistný projev, ale je stále považován za projev nenávisti z pohledu člověka. V této diplomové práci jsem tyto útoky důkladně studoval a zjistil jsem, že mnohé z nich trpí špatnou kvalitou a často nezachovávají sémantiku věty. Na základě svého výzkumu jsem identifikoval, že problém spočívá v metrice podobnosti, která používá Universal-Sentence-Encoder (USE) [39]. K vyřešení problému navrhuji nový přístup nazvaný Semantics-Preserving-Encoder (SPE), který nahrazuje USE v metrice podobnosti. Díky supervizovanému učení našeho přístupu bychom měli lépe zachytit sémantiku. To se skutečně prokázalo a metrika podobnosti používající naše SPE dává velmi dobré výsledky, a to na několika datasetech. Nakonec navrhuji nový adversariální útok, který používá tuto metriku a modifikuje věty na úrovni znaků i slov. Tento útok produkuje vysoce kvalitní adversariální příklady a je také mnohem rychlejší než stávající útoky.

Klíčová slova: Adversariální útok, textové klasifikátory, NLP, semantická metrika, neuronové sítě, vektory vět

Překlad názvu: Adversarialní útoky na klasifikátory textu

Contents

Figures

Tables

Chapter 1

Introduction

To breathe life into inanimate objects is thought humanity has been dreaming of for centuries. Thanks to the advances in the machine learning field we are closer to this goal than ever. Looking at the timeline of the recent progress in this field, we have moved from simple rule-based systems to algorithms with an ability to learn. These algorithms have gradually become part of our daily lives and most of us do not even notice their presence. They are used for product recommendation [1], spam filtering [2], speech recognition [3] and many more. Even large companies rely on these algorithms extensively, after all, they generate most of their income.

But these algorithms are not just a source of income for the rich, they are also in place for security purposes. Specifically, they often search for attackers in the network [4], they keep us safe while driving, or even drive instead of us [5]. Performing a successful attack and fooling these algorithms could have devastating consequences. Unfortunately, such attacks exist [6] [35] [37] [38].

Machine learning algorithms are still far from human-level intelligence. They are not as robust as one would think, and they can be easily fooled with a small perturbation unnoticeable to the human eye - adversarial examples. For example, we can modify a stop sign with small pixel noise, which is invisible to the human eye but will fool the system to read the stop sign as a speed sign [6].

This problem is one of the obstacles to the adoption of machine learning algorithms in these core systems and has not yet been solved [8]. Intuitively one might think that hiding the model will make these attacks impossible, due to the unknown gradient. But it turns out that for a majority of the attacks the model knowledge is not needed at all [7].

The idea of defense mechanisms against these types of attacks lies in the so-called adversarial training [8]. We take these perturbed and modified inputs and include them in the training dataset. This way we can make the

model more robust.

Even though these attacks are the most common in the computer vision field, they can work equally well in a discrete space like language. In this context, we can imagine this attack as the lowest modification to the text that fools the model. In this case, the human eye can obviously spot the difference, but if we preserve the semantics, the text should be considered the same. For example, we can introduce a typo, exchange a word with its synonym or we can paraphrase the entire sentence.

The core idea of this attack is to preserve the semantics of the original sentence. If we changed this, the attack would no longer be an attack by definition. Unfortunately, this is the case with some of the existing attacks [9], which do not often preserve the semantics.

The goal of this thesis is to overcome this problem and create a metric that would be able to successfully detect the changed semantics of the sentence by producing better vectors in the latent space. This would allow us to create better quality adversarial examples, which would result in a better attack algorithm. We could also use this metric to automatically and reliably decide if the attack was successful or not.

To give an outline of how this work is structured, in Chapter 2 the reader is introduced to the theoretical background, which our work builds upon. Following this chapter, adversarial attacks are explained, and the most well-known attacks are shown in Chapter 3. In Chapter 4 the method is described. Chapter 5 evaluates the performance of our solutions on several benchmarks. Results are shown in Chapter 6. After that, we discuss and conclude our achievements (Chapter 7, Chapter 8).

Chapter 2

Theoretical background

In this chapter, I would like to introduce the reader to the theoretical background upon which my work is built. In Section 2.1, we start with a brief introduction to classification problems in the field of natural language processing. Then, the reader is introduced to the basics of neural networks in Section 2.2, which is a key concept to understand. Once the basics of these computational models are introduced, more advanced concepts of deep learning are described, such as recurrent neural networks 2.2.1 and transformers architecture 2.2.2. Finally, the problem of sentence representation in NLP 2.3 and the basic concepts like the bag of words, N-grams, and word embeddings are explained.

2.1 Text Classification Tasks in Natural Language Processing

As the name of this section suggests, text classification in natural language processing is a task where we assign labels to the text in the form of words, sentences, paragraphs, documents, queries, etc. A more formal definition of the classification problem can be found in [10] [11], which both states the same. Text classification is the process of assigning a label from a set of labels to any text document provided as input.

The extension of this can be to categorize text documents into more than one class, which is called multi-label text classification. The only difference is that there is no limit to the number of classes to which the text can be assigned.

A typical example of the classification problem is sentiment analysis, where we want to decide the sentiment of a text, e.g. decide if a movie review is positive or negative. We can also use it to detect spam, answer questions,

categorize news, check if the hypothesis is true given the premise, and so on.

The possible applications of this process are immense. This puts pressure on the automation of text classification because doing it manually is impossible with the amount of data that fluctuates over the Internet.

However, this is a very difficult task due to the unstructured data of the text. It is not in a form of SQL tables with predefined features, which many engineers work with. On the other hand, if we use the right tools, we can gain a great deal of insight from the text, which we would not get from other sources.

Thanks to the advances in the machine learning field, we do not have to classify the text manually. Unfortunately, a small group of annotators with deep domain knowledge will always be needed, not only because of the extensions and creation of datasets but also because of the need for human evaluation of the algorithms themselves. But because of the previous work of such people, we now have a large amount of labeled data, which are commonly used as training data for our machine learning models.

2.2 Feedforward Neural Networks

The idea of using artificial neurons as a mathematical model was there for a long time [12], but for decades nobody seemed to make it work better than existing approaches. But due to the advancements in the last decades, these models are now considered a state-of-the-art approach in many computer science fields like computer vision, speech recognition, NLP etc.

The goal of a feedforward neural network model is to approximate an unknown function $f^* : R^m \rightarrow R^n$ [13], where m is the dimensionality of the input vector and n is the dimensionality of the output vector. For example, we can have the previously mentioned function f^* that maps the vector x to the vector y as follows: $f^*(x) = y$. The goal of the feedforward neural network is to approximate function f^* with function f, where $y = f(x, w)$ and w are the learned parameters that result in the best approximation of that function.

A simple feedforward neural network typically consists of three functions $f^1 f^2 f^3$. These functions are called layers, and together they create a function $f(x) = f^3(f^2(f^1(x)))$. These layers are typically called input, hidden, and output layers. When we have multiple hidden layers, we can refer to our model as a deep feedforward neural network. Our model can also be represented as a directed acyclic graph (DAG) because it does not have recurrent connections. In the model, information always moves forward, thus the name feedforward. This is in contrast to networks with recurrent connections, which are discussed in the next section.

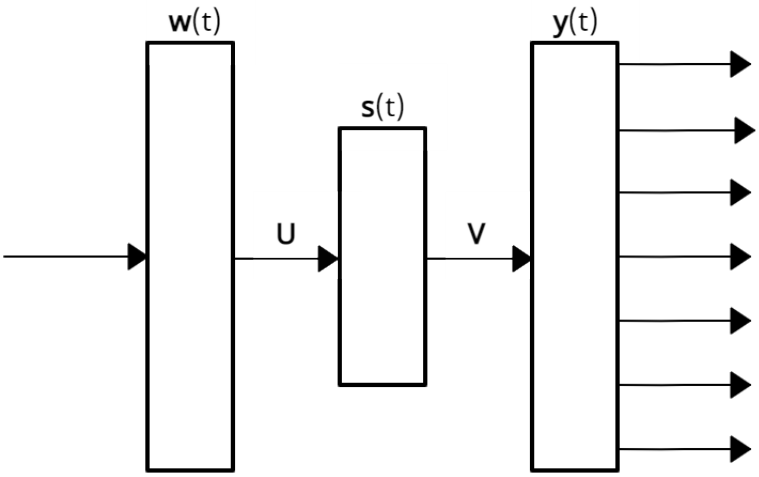


Figure 2.1: Illustration of the feedforward neural network as a matrix-vector multiplication as in [19].

Each layer consists of at least one neuron, and all layers except the input layer use some kind of a non-linear activation function on each neuron. Typical activation functions are sigmoid, tanh and relu. Due to the non-linear activation function, the data can be separated even though they are not linearly separable [15].

We can also define the feedforward neural network as a matrix-vector multiplication [19]. An illustration of this concept is depicted in Figure 2.1, where w(t), s(t), y(t) are the input, hidden and output layer vectors, U and V are weight matrices. U is the weight matrix between the input and the hidden layer, V is the weight matrix between the hidden and the output layer, b_0 and b_1 are the biases. The output values for each layer can be computed as follows.

$$s(t) = f(Uw(t) + b_0) \tag{2.1}$$

$$y(t) = g(Vs(t) + b_1) \tag{2.2}$$

where f(z) and g(z) are ReLU and softmax activation function:

$$ReLU(x) = max(0, x) \tag{2.3}$$

$$Softmax(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.4}$$

Optimization

Neural networks are typically trained by an optimization algorithm called stochastic gradient descent (SGD) [14] with back-propagation [16]. Training is typically done on a dataset, which is a sample of the probability distribution over the input space of the NN. In the case of supervised learning, we have input-output pairs. If we recall our previously mentioned function f with learn able parameters w [13]. Our goal is to approximate the unknown function f^*. If we want to find parameters to approximate our unknown function, we first have to define a loss function. This loss function measures the difference between the output of the function f and the true value. The most common loss function for classification tasks is cross-entropy, which can be defined as

$$-\sum_{c=1}^{M} y_{o,c} \ln(p_{o,c}) \tag{2.5}$$

In this equation, M is the number of classes, y is the binary indicator, which decides whether the label c is the correct classification for the observation o, and p is the predicted probability if the observation belongs to class c. Minimizing this error and adjusting w (weights) through back-propagation [16] is called learning.

For each training sample x and label y, SGD performs the following weight update:

$$w = w - \eta \nabla_w J(w, x, y) \tag{2.6}$$

where J is the objective function and w are the model parameters. The important factor is the size of the learning rate. It determines the size of steps which are in the direction of a slope. The learning rate is usually constant or decreases during the training. The learning rate can also be optimized automatically through specialized optimizers, such as Adam [31].

2.2.1 Recurrent Neural Networks

Recurrent neural networks (RNN) [16] are an extension of feedforward neural networks. Due to their recurrent nature they are suitable for sequential/time series data like videos, speech recognition and so on.

The simplest form of RNN is the Elman network [17], which consists of an input, hidden and output layer. For each time step, the state of the hidden layer is saved and used in the next time step along with the input. Because of this, the network can maintain some sort of memory through the hidden layer.

Mathematically, we can define RNN as an extension of NN from the previous Section 2.2. Following the description in [19], the equation is just extended

by a matrix W, between the input and hidden layers and the biases b_2. The hidden vector is then calculated as follows:

$$s(t) = f(Uw(t) + b_0 + Ws(t-1) + b_2) \tag{2.7}$$

The rest of the equation remains the same.

Training can be done in the same way as in feedforward neural network by using SGD with a back-propagation algorithm. However, as stated in [19], better performance can be achieved by using Back-propagation Through Time algorithm described in [30].

RNN, despite their tremendous success, have a few problems. They do not perform well when handling large sequences of texts [18], e.g., long paragraphs. Due to the long sequences, it is hard to propagate the gradient (vanishing, exploding gradient). As a consequence, the model might have already forgotten what it learned at the beginning of the sequence by the time it reaches the end. This issue can be particularly solved by using LSTM [20] or the gradient clipping technique [19]. However, a bigger problem of RNN is sequential input processing, which makes parallelism not really possible.

2.2.2 Transformers

Due to the problems of RNN, a novel architecture called Transformers was introduced [21]. Through this architecture, we were able to train large models like GPT-3 [25], T5 [24], BERT [23], because of the power of parallelism. This architecture does not have any recurrent connections. It is based only on feedforward neural networks with an attention mechanism. Surprisingly, even without recurrence, the architecture is able to encode sequential data.

The architecture of the model can be seen in Figure 2.2. The main advantage is that the input sequence can be passed in parallel. Architecture is built on encoder-decoder blocks. An important feature is a positional encoder, which is a vector that contains information on the distances between words in a sentence. This vector is applied to word embedding, thus we can get word vectors with positional context information. This is then passed into the encoder block. The encoder block consists of two parts: multi-head attention and feedforward neural network. Attention gives us information about what part of the input we should focus on. At each time step, the feedforward neural network is applied to the attention vector.

The decoder works similarly to the encoder block with the context word embedding part. Attention vectors and vectors from the encoder are passed to another attention block, where the layer determines how word vectors are related to each other. Then the feedforward layer with softmax is applied. Even though this architecture was developed mainly for translation purposes, it became the SOTA approach in several NLP tasks.

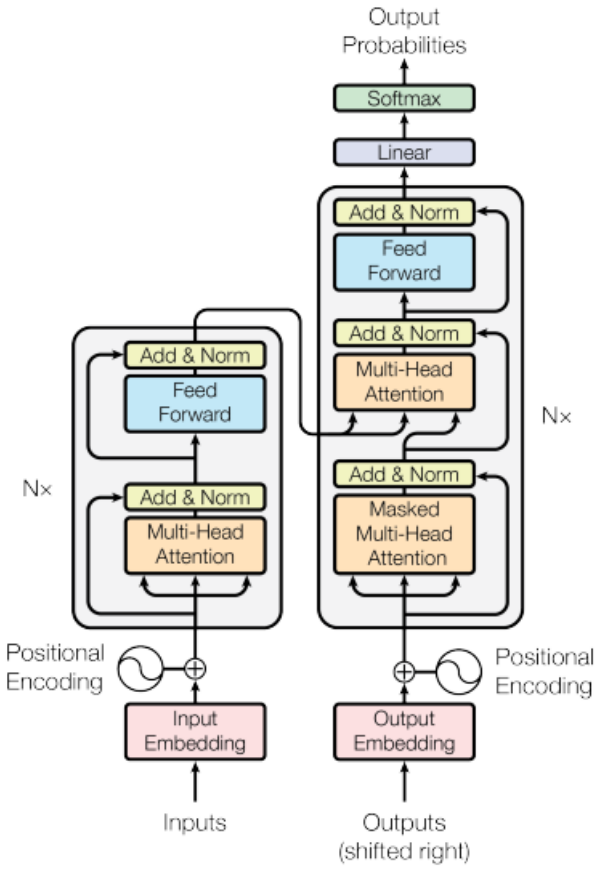


Figure 2.2: Transformers model architecture [21].

2.3 Sentence Representation in NLP

From the previous section, we are familiar with how a neural network works with numbers, especially vectors, but we want to work with text. An important question arises. How to convert the text into vectors. In this section, we will show the most common ways of achieving this, starting with the most famous one - the bag of words, following with some of the more advanced techniques, such as N-grams. Lastly, we finish this section with word embeddings, which is the best text representation we have, because it captures semantics.

2.3.1 Bag of Words

The simplest way to transform a word into a vector is to define an input layer with a node for every word in the vocabulary [22]. Each node would represent one word in our vocabulary. Thus, if we would like to only input one word into the neural network, the input vector would be all 0 except at one index

with 1, to which our word is assigned. This representation of a word is called one-hot encoding.

If we extended this idea to a sentence representation, we would have an input vector with the same length, again filled with zeros, except for the indices where the words occur in the given sentence. This is called one-hot bag-of-words encoding.

Another extension of this would be not to put the 1 on indices where a given word appears, but to count how many times the word appears in the sentence, document, etc. This way we would also have information about the word frequency. This is called Bag of Words (BoW) [22].

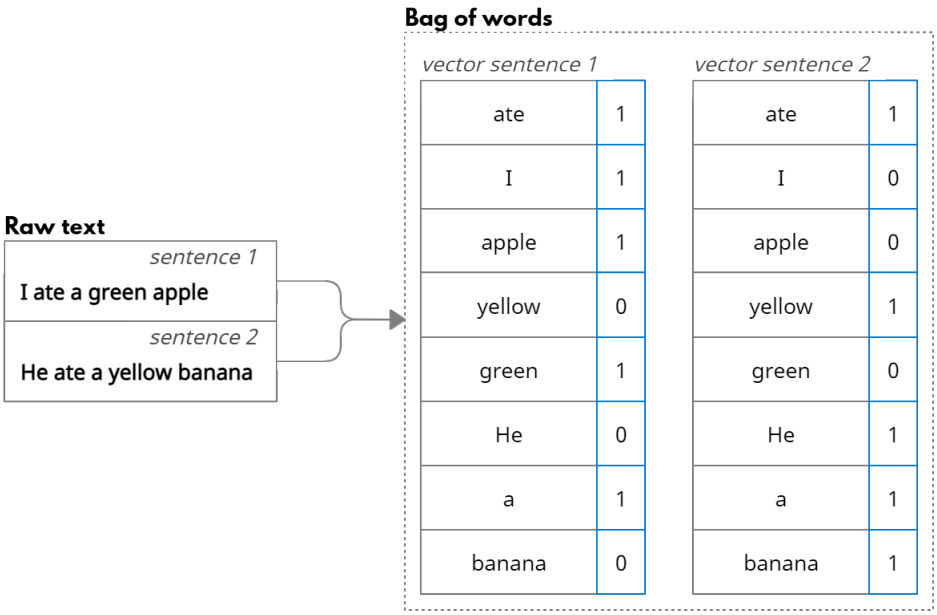


Figure 2.3: Example of a Bag of Words.

An example is shown in Figure 2.3. If we had a vocabulary with eight words, the BoW input vector would have a dimension of eight. If we wanted to input the word 'apple', the vector would be 00100000. If we wanted to input the sentence 'apple apple banana', the input would be 00200001.

However, in this representation, we lose information about the word order. Thus, the 'apple banana' sentence is the same as 'banana apple'. But the word order is usually important because it tells us a lot of information about the meaning. Another problem is that there is no semantic information about the word vector. We would expect words with the same meaning to be close to each other, but this is not reflected in this representation.

2.3.2 N-gram Features

The problem of word order information loss of BoW [22] explained in the previous section is partially solved by N-grams.

N-gram can be defined as a continuous sequence of items, in our case words, from a given text. The length of the continuous sequence of items is defined by N. Usually N with a value of 2 or 3, called bigrams or trigrams, is used. A higher N leads to far more combinations, which are computationally impossible to store.

Bag of Words can be viewed as a bag of 1-grams. To introduce the word order, at least partially, we could modify the BoW to use N-grams, so that we would have a bag of N-grams. A great example can be shown in a simple sentence: 'I am David'. In the bi-gram model, we would have a vocabulary with 'I am', 'Am David'. With a Bag of 1-grams model, we would have 'I', 'am', 'David'.

2.3.3 Word Embeddings

However, even the Bag of N-grams does not reflect any relationship between the vectors. Also, these vectors have high dimensionality and are sparse. The ideal scenario would be to have vectors with many times fewer dimensions, which contain some useful semantic information [26]. These vectors are called word embeddings.

There are several methods to generate these word vectors, but the most successful is the one that uses neural networks [27]. Some of these methods are used in our work, and it is necessary for the reader to understand how they work.

FastText

FastText is a library, which is capable of creating both supervised and unsupervised word embeddings and text classifiers [65] [66]. It can be viewed as an extension of the famous word2vec library [27]. A major difference is that in word2vec the smallest unit was the word in the vocabulary.

In the fastText approach, each word is represented as a bag of character N-grams. This allows us to compute the word representation for words that have not been seen in the training data. Another advantage is that the usage of character N-grams is also helpful in captivating the semantics of morphologically rich languages [65].

Word embeddings are typically trained unsupervised with the usage of skip-gram or CBOW [27]. But fastText also enables them to be trained on classification tasks [65]. These approaches differ greatly. In unsupervised learning, words that appear in the same context have similar vectors. Thus, the antonyms will be close to the synonyms, which is the problem we stated in the introduction.

However, if we train word embeddings on a supervised task, the meaning of our word embeddings will be different. The words that are the most discriminative for a certain label will be close to each other. A good example would be sentiment analysis on a movie review dataset. With supervised training, the words *'best'* and *'worst'* would be far from each other because they result in a different label. However, with unsupervised learning, they would have vectors close to each other, because these words usually appear in the same context.

The advantage of the fastText classifier is that it achieves almost state-of-the-art performance and is faster than current deep learning approaches on many orders of magnitude [65]. This is due to the simplicity of the algorithms used - it is a linear model. The architecture is similar to that of CBOW, but the middle word is replaced by the label [65]. There is only one linear layer, which computes $h = Ax$ for every input x. The h is the vector representation of a word in latent space. Then the output logits are computed $y = Bh$. The output matrix B is representing the classifier, which is multinomial logistic regression [28]. The A is the input matrix that represents the word vectors. Then softmax f is applied. The goal is to minimize the negative likelihood in the classes [65]:

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log(f(BAx_n)), \tag{2.8}$$

where y_n is the label. Importantly, the dimension of the matrix A is usually only 10, so the vectors in the latent space have much lower dimensionality than the other approaches, which usually map vectors in a 512-dimensional space.

In our use case, we particularly need sentence vectors, not just word vectors. This is enabled with the fastText library as well [65]. For each word in the sentence, the vector is summed and then divided by the number of words in the sentence. This is how the sentence vector is obtained. If the classification is needed, we take our B matrix and multiply it. After that, softmax is calculated and we have a prediction score.

Universal-Sentence-Encoder

USE is the algorithm that transforms sentences into 512-dimensional sentence embeddings [39]. It does not employ the averaging of words in the sentence

like fastText [65]. Instead, it is trained directly on the sentences itself. The goal of this approach is to have a general representation of the sentences, and therefore it is trained on a variety of NLP tasks.

In the implementation, we have 2 options of how to encode the sentence into an embedding. The first approach is to use the transformer encoder from the transformers architecture [21]. This results in more accurate results but is much slower than the other approach called Deep Averaging Network (DAN) [29]. This approach is similar to the fastText approach, but among the words also the bi-grams are averaged. This results in reduced accuracy, but much better time complexity.

The training of the selected architecture is on unsupervised problems along with supervised training on SNLI corpus [32]. One of the unsupervised tasks is similar to skip-gram architecture, but instead of words, we predict sentences. Another unsupervised task is to choose the correct response for input, among the given responses. The supervised task, on which the architecture is trained, is SNLI corpus [32], where the model has to choose if the hypothesis entails, contradicts, or is neutral to the premise.

BERT

BERT is now considered a state-of-the-art approach in word embeddings [23]. The acronym stands for Bidirectional Encoder Representations from Transformers. From that, the reader can correctly assume that it is built on the transformer architecture described in Section 2.2.2, where the bidirectional encoder is used. The advantage of this model is that it can capture a bidirectional context.

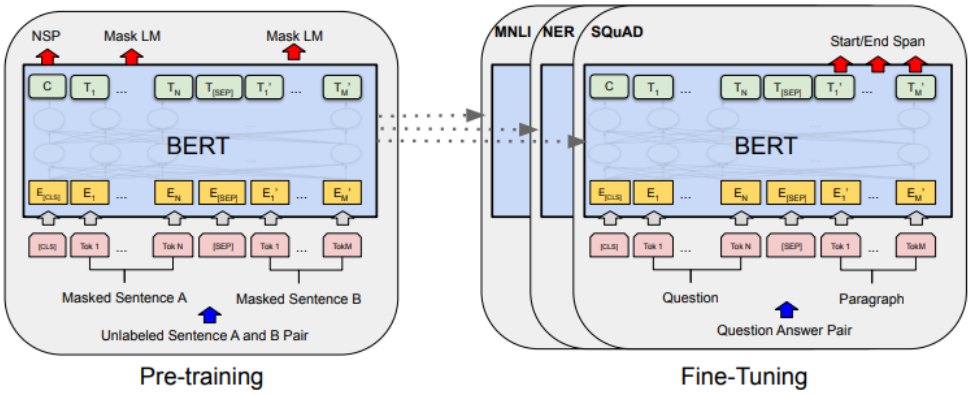


Figure 2.4: BERT model [23].

The idea behind learning word embeddings in BERT is similar to skip-gram [27]. An input sequence is masked with mask tokens and the goal of this model is to predict the original value of the masked tokens from their surroundings.

The model is also trained on the next sentence prediction task. In this task, the model tries to predict if the sentence is after the sentence given.

This first part of training is called the pretraining phase. The model is unsupervised-trained on NSP and Masked LM tasks. But it can be fine-tuned to perform downstream tasks like sentiment analysis or question-answer pairs. This can be seen in Figure 2.4, where a pre-trained model was fine-tuned on several tasks.

Fine-tuning is usually done by adding another feedforward neural network, as we described in Section 2.3.3. This layer is attached to the end of the model, e.g. for the sentiment analysis, a classification layer is added on top of the output.

Chapter 3

Related work

In this chapter, we will first have a brief look at the theory behind adversarial examples for text classifiers in Section 3.1. Then, the basic taxonomy will be presented in Section 3.2 and attacks are categorized. Lastly, state-of-the-art attacks are presented and compared with each other in Section 3.3.

3.1 Adversarial Examples in Machine Learning

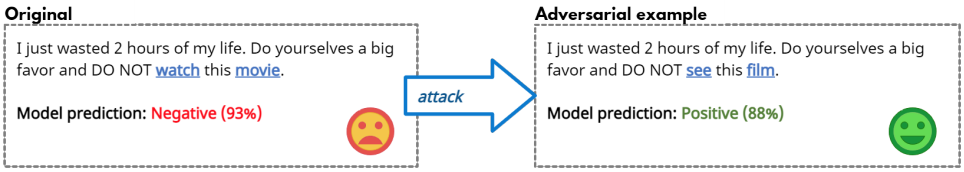


Figure 3.1: Adversarial example for text classification.

In this section, I want to define the adversarial attack as broadly as possible and then specify it for our case: text classification ([33]). Adversarial attack can be defined as follows. Let f be a threat model, x is the input and y is the correct label. We assume $f(x)=y$. An adversarial attack is the process of generating x', which is the modification of x, such that $f(x') \neq f(x)$. The modification should be minimal, so that human predictions on x and x' are the same.

In our case, f is the text classifier, x is the sequence, and y is the correct label. The goal of an attack is to create x', by modification of x, such that $f(x') \neq f(x)$.

In computer vision, the modification is usually an addition of some pixel noise. In our case, it is often an addition, removal, or substitution of a letter or a word, as Figure 3.1 demonstrates.

The next part of the definition is the proximity of x and x'. In computer vision, such modification in the continuous space can barely be seen, however, the text perturbation is more likely to be spotted by a human observer. So, the goal is to require semantic similarity. This is usually done by defining a similarity metric and a threshold. If the result of the similarity metric exceeds the threshold, then f and x' are not similar, therefore it is not an adversarial example. Typically, the similarity metric uses sentence encoding techniques, such as USE [39], to calculate word embeddings and then measure their cosine similarity in the latent space.

3.2 Categories of Attacks

To have a good comparison of the attacks, we need to understand their differences, which can be significant. Attacks differ in their access to the model - complete knowledge of parameters, only the softmax output, or nothing at all. They also differ in the text perturbations - characters, words, sentences etc.

We can categorize intuitively, which has already been done in literature before [33]. All the categorizations available are alike. The taxonomy defined in [33], where the approaches were systematically analyzed and divided into categories, is a good starting point on which other works are based or which they extend, such as [34]. I would like to describe several aspects of this classification in detail: adversary's knowledge, adversarial specificity, and granularity [34], which are the properties that will be in focus the most.

- *Adversary's knowledge* is based on how much information we know from the DNN model. We divide this category into two parts: a white-box attack and a black-box attack. In a white-box attack, we know all the information about the model - the weights, architecture, loss function, activation function, training data, etc. In a black-box attack, we do not have access to the model. The only information available is the output of the model or the confidence scores.

- *Granularity* is an extension property provided in the extended taxonomy by [34]. It categorizes an attack based on the level at which the perturbations are made. It could be done at the character-level, the word-level, or the sentence-level.

- *Adversarial specificity* allows us to split the attacks into two categories as well. Targeted attacks try to generate an example that misguides DNN to a specific class. For example, we could purposely classify a stop sign as a speed sign. In the non-targeted attack, we do not care about the class, unless the output is wrong. For example, we do not care if a

stop sign is classified as an elephant or a giraffe unless it is not classified as a stop sign.

We could also mention a similar categorization, which was done in the OpenAttack paper [49].

Attacks are filtered according to the model access ability into four classes: gradient-based, score-based, decision-based, and blind-based. Gradient-based is equal to our white-box category. These attacks require complete knowledge of the model. The black-box models are covered in the remaining 3 categories: score-based, decision-based, and blind-based. In the first one, knowledge of the classification probabilities of the model is needed. The second one only needs to know the final predicted class and the last one does not need any information at all. Then attacks could also be filtered according to the level of perturbations e.g. char, word, and sentence-level. Finally, attacks can also be targeted or non-targeted.

We can see that the categorization of the technique is very similar across different literature. However, it is worth mentioning that most of the models support both targeted and non-targeted attacks. Therefore, we will not consider this last category and only focus on the model accessibility and the level of perturbation.

3.3 Adversarial Attacks

From the previous section, we are familiar with the classification of the attacks. The current approach in the field of textual adversarial attacks focuses on black-box attack models, which are a more realistic scenario because in reality models are not accessible most of the time. That is the reason why we have also focused on this kind of attack and its problems. Despite this fact, we will still mention how the white-box attacks are constructed.

3.3.1 White-box Attack Models

Most of these attacks use the fast gradient sign method [50], which tricks the neural network into making a wrong decision due to the knowledge of the gradient. This is quite trivial if we have access to the victim's model. The other common practice is to use the forward derivative method [51], which constructs adversarial saliency maps indicating how to produce adversarial samples.

3.3.2 Black-box Attack Models

We will now focus on the more common black-box setup, where the gradients are not available and the selection mechanism is not so straightforward. These attack models can be divided thanks to the *Granularity* category mentioned in Section 3.2.

Character-level Attacks

One example of a character-level attack could be DeepWordBug [35]. This algorithm generates small text perturbations by swapping adjacent characters, deleting, inserting a character, or substituting it with a random one. This causes the deep learning model to misclassify the text input most of the time [35].

The other type of attack could be VIPER [37], which aims for a visually similar character substitution and does not need any information from the victim's model.

Word-level Attacks

In the TextFooler [40], words are ranked in a sentence by their prediction relevance. The most important ones are replaced with the most similar synonyms. These synonyms are found due to the word embedding being optimized for this task [41]. Another word-level attack is BERT-Attack [38], which generates adversarial examples using a pre-trained BERT [23] language model. Then it greedily replaces tokens with the language models prediction. Other attacks with a similar idea are BAE [47] and CLARE [42]. For instance, CLARE can not only replace words with others but also supports insertions and merges. Therefore, the output and input lengths can differ.

Sentence-level Attacks

The sentence-level attacks usually create paraphrases of the original at sentence-level as SCPN algorithms do [44]. This algorithm is considered blind, so it does not need any information about the model. The other option is for example adding distracting sentences [45] or text generation using encoder-decoder [46].

Attacks	Accessibility	Perturbation	Core Idea
VIPER [37]	Blind	Char	Character substitution to look visually similar
DeepWordBug [35]	Score	Char	Character manipulation (e.g. substitution, swapping, insertion, deletion)
TextFooler[40]	Score	Word	Word substitution with synonyms
Genetic [52]	Score	Word	Word substitution with a usage of genetic algorithm
BERT-Attack [38]	Score	Word	Word substitution with usage of a BERT language model
BAE [47]	Score	Word	Word insertion and substitution
CLARE [42]	Score	Word	Contextualized perturbations with replace, insert and merge operations
SCPN [44]	Blind	Sentence	Sentence paraphrasing
GAN [46]	Score	Sentence	Usage of encoder-decoder to generate text

Table 3.1: In the first row "Attacks" refers to the attack models. "Accessibility" refers to the accessibility to the victim's model. "Perturbation" states on which level the attack was performed: Char, Word, or Sentence. The last column "Core Idea" explains the main idea behind the attack model.

Technique Summary

In summary, the char-level attacks do not have a problem with antonyms, due to their simplicity in swapping adjacent characters, deleting, or inserting a character. At the word-level, attacks often need information about the model scores. All the algorithms mentioned above enforce semantic similarity, where Universal-Sentence-Encoder [39] is used as a constraint. At the sentence-level attack, paraphrasing or adding a disruptive sentence is often the case.

To provide an even better understanding of the information mentioned above, some of the commonly known attacks are listed in Table 3.1 together with the information about the accessibility, perturbation, and their core idea.

Chapter 4

Method

In this chapter, I would like to introduce our new Semantics-Preserving-Encoder (SPE) as a replacement for Universal-Sentence-Encoder [39] in a similarity metric, along with an adversarial attack, where this metric is used. In the first Section 4.1, the motivation behind the development of the new metric is described. The SPE itself is then outlined in Section 4.2, and a similarity metric using this approach is defined next in Section 4.3. Lastly, the new adversarial attack is proposed and described in detail in Section 4.4.

4.1 Motivation Behind the New Metric

The motivation to create a new metric arises after looking closely at the adversarial examples generated by SOTA algorithms such as TextFooler [40]. The problem I have identified is that some of the attacks do not preserve the semantics and therefore cannot be considered successful. Interestingly, the same problem has already been observed in other sources, such as [9].

A good example of this problem can be seen in Figure 4.1, where we have a pair of sentences to demonstrate such a situation. As we can see, the original sentence was successfully modified by the adversarial example in such a way that the prediction of the model changed from positive to negative. Also, the cosine similarity between the sentences is higher than the threshold, therefore, this attack would be considered successful by a SOTA attack algorithm. However, even though the two sentences are close to each other in the vector space (based on their cosine similarity in relation to the threshold), each sentence has a substantially different meaning, which should be reflected in the word embedding, and finally, the attack should not be considered successful.

To further examine the cause of the problem, we need to look at the similarity metric in detail. A majority of attacks use the Universal-Sentence-

Encoder, described in 2.3.3, which encodes the original and the perturbed sentence into a 512-dimensional vector space and then measures the cosine distance between these vectors. If the cosine distance exceeds a certain threshold, the sentences are considered to be similar, otherwise they are considered dissimilar.

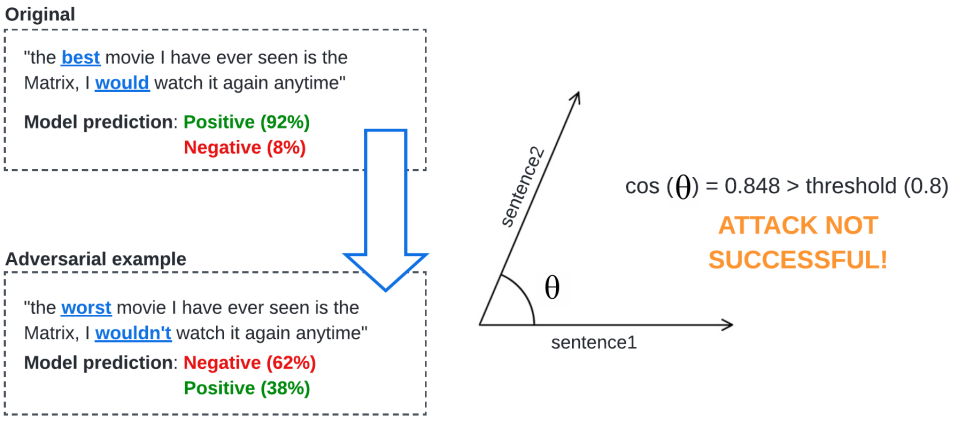


Figure 4.1: An adversarial example generated by an adversarial attack algorithm that does not preserve the original meaning.

One point of view sees the threshold as the main problem [9]. Generally, attacks have a low cosine distance between the vectors set as their threshold and the authors of [9] propose to increase it. When the threshold is increased to 0.95, the semantic similarity issue becomes much more scarce, which was confirmed by a human study that evaluated more attacks to be successful in comparison with the lower threshold setting [9]. Because of the stricter criteria of what is considered to be similar (higher threshold), the semantics of the sentence is more likely to be preserved. However, the downside is that the overall attack success rate drops significantly by more than 70% [9], making this approach difficult to apply.

In my opinion, the problem lies in the Universal-Sentence-Encoder [39] itself more than in the threshold. This encoder is mostly trained on unsupervised tasks such as skip through as described in Section 2.3.3. The only supervision comes from the SNLI dataset. Since this problem with semantic similarity cannot be easily auto-detected at this stage and we are largely dependent on human evaluation to even recognize this problem, we cannot expect an encoder mostly trained on unsupervised tasks to recognize and avoid it.

Building on that, we could create a new encoder that takes advantage of human-labeled datasets. Thus, the encoder will be able to recognize which words change the semantics or the label and prevent examples in Figure 4.1 from happening, by producing better sentence vectors in the latent space. This way we should be able to generate high-quality attacks with the same or higher performance while avoiding the substantial attack success rate drop of the increased threshold approach.

4.2 Semantics-Preserving-Encoder

As stated in the previous section, we decided to create a new sentence encoder called Semantics-Preserving-Encoder that will tackle the problem of changed semantics by using our new supervised sentence embeddings. As a result, the words that are the most discriminative for the given label will be close to each other in the vector space. For the implementation, the fastText library [65] has been chosen due to its simplicity, speed, and the results of the classifiers, which are on par with deep learning approaches.

The core idea of our SPE algorithm is to combine multiple classifiers trained on different tasks, which will allow us to have a diverse set of different sentence vectors (each classifier creates a sentence vector from the input sentence as described in 2.3.3). The sentence vectors will differ from each other because each classifier produces its vector according to the task on which it was trained. Therefore, the diversity of the classifiers implicates a diverse set of sentence vectors. By combining several sentence embeddings from different classifiers, we can create a robust classifier, which can produce a high-quality embedding for a broad range of topics.

In the implementation of our Semantics-Preserving-Encoder, we first select and train a set of the classifiers to be used. Then, we simply average the different sentence embeddings received from each classifier to create one general sentence embedding as our final embedding as shown in Figure 4.2.

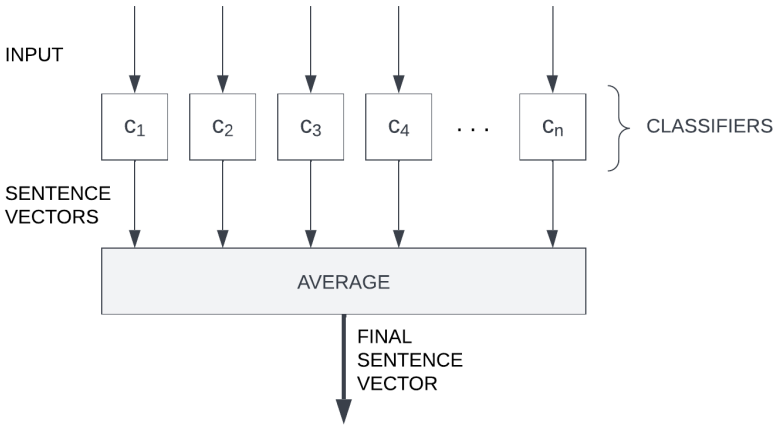


Figure 4.2: Concept of our Semantics-Preserving-Encoder which is used in our metric.

The classifier selection and training process are key to achieving a robust solution with high-quality results. To give a general outline of this process, we can distinguish two steps. First, to train several fastText [65] classifiers on general NLI tasks such as STS Benchmark [63] and SNLI [62], which will

provide general language understanding. Second, to employ several classifiers trained on more applicable datasets, such as Yelp reviews [59], Amazon reviews [61]. The selection process of the classifiers, their detailed description, and their hyper-parameters are explained in depth in Chapter 5.

As mentioned above, we have used the fastText library [65] in our SPE. Due to the speed of fastText [65] classifiers, we should be able to create sentence vectors quickly with SOTA performance for the given task. Another advantage is the reduction of the dimensionality of the vector space. Since the hidden layer of these classifiers is 10, we will have 10-dimensional vectors instead of 512-dimensional ones. This way can put more information into fewer dimensions, which results in more efficient space storage.

The pseudocode of our SPE algorithm can be described as follows:

Algorithm 1 Semantics-Preserving-Encoder

```
1: Input: Original sentence and perturbed sentence
2: Output: Original sentence vector and perturbed sentence vector
3: original_vectors ← []
4: perturbed_vectors ← []
5: for classifier in classifiers do
6:     original_vector = classifier.get_sentence_vector(original_sentence)
7:     perturbed_vector = classifier.get_sentence_vector(perturbed_sentence)
8: end for
9: original_averaged_vector ← original_vectors.mean()
10: perturbed_averaged_vector ← perturbed_vectors.mean()
11: return original_averaged_vector, perturbed_averaged_vector
```

The logic of our SPE is described in the pseudocode in Algorithm 1. The input of our function is the original and perturbed sentences, and the output is their vector embeddings. A "for each" follows on line five of the code, in which we iterate over the models and create sentence vectors of the original and perturbed sentences. These vectors are stored and later averaged to a final vector for each sentence.

4.3 Similarity Metric Using SPE

Similarity metric in adversarial attacks tells us if the sentences are similar or not. Typically, Universal-Sentence-Encoder is used to create sentence embeddings, and then the cosine distance is measured between these vectors. Cosine distance between two vectors x and y is defined as:

$$\cos(\mathbf{x}, \mathbf{y}) = 1 - \frac{\sum_{i=1}^{n} \mathbf{x}_i \mathbf{y}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{x}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{y}_i)^2}} \tag{4.1}$$

If the cosine threshold is surpassed, sentences are similar.

In our similarity metric, we substitute USE with our SPE, for the already mentioned reasons. The rest of the similarity metric remains the same.

We also need to define the cosine threshold for our metric, which we expect to have a big impact on the accuracy. In many adversarial attacks, the cosine distance is set to 0.85, however, study [9] suggests increasing the distance to 0.95, which achieves the most accurate results. This is the reason why we also use 0.95 as our cosine threshold on our adversarial datasets. Nonetheless, for the NLI datasets, we decided to use the original 0.85 due to the nature of NLI.

4.4 Combined Adversarial Attack

Building on the new metric, the goal was to create a new adversarial attack that produces less problematic and semantically incorrect examples than other existing attacks. We should be able to achieve this mainly by substituting USE[39] with our SPE in the similarity metric. This should result in better quality attacks with a higher success rate.

Analysis of existing attacks shows that the vast majority of them focus on just one of the three granularity categories - char, word, or sentence and that there are very few that combine these approaches [36]. Motivated by the fact that the combined approach has not been explored as much and the fact that it could produce more diverse attacks that are less likely to be detected, we decided to take the combined approach. Thus, we will create an algorithm that changes the sentence on both char and word level.

To be more specific, our adversarial attack will be an untargeted one and will be performed on a black-box model, meaning that nothing except the final softmax will be available. The black box scenario is the most realistic because most of the deployed machine learning models are not fully accessible, the access is limited to their final probabilities.

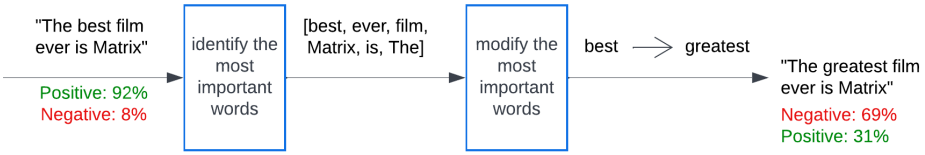


Figure 4.3: Scheme of the adversarial attack workflow.

Generally speaking, an adversarial attack receives the original sentence on input, analyses and modifies it, and therefore produces a perturbed sentence on the output. A schematic example of this workflow is shown in Figure 4.3.

In this workflow, we can distinguish the following two steps that need to

be executed:

1. Identify the most important words
2. Modify these words slightly while keeping the constraints

4.4.1 Identification of the Most Important Words

To perform an attack, we need to identify which words to modify or replace in the next step of the attack. Intuitively, it should be the words with the highest impact on a classifier. Existing attacks determine the importance of a word in several ways, but generally, a series of operations are performed sequentially on each word in a sentence, which assigns the word a final score equivalent to its importance.

DeepWordBug [35] uses a series of quite complex scoring functions that are weighted to get the final score of a word. The scoring process is based on the sentence being split into two by removing a word. Then, the two sub-sentences are fed into a classifier, and the classifier's outputs are compared. Finally, the differences are weighted and the final score of the word is estimated.

TextFooler [40] uses a simpler approach in which the importance of a word is determined by its removal from the sentence. This modified sentence is fed into a classifier and the difference between the prediction results is measured. The bigger this difference is, the more important the word is.

For our attack, we have chosen the simpler approach in determining word importance identical to the one of TextFooler [40] due to its simplicity and effectiveness.

The implemented word importance function can be described as follows. Given a document, meaning a sentence of words, $X = (x_1, x_2, ..., x_n)$ and a classifier F, where $y = F(X)$ is the final prediction. We construct a ranking function *rankit*, which is defined as the difference between the original X and the X', where the i-th word is removed:

$$rankit = F(x_1, x_2, .., x_i, x_i + 1, .., x_n) - F(x_1, x_2, .., x_i + 1, .., x_n) \quad (4.2)$$

It is important to mention that a majority of attacks remove the stop words from the ranking function. While TextFooler [40] argues that this is done to avoid grammar destruction, we observe that a certain degree of grammar destruction is almost inevitable in these kinds of attacks and sometimes it is even necessary e.g. typos. Moreover, in our observation, the stop words also influence the classifier. Given these facts, we have decided not to filter any stop words from our ranking function.

4.4.2 Modification of the Important Words

When the word importance ranking is completed, we can proceed with the word modification. In this step, the most important words are modified in order to change the output of a classifier while also maintaining the original meaning. Three important factors need to be determined: the levels of granularity of these modifications, the technique used to achieve them, and the constraints to be fulfilled.

As mentioned in Section 4.4, our attack applies the modifications on char and word level. The technique we have chosen for the char level modifications is based on the results of study [67], which show that char level modifications are more likely to preserve the meaning of the text when only a few characters are changed. Therefore, our char-level modifications have a scope limited to a single char per word. Specifically, we either insert a random character into the word, delete a random character from the word, or swap adjacent characters. This is similar to the logic used by DeepWordBug [35].

The word level modifications can also be approached in different ways. One option is to use a pre-trained language model like BERT [23] to choose the word substitution, merging, or deletion. Another simpler solution is to try to substitute the word with its synonym, which is implemented in TextFooler [40]. We decided to implement the second approach.

For the word for synonym replacement, we had to use a pre-trained word embeddings model such that the synonyms as word vectors are close to each other. For this reason, we could not use a model like word2vec [27], which does not distinguish between synonyms and antonyms and would consider both to be close to each other. Fortunately, there are context-aware models like GloVe [68] that are able to identify synonyms as the k-nearest neighbors of a word and the antonyms are filtered out. In our attack, we use GloVe [68] model for synonym replacement, where experimentally we chose k to be 10.

In summary, our modifications are:

1. **Insertion** - insert a random char into the word
2. **Deletion** - delete a random char from the word
3. **Swap** - swap adjacent chars in the word
4. **Substitution** - substitute a word with a synonym

Importantly, for the attack to be successful these modifications need to fulfill the following constraints:

- Preserve the semantics similarity

- Fool the classifier

The first constraint should be mostly ensured by our metric. The second constraint is to fool the classifier, meaning that our modifications to the sentence (the perturbed sentence) need to change the final decision of the classifier.

4.4.3 Pseudo-code Description

The principles described above are applied in code in Algorithm 2, which realizes our untargeted black-box adversarial attack. In summary, our attack ranks the words and sorts them by their importance first, then the words are modified until the constraints are fulfilled and the attack is successful.

Algorithm 2 Our adversarial attack

```
1: Input: Original text: text, our modification function: modification,
   our similarity metric: our_metric, text classifier: classifier, truth label:
   original_label, similarity threshold: threshold
2: Ouput: Perturbed text: perturbed_text or None
3: importance_score ← [−1] ∗ len(text)
4: for i in len(text) do
5:     importance_score[i]= rank_it(i, text)
6: end for
7: importance_score.sort(reverse=True)
8: perturbed_text = text
9: for word in importance_score do
10:    perturbation = modification(word, perturbed_text, original_label,
   classifier)
11:    perturbed_text ← substitute word in text with perturbation
12:    if classifier(perturbed_text) != original_label then
13:        if cos_dist(spe(perturbed_text, text)) > threshold then
14:            return perturbed_text
15:        end if
16:    end if
17: end for
18: return None
```

In the pseudo-code, several core sections can be distinguished. Lines 1-2 specify the input and output, lines 4-7 execute the ranking of the most important words and sorting in the descending order, and finally, the word modification stage is on lines 8-18.

The word modification is implemented in a "for loop" iterating over the sorted list of the most important words. The modification function is applied to each word, attempting to perform: insertion, deletion, or swapping on char

level or synonym substitution on word level. Out of these, the modification which changes the classifier probabilities the most is chosen. Then, on line 11 the original word is replaced with the modified one and on the next line, the constraints are checked. If the constraints are fulfilled, the perturbed sentence is returned and the attack is successful. Otherwise, the iteration over the ordered list of words continues and the whole process is repeated for the second most important word and so on. When no perturbation that satisfies the constraints is found, *None* is returned and the attack was not successful.

Chapter 5

Experiments

In this chapter, the experiments which were conducted to create, evaluate and compare our similarity metric and attack to the existing solutions are described. The results of these experiments can be found in Chapter 6.

In the first Section 5.1, the evaluation tasks that were proposed to measure the effectiveness and robustness of our methods are described. In the following Section 5.2, datasets, properties, and evaluation of the proposed similarity metric with SPE are presented. The last Section 5.3 focuses on the adversarial attack - the datasets, setup and methods of evaluation.

5.1 Tasks

To evaluate and compare our metric with SPE and the proposed adversarial attack with the others, evaluation tasks had to be defined and created.

First, we propose a metric evaluation task designed to compare our metric using SPE to the metric using USE [39] with both DAN and Transformers models. For metrics evaluation, we need datasets to be able to compare the quality of the two approaches. To achieve results that are not context-biased, it is necessary to use various datasets for the evaluation.

Building on that, we used 5 different datasets for the evaluation, 2 of which are aimed at evaluating the general performance and 3 of which are our own datasets focused on semantic similarity.

To evaluate the general performance of the metrics, we performed testing on two existing NLI datasets - MRPC [69] and SICK [72], which are described in detail in Section 5.2.1.

Due to the character of the semantic similarity preservation and the lack of existing datasets that would target this problem, using only the existing

datasets would not be sufficient. That is why we created our datasets - 3 in total that are also used for the metric evaluation. We call them datasets with contradictory and unrelated sentence pairs - DCUS. More details on the creation of these datasets can be found in Section 5.2.1.

The second task is designed to evaluate our proposed adversarial attack and compare it with SOTA attacks such as TextFooler [40] or DeepWordBug [35]. To do this, first of all, we need a victim model on which the adversarial attacks will be performed. For this task, we have chosen to use the pre-trained model BERT [23] as our victim model, which we fine-tuned on down-stream tasks like movie sentiment analysis, hate-speech, and offensive language detection. Once the adversarial attacks are performed on the victim model, statistics like success rate and modification rate can be measured.

In addition to these statistics, we need to evaluate how the semantics is preserved in the attacks. This can currently only be done through human evaluation, which can determine if the attack was successful or not. The specifics of the evaluation by human judges are described in Section 5.3.5.

In summary, two evaluation tasks are proposed:

1. **Metric evaluation** - evaluate the accuracy of our metric using SPE in comparison with the metric using USE [39] on 5 datasets in total - 2 NLI datasets and 3 own DCUS datasets.

2. **Adversarial attack evaluation** - evaluate the accuracy of our proposed attack in comparison with SOTA attacks (TextFooler [40], DeepWordBug [35]); evaluate the success rate by human analysis.

5.2 Metric Definition and Evaluation

In this section, the properties related to the metric using SPE are described. Firstly, the datasets are presented, then the model training with the hyper-parameters is described. Lastly, we conclude with the specification of the automatic evaluation process of the similarity metrics.

5.2.1 Datasets

The following section focuses on the datasets used for the fastText classifiers training [65] that are incorporated into SPE and the datasets used for metric evaluation.

Datasets Used To Train FastText Classifiers

As previously described in Chapter 4, our Semantics-Preserving-Encoder combines multiple fastText classifiers [65]. In this set of classifiers, we want to equally represent classifiers trained to provide a general language understanding, which will be trained on more general NLI datasets, and classifiers trained on more down-stream tasks like sentiment analysis or news categorization that will provide a more specialized knowledge.

Therefore, we need to determine which and how many datasets, NLI or down-stream, will be used in our SPE, which was done experimentally. We have trained 30 different classifiers, created numerous combinations, and comparisons of different aspects of SPE to find the best performing combination. The factors that were taken into consideration and the results of these experiments are further described in Section 6.1.2. In the end, we have determined that having 7 classifiers is sufficient. These classifiers were trained on 7 datasets in total - 4 NLI datasets: SNLI [32], COLA [69], RTE [69], SST2 [69], and 3 down-stream datasets: StackOverflow [71], Emotion [70], Yelp Review Polarity [59].

Because these classifiers are a key component of SPE, we find it necessary to describe them in depth:

1. **SNLI** [32] - Stanford Natural Language Inference Corpus is an English dataset for NLI created by crowd workers. It contains 570k human-written sentence pairs, which were manually labeled with one of 3 labels - entailment, contradiction or neutral.

2. **COLA** [69] - Corpus of Linguistic Acceptability from GLUE [69] contains sentences from books and journal articles that are labeled into two categories - whether it is a grammatically correct English sentence or not.

3. **RTE** [69] - Recognizing Textual Entailment from GLUE [69] consists of text data from news and Wikipedia that are labeled with one of three labels - neutral, contradiction, and entailment.

4. **SST2** [69] - Stanford Sentiment Treebank dataset from GLUE [69], where movie reviews are annotated with their sentiment - positive or negative.

5. **StackOverflow** [71] - consists of 60k StackOverflow questions from years 2016-2020. Each question is annotated with one of three labels - hq, lq_edit and lq_close, which determines the quality of a question.

6. **Emotion** [70] - contains 16k Tweets, which are labeled with one of six emotions - anger, fear, joy, love, sadness and surprise.

7. **Yelp Review Polarity** [59] - contains 598k Yelp reviews, where each review is assigned a binary label - negative or positive. The binary classification is extracted from the original Yelp reviews dataset [59], where 1 or 2 stars are considered negative and 3 or 4 are considered positive.

NLI Datasets

To prove the robustness of SPE applied in the similarity metric, we performed the metric evaluation on several NLI tasks, specifically on MRPC [69] and SICK [72] datasets.

MRPC stands for Microsoft Research Paraphrase Corpus [69], which contains sentence pairs that are binary labeled if they are semantically similar or not. The SICK dataset stands for Sentences Involving Compositional Knowledge [72] and it consists of sentence pairs annotated on a scale of 1-5, indicating how semantically similar they are, where 1 is the least similar, 5 is the most.

However, for our purposes a binary label was needed instead, that is why we modified the labeling of the SICK dataset as follows: when the original label is smaller than 3, the sentences are now considered dissimilar, and when the original label is bigger than 3, they are considered similar; when the label equals 3 it is considered borderline, and the pair is excluded from the modified dataset. After these modifications were made, the modified SICK dataset is referred to as SICK-Adjusted.

DCUS - Our Own Datasets

To provide an in-depth comparison between the metric with SPE versus USE, three new datasets were created and then used in the metric evaluation process. These datasets with contradictory and unrelated sentence pairs (DCUS) were created by human evaluation of adversarial attacks.

A set of sentence pairs from adversarial attacks, performed especially by TextFooler [40] attack, was presented to human annotators to be labeled. These attacks were done on 3 datasets - Rotten Tomatoes [60], hate speech [73] and offensive language Tweets [74], which represent some of the most common applications of the adversarial attacks. Especially hate speech and offensive language auto-detection is currently an emerging problem on the Internet.

For human evaluation, there were 3 annotators in total with English level C1 or higher that were instructed to label each sentence pair, the original and perturbed sentence, if it preserves the meaning. For example, if the original

film review is positive, the perturbed sentence should also be a positive film review in order to be labeled as meaning preserving. Also, there were some sentence pairs where the original sentence was modified to such extent that it was nonsensical, these cases were labeled as not successful by the human annotators.

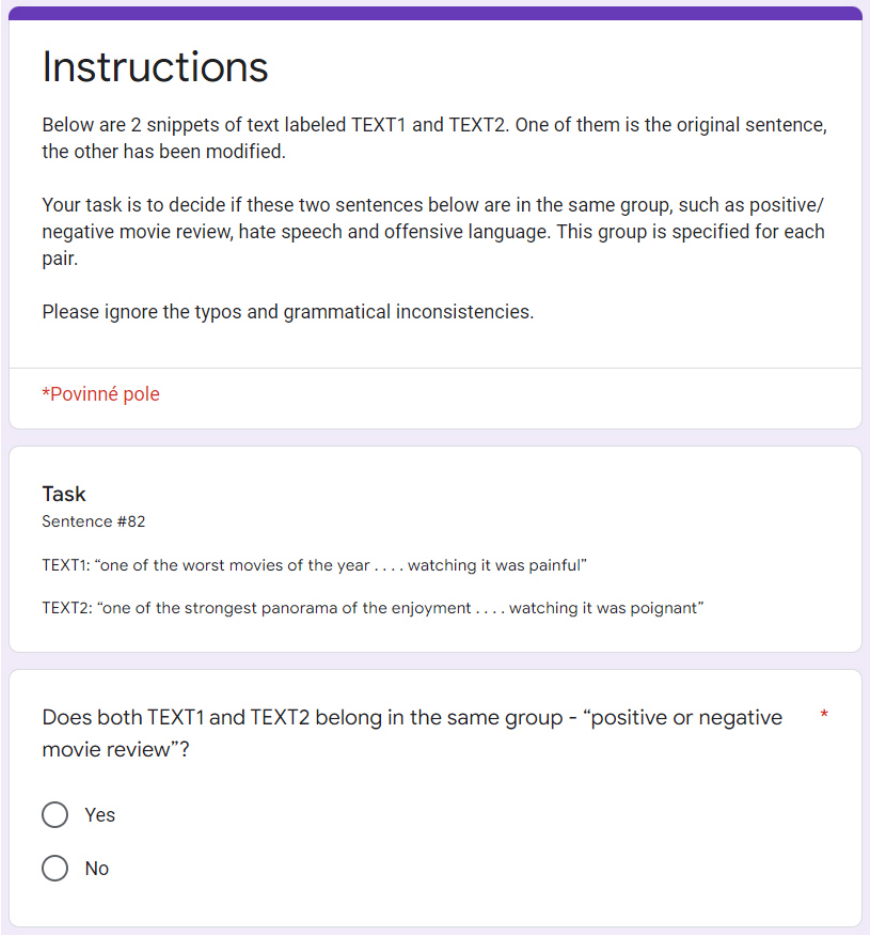


Figure 5.1: The instructions and questionnaire used for DCUS creation.

The dataset labeling process was carried out through a questionnaire handed over to the annotator. For each sentence pair, the original and perturbed sentences were presented, where the perturbed sentence is the sentence produced by the attack as a result of a series of modifications of the original sentence in order to fool the classifier. For the purpose of neutrality of the annotator, we did not specify which sentence is the original and which is the perturbed one in the questionnaire. For each sentence pair, the annotator was asked to answer a simple Yes/No question, whether both sentences belong in the same group, such as a positive or negative movie review. An example of the questionnaire is shown in Figure 5.1. It is important to mention that the annotators also had access to an online translator of their choice because some of the attacks use archaic or non-English word replacements.

Finally, the human labeling was post-processed. Because we had 3 annotators we have decided to consider the attack successful when 2 or more human judges agree. Meaning that when 2 people labeled the pair as meaning preserving, the attack is considered successful, otherwise, it is unsuccessful.

This allowed us to build 3 binary labeled DCUS datasets. For simplicity, we refer to them as DCUS-RT for Rotten Tomatoes, DCUS-Hate for Hate Speech, and DCUS-Offensive for Offensive language. Examples of some of the sentence pairs from the datasets can be seen in Figure 5.2. In the picture, the first sentence pair always shows a successful attack, where meaning is preserved. The rest of the examples are false positives, which have the cosine distance within the threshold, but do not preserve the meaning.

Movie reviews

Original and perturbed sentence	Meaning preserved
"more trifle than triumph"	TRUE
"more trifle than victoria"	
"one of the worst movies of the year . . . . watching it was painful"	FALSE
"one of the strongest panorama of the enjoyment . . . . watching it was poignant"	
"full of surprises"	FALSE
"full of stumped"	

Offensive speech

Original and perturbed sentence	Meaning preserved
"@user Grew up listening to K.I.D.S s**t breaks my heart RIP"	TRUE
"@user Grew up listening to K.I.D.S scheisse breaks my heart RIP"	
"@user @user So do you think she is a Christian?"	FALSE
"@user @user Where making you brainchild she is a Christian?"	
"@user He can s**k it."	FALSE
"@user He can breast it."	

Hate speech

Original and perturbed sentence	Meaning preserved
"US VP arrives in Ecuador, meets Venezuelan migrants"	TRUE
"US VP get in Equator, abide Cuban immigrants"	
"More Evidence that the US Constitution Does Not Mandate Federal Control of Immigration"	FALSE
"More Evidence that the AMERICANS Forming Does Not Duty Confederacy Control of Immigration"	
"How you b*****s look with your snapchat filters"	FALSE
"How you chickens look with your snapchat filters"	

Figure 5.2: Sample sentences from our 3 datasets: DCUS-RT (Rotten Tomatoes), DCUS-Hate (Hate speech), and DCUS-Offensive (Offensive language).

Datasets Used For Evaluation In Summary

To summarize, the datasets that were used to evaluate how the similarity metric using SPE compares to USE were 5 in total - 2 NLI datasets, MRPC [69] and SICK-Adjusted (modified SICK dataset [72]), and 3 own DCUS datasets. Table 5.1 shows the basic statistics of all the datasets. Naturally,

MRPC [69] and SICK-Adjusted [72] are the largest, but the classes are not well balanced. This can result in a situation where a dummy classifier that predicts only one label can achieve a decent accuracy on the unbalanced dataset. However, this is negligible for our purposes of testing NLI.

On the other hand, our DCUS datasets contain fewer sentence pairs but are much more determinant of the quality of the metric considering the semantics meaning preservation. It is important that these datasets are nearly balanced, containing a comparable number of equivalent and not equivalent pairs. Thus, when a high score is achieved on these datasets, it indicates that the metric is able to detect not only if the sentences are similar, but also that they are dissimilar.

Dataset	Average length [words]	Number of classes	Test examples
MRPC	18	2	1725
SICK-Adjusted	10	2	4762
DCUS-RT *(ours)*	13	2	220
DCUS-Hate *(ours)*	20	2	224
DCUS-Offensive *(ours)*	23	2	121

Table 5.1: Statistics of 5 datasets in total that were used for metrics evaluation.

5.2.2 Training FastText Models For SPE

In this section, the hyper-parameters of the fastText models [65] that we use in SPE will be specified. FastText models, despite their complexity, are conceptually simple. They employ basic logistic regression and include many hyper-parameters that allow us to adjust the model to our use case [65]. Some of the most important hyper-parameters are learning rate, dimensionality of vectors, loss function, and number of epochs. Other parameters such as the minimum and maximum length of char n-grams, word n-grams, number of buckets, minimal and maximal word and label occurrences can be also chosen [65].

In our implementation, most of these hyper-parameters are left with their default value [65]. The hyper-parameters which we have found important to modify are:

1. **epoch** - number of epochs
2. **lr** - learning rate
3. **dim** - hidden layer dimensionality (size of the word/sentence vectors)
4. **wordNgrams** - maximum length of word n-grams

5. **minn** - minimum length of char n-gram
6. **maxnn** - maximum length of char n-gram
7. **loss** - loss function used

For the dimensionality of the hidden layer, we have chosen the value 10, which corresponds to the value in the original fastText paper [65]. In comparison with the value 512 of USE [39], this number is very small, yet we are able to concentrate far more information into just 10 dimensions while achieving SOTA results on classification problems. For the loss function, we use a simple softmax function because our classification problem only has a few classes.

The remaining hyper-parameters were fine-tuned for our use case, which can be done with an automatic tool integrated into the fastText [65] library that finds the best values for the given task for us. The complete list of these hyper-parameters for each classifier is shown in Table 5.2.

Classifier	epoch	lr	minn	maxnn	wordNgrams
SNLI	5	0.05	3	6	4
COLA	1	0.09	0	0	5
RTE	11	0.09	6	3	1
SST2	55	0.04	6	3	5
StackOverflow	23	0.05	6	3	5
Emotion	6	0.073	6	2	3
Yelp Review Polarity	5	0.05	0	0	2

Table 5.2: Hyper-parameters of the fastText classifiers [65] used in our Semantics-Preserving-Encoder.

Another advantage of the fastText library [65] is the possibility to specify the size of the model. The size compression is realized through a quantization method [75], which is very effective. For example, if we take a 409MB classifier with 0.957 accuracy rate on Amazon Review Polarity dataset [59], we are able to reduce it to 1.5MB while maintaining the same accuracy rate [75].

Similarly, in our SPE where 7 classifiers are integrated, if each classifier had over 400MB in size, it would make it very difficult to work with the metric that implements it. Quantization solves this issue for us and helps us tremendously. Finally, we decided to limit our model size to 2MB.

5.2.3 Defining Metric Evaluation

This section explains the means of evaluation and comparison of the metric using SPE with others.

Regarding semantic similarity, the most important criterion is accuracy. It is measured for both metrics with SPE and USE [39] on the previously mentioned datasets. Accuracy is calculated as the number of correctly classified examples divided by the total number of examples. We also measure the time complexity of the algorithms.

5.3 Adversarial Attack Evaluation

This section outlines the properties of the adversarial attack, which integrates the metric with SPE. Firstly, the datasets are presented, then the model training and hyper-parameters that we used are described. Finally, the evaluation process is explained from the automated evaluation and human evaluation perspectives.

5.3.1 Datasets Used To Fine-Tune BERT Model

To perform an adversarial attack, we need a classifier. We use the pre-trained BERT model [23], which needs to be fine-tuned for our use cases. The fine-tuning is performed on three datasets well fitted for our problem. The last two datasets were chosen to reflect some of the most common problems in the Internet society, where hate speech and general disinformation are widely spread on both social media and various discussion forums.

The following datasets are used for the fine-tuning of our BERT model:

1. **Rotten Tomatoes** [60] - dataset of short movie reviews that are binary labeled as positive or negative.
2. **Hate Speech TweetEval** [73] - dataset of Tweets with hate-speech characteristics, which are labeled as hate or non-hate.
3. **Offensive Language TweetEval** [74] - dataset of Tweets with offensive language characteristics, which are labeled as offensive or non-offensive.

The statistics of all three datasets are specified in Table 5.3, where we can see that all datasets are robust, have binary labels, and are split into a test and train subset.

5.3.2 Training BERT Victim Model

Now to fine-tune our pre-trained BERT model [23] was, we need to define hyper-parameters for it. Despite the more complex nature of this model compared to fastText classifiers [65], we discovered that fewer hyper-parameters

Dataset	Average length [words]	Number of classes	Train subset	Test subset
Rotten Tomatoes [60]	18	2	8530	1066
Hate Speech TweetEval [73]	20	2	9000	2970
Offensive Language TweetEval [74]	22	2	11916	860

Table 5.3: Statistics of 3 datasets in total that were used in experiments with the adversarial attacks.

have an impact on the final model accuracy. Therefore, for our BERT model, most of the hyper-parameters were left with their default values and only the following parameters were modified:

1. **epoch** - number of epochs
2. **lr** - learning rate

The learning rate was set to 2e-05 and the number of epochs was set to 10 for all the datasets. The maximum sequence length also needed to be specified, which was done according to the dataset properties.

5.3.3 Adversarial Attacks Configuration

At this point, we have explained the datasets and training of the model used in our attack. However, the adversarial attack itself also has some hyper-parameters that can be specified.

There are many adversarial attack algorithms available, thus we have only chosen the ones which are considered SOTA - TextFooler [40] and DeepWordBug [35]. Our proposed attack will be compared to these algorithms.

The hyper-parameters of both of these algorithms are set to match the original configuration from their respective papers [40] [35]. The parameters of our combined adversarial attack are the following:

1. **Stop words** - stop words are not ignored, they can be modified and ranked just like the other words.
2. **Char insert, delete, swap** - all the characters in a word can be swapped, deleted, including the first and last characters. The same applies to the char insertion, where a random char can be inserted to a random position in a word.
3. **KNN** - the k-nearest neighbors of a word in embedding space; set to 10.

4. **Cosine distance threshold** - cosine distance between the original and the modified sentence has to exceed the threshold value 0.95 to be considered successful.

For the implementation of our attack, TextAttack framework [48] was used, which enabled us to reproduce the results of the SOTA adversarial attacks. I have modified this framework to meet the requirements of our use case. Finally, I have added our similarity metric, attack, and other necessary functions to the framework.

5.3.4 Automatic Evaluation

To compare adversarial examples, we need to define means to measure the performance of the attacks. Similarly to metric evaluation, the most important metric here is also accuracy.

First, we determine the accuracy of the original victim model and then we measure the accuracy of the model on our adversarial examples. This is called an accuracy after-attack. We also measure the time needed to generate an adversarial example. The last important indicator of the overall success is the modification rate, which is defined as the percentage of tokens that had to be modified to create a successful attack.

These 4 main factors were measured and used for adversarial attack evaluation are:

1. **Accuracy** - the number of correctly classified examples
2. **Accuracy after-attack** - the number of correctly classified examples, when model is fed with adversarial examples
3. **Time** - the average time needed to create one adversarial example
4. **Modification rate** - the percentage of modified tokens

Throughout this work, we also use the term attack success rate, which refers to the number of successful attacks. This can also be calculated from the after-attack and the original accuracy.

5.3.5 Human evaluation

From the previous section, it may seem that we have everything needed to evaluate the performance of our adversarial attack. Adversarial attacks are considered successful if they preserve the semantics of the original sentence and

manage to change the label of a classifier. However, as we already know, the meaning of preservation is a very complex topic and is very hard to determine automatically. After all, sometimes even multiple human annotators cannot uniformly agree on one example.

This is why we have proposed to include an extra step of evaluation - human evaluation. During this process, we performed a human analysis of all three adversarial attacks - TextFooler [40], DeepWordBug [35] and our combined attack. These algorithms attacked the fine-tuned BERT [23] model on three datasets. Overall, we generated 100 attacks on each dataset for each attack, thus resulting in 900 adversarial examples in total.

These sentence pairs were then presented to 3 annotators, who assigned each pair a binary label stating whether the example preserved its meaning or not. This evaluation process was performed in the same way as the one described for DCUS dataset creation in Section 5.2.1. Based on this evaluation, we were able to measure the real success rate of the attacks.

Chapter 6

Results

At this point, we assume the reader is familiar with the theoretical background, related work, method, and experiments that have all been explained previously. In this chapter, we build on these prerequisites and present the results of the experiments described in the previous chapter.

The following results will be presented. The first Section 6.1 describes the results obtained throughout the development and evaluation of the metric using SPE on various datasets. Then in Section 6.2, we evaluate our adversarial attack and compare it with the TextFooler [40] and DeepWordBug [35] baseline. In addition, the results of the human evaluation of adversarial attacks are presented.

6.1 Metric Development and Evaluation

The scope of this section includes the results obtained throughout the development of the metric and we will introduce them chronologically. Firstly, we will describe the results obtained on the datasets that were used to train our fastText classifiers, then the results of the experiments that were performed to build similarity metric based on SPE. Finally, our metric is evaluated on predefined tasks and compared to the metric based on USE [39] with DAN and Transformers.

6.1.1 FastText Models Evaluation

Our metric uses Semantics-Preserving-Encoder, which implements a set of fastText classifiers [65]. These were trained on seven datasets 5.2.1 with the hyper-parameters stated previously. Each dataset consists of training and testing subsets, which allows us to test the classifiers on the dataset once the

training stage is finished. The final accuracy rate obtained on the testing subset of each dataset is presented in Table 6.1.

Classifier	Test accuracy rate ↑
SNLI	0.595
COLA	0.686
RTE	0.563
SST2	0.829
StackOverflow	0.891
Emotion	0.898
Yelp Review Polarity	0.957

Table 6.1: Accuracy rate of the fastText classifiers used in SPE on the test set for each dataset.

Based on the results in Table 5.2, we can conclude that better results are achieved on datasets that are more down-stream, such as Emotion, Yelp Review Polarity, or Stack Overflow, where the training was very successful in terms of accuracy.

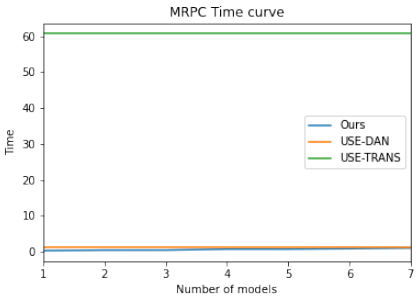
However, more general tasks such as NLI resulted in a much lower accuracy rate. My explanation is that these problems are very abstract and require a deep language understanding, which we are still unable to reproduce artificially at this point. But overall, even on the less successful tasks, our accuracy is very reasonable. Especially given the fact that we have 3 labels - entailment, contradiction, and neutral, we are still more than two times better than a random classifier.

6.1.2 Fine-tuning Our SPE for the Similarity Metric

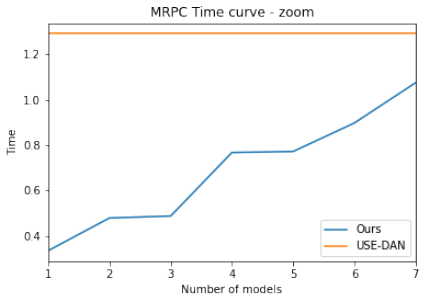
Due to the character of our Semantics-Preserving-Encoder, which combines multiple classifiers, some parameters had to be determined experimentally. We have experimented with the number of classifiers and which ones to select to produce the best results.

Overall, we have trained 30 classifiers that we experimented with and determined whether to include them or not. The main focus of the experiments was to observe how the number of models used influences the time needed to score a particular dataset and the performance in the means of accuracy. In both approaches, we have tested these qualities across different datasets where the similarity metric with our SPE was compared to USE with DAN and Transformers models.

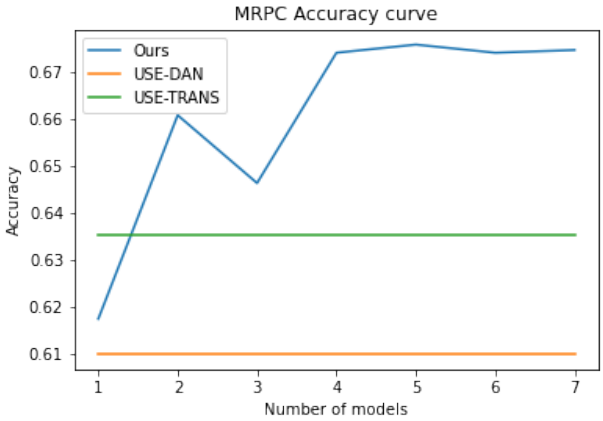
Because of the large scale of the conducted experiments on this topic, we have decided to only include an illustrative example of the results of the experiment on MRPC [69] dataset visualized in Figure 6.1. Based on the



(a) : Time curve



(b) : Time curve - zoomed in



(c) : Accuracy curve

Figure 6.1: Graphs show how the number of models impacts the time and accuracy on MRPC dataset for our metric, USE-DAN, and USE-TRANS metrics.

overall results, we have determined that having exactly 7 classifiers leads to high-quality robust results across all datasets.

First, in Figures 6.1a and 6.1b, we experimented with the number of models used in SPE (x-axis) and how it influences the time needed to score the dataset (y-axis) when it is used in a similarity metric. From Figure 6.1a we can see that our solution is much faster than the one using USE with Transformers. When we zoom in on the same graph in Figure 6.1b, we can see how our metric compares to the metric using USE with DAN in more detail. It is clear that when the number of classifiers increases, the time increases as well. In conclusion, with the finally selected 7 classifiers, we are still faster than both Universal-Sentence-Encoder algorithms.

Secondly, in Figure 6.1c we experimented with the number of models used in SPE (x-axis) and how it influences the performance of the similarity metric, specifically the accuracy (y-axis) in comparison with USE. In this particular dataset, we can see that having just 4 classifiers would be sufficient. However, the results on the other datasets have shown that only 4 classifiers

are insufficient, which is why all 7 classifiers are needed. Overall on the MRPC [69] dataset, USE with DAN is faster than USE with Transformers, but achieves worse results, while our approach is both the fastest and the most accurate with 7 classifiers.

Instinctively, the addition of more classifiers could increase the performance on some of the tasks. On the other hand, it could decrease for the others at the same time. After numerous experiments, the most robust and accurate combination of the classifiers is the one described in Section 6.1.1. However, I strongly believe more accuracy is achievable.

6.1.3 Similarity Metric Evaluation

Finally, we compare the similarity metric using our SPE against the similarity metric using USE-DAN and USE-Transformes [39], which are both used in adversarial attacks to measure the sentence similarity. The metric was evaluated in terms of accuracy rate on 5 datasets in total, 3 of them are our own. Results are shown in Table 6.2.

The results show that our metric performs better than both USE based metric approaches on all the datasets. For the NLI tasks - MRPC and SICK-Adjusted, the difference in performance between our metric and the next best performing one is 4% on MRPC and 17% on SICK-Adjusted. Results on the other 3 datasets with contradictory and unrelated sentence pairs (DCUS) also show this improvement in accuracy - around 8% on average.

Dataset	USE-DAN	USE-TRANS	SPE (ours)
MRPC	60.9	63.5	**67.4**
SICK-Adjusted	39.1	38.1	**55.2**
DCUS-RT *(ours)*	57.7	60.4	**65.9**
DCUS-Offensive *(ours)*	53.7	62.8	**64.4**
DCUS-Hate *(ours)*	54.9	54.9	**71.4**

Table 6.2: Results of similarity metric using SPE on each dataset measured in accuracy (percentage) in comparison with USE.

I believe that even better results can be achieved if we elaborated the fine-tuning process even more. Selecting different new classifiers or their combination with the currently used ones could produce even better results.

It is important to highlight that similarity metrics using our SPE or USE are not fine-tuned on any of these datasets. They are both general techniques how to measure semantics similarity between the sentences. Much higher accuracy can be achieved on any of these datasets with a specific classifier for each task, but it would not be as robust as our metric.

6.2 Adversarial Attacks Evaluation

This section demonstrates how our adversarial attack stands in comparison with the SOTA approaches, specifically TextFooler [40] and DeepWordBug [35]. To give a complex outline of the performance in the context of semantic similarity, our attack is evaluated both automatically in Section 6.2.1 and manually by human judges in Section 6.2.2.

6.2.1 Automated Evaluation

Rotten Tomatoes [60]				
Model	Original acc [%] ↑	After-attack acc [%] ↓	Time [s]↓	Mod. rate [%] ↓
TextFooler[40]	99	25	112	**16.6**
Deepwordbug[35]	99	**8**	88.7	22.6
Our attack	99	9	**55**	23.7

Offensive language [74]				
Model	Original acc [%] ↑	After-attack acc [%] ↓	Time [s]↓	Mod. rate [%] ↓
TextFooler[40]	90	35	125.2	17.5
Deepwordbug[35]	90	44	96.4	**17.3**
Our attack	90	**31**	**69.9**	23.9

Hate speech [73]				
Model	Original acc [%] ↑	After-attack acc [%] ↓	Time [s]↓	Mod. rate [%] ↓
TextFooler[40]	93	36	127.9	**14.5**
Deepwordbug[35]	93	38	109.4	16.5
Our attack	93	**21**	**67.6**	25.1

Table 6.3: Results of our attack on 3 datasets in total in comparison with TextFooler [40] and DeepWordBug [35]. Performance is measured in original accuracy (Original acc), after-attack accuracy (After-attack acc), time (Time), and modification rate (Mod. Rate). Bold font indicates the best performance for each metric. All numbers are reported on 100 test instances. Symbols ↑ (↓) represent that the higher (lower) the better.

First, we have performed an automated evaluation of the results of our attack in comparison with TextFooler [40] and DeepWordBug [35] on 3

datasets. As mentioned previously, our attack implements the new metric and combines the char level and word level attack.

The results are shown in Table 6.3, where we measure the overall performance using the success rate of the fine-tuned BERT model [23] (Original Accuracy), accuracy after-attack (After-attack acc), the time needed to perform the attack (Time) and the modification rate (Mod. rate).

From the results in Table 6.3, we can see that our attack outperforms the other approaches in a majority of the measured parameters.

The accuracy after-attack is in most cases by far the best, in other cases close to the best. We can imagine that the process behind the after-attack accuracy is feeding the classifier with our adversarial examples and observing how the accuracy rate drops, which is expressed by the after-attack accuracy, where ideally we want it to be very low.

Another factor to be evaluated was the time it took to carry out the attack. Our attack is the fastest, generally approximately 1.5-2× faster than TextFooler [40] or DeepWordBug [35]. This is mainly due to the speed of our SPE in the similarity metric, which is used in the attack.

Looking at the modification rate, we can assume that our attack performs more modifications than others. But the difference is not that significant, especially when we take into consideration that DeepWordBug [35] attacks only on char level.

Overall, our attack has been proven successful and better than the compared SOTA approaches. The only factor in which it has been slightly worse is the modification rate. However, this does not imply that the quality of the attack itself is worse in terms of output quality or semantic similarity. This should be better determined by the human evaluation in the next section.

6.2.2 Human Evaluation

The results of the human evaluation are shown in Figure 6.4 together with the results of the automated evaluation from Table 6.3. The human evaluation results were included in the computation of the real after-attack accuracy in the rightmost column (Real after-attack acc). Thanks to human evaluation, we are able to detect some attacks as false positives and get the final real after-attack accuracy rate. We can then evaluate how the after-attack accuracy has improved compared to the real after-attack accuracy.

Based on the results, we can conclude that our attack using SPE in the similarity metric produces better quality attacks than the existing SOTA approaches, achieving the best accuracy rate across all tasks. However, many adversarial examples produced by our method were still marked false, meaning

Rotten Tomatoes [60]					
Model	Original acc [%] ↑	After-attack acc [%] ↓	Time [s]↓	Mod. rate [%] ↓	Real after-attack acc [%] ↓
TextFooler [40]	99	25	112	**16.6**	45
Deepwordbug [35]	99	**8**	88.7	22.6	40
Our attack	99	9	**55**	23.7	**37**

Offensive language [74]					
Model	Original acc [%] ↑	After-attack acc [%] ↓	Time [s]↓	Mod. rate [%] ↓	Real after-attack acc [%] ↓
TextFooler [40]	90	35	125.2	17.5	56
Deepwordbug [35]	90	44	96.4	**17.3**	74
Our attack	90	**31**	**69.9**	23.9	**52**

Hate speech [73]					
Model	Original acc [%] ↑	After-attack acc [%] ↓	Time [s]↓	Mod. rate [%] ↓	Real after-attack acc [%] ↓
TextFooler [40]	93	36	127.9	**14.5**	54
Deepwordbug [35]	93	38	109.4	16.5	51
Our attack	93	**21**	**67.6**	25.1	**45**

Table 6.4: Overall results of our attack from both automated and human evaluation. Performance is measured in original accuracy (Original acc), after-attack accuracy (After-attack acc), time (Time), modification rate (Mod. rate), and real after-attack accuracy (Real after-attack acc). Bold font indicates the best performance for each metric. All numbers are reported on 100 test instances. Symbols ↑ (↓) represent that the higher (lower) the better.

that our solution still carries some of the issues other SOTA approaches are currently facing.

Chapter 7

Discussion

In this chapter, I would like to further discuss the results of Chapter 6, highlight the general aspects of our metric and attack and suggest potential improvements that could be explored in the future. First, these points will be discussed for the metric using Semantics-Preserving-Encoder, then for the adversarial attack.

7.1 Similarity Metric Using SPE

To recapitulate, our hypothesis was that the problem of false positive adversarial attacks lies in the semi-supervised technique of Universal-Sentence-Encoder (USE) [39]. To improve this, I have presented a new technique called Semantics-Preserving-Encoder (SPE), which stands on multiple supervised fastText classifiers [65].

We have shown that our metric, which uses SPE instead of USE [39], performs better on various datasets. The difference between these two encoders is even more significant on the more specific datasets, which is observed on our datasets with contradictory and unrelated sentence pairs (DCUS). On all three DCUS datasets our metric with SPE is able to differentiate between false positives adversarial attacks more than 10-20% better when compared to the metric with USE [39]. This result is even more substantial considering the simple and fast approach of our concept.

The results on DCUS datasets show that USE [39] falls behind in terms of adaptation to more specific tasks. We can imply that, in this case, the universality of USE [39] is a disadvantage. Reflecting on the real-world application, attacks are usually aimed at a specific classifier, e.g. hate speech, offensive language, or spam. Therefore, having a more specific sentence embedding, such as our SPE, that is suited specifically for the relevant

platform is advantageous and allows for better detection of the semantically incorrect adversarial examples.

Although our approach largely improves the existing semantics metric, it is important to note that for a different cosine threshold different results may be obtained. As mentioned previously, the threshold we used was based on the suggested optimum in paper [9] with respect to the quality of the adversarial examples.

As demonstrated in Figure 6.1, the addition of more classifiers increases the time complexity. The asymptotic complexity of SPE is $\mathcal{O}(n)$, same as USE with DAN, while for USE with Transformers it is $\mathcal{O}(n^2)$ [39]. However, looking at the real-time complexity, a significant difference can be observed between our SPE and USE, where SPE is able to vastly reduce the computation complexity. The first aspect is the size of the hidden layer, which affects the matrix dimensions and the complexity of matrix multiplication. While USE has a hidden layer of size 512 [39], SPE only has a size 10. Moreover, our SPE only needs to perform 7 matrix multiplication operations with small matrices, in contrast, to USE [39], which needs to perform these operations with 512-dimensional vectors and therefore very large matrices. The final time complexity of one step is defined as follows:

$$O = \sum_{i=1}^{7} V_i * H_i \tag{7.1}$$

where V_i is the vocabulary size for i-th classifier and H_i is the size of the hidden layer for i-th classifier.

Even though the results of SPE in terms of time complexity are already substantial, I believe there is still room for improvement. The time complexity of SPE could be further reduced if we created just one classifier from the seven currently used. This could be achieved by merging the 7 datasets, on which the currently used classifiers were built, together into one, which would then be trained. This would allow us to minimize the matrix multiplication to just a single operation and to add new models easily. The model addition process would be independent of the overall time complexity of the metric, because we would only need to rebuild and retrain the single classifier with the new model included.

7.2 Adversarial Attack

I have presented a new adversarial attack, which builds on the metric with SPE and combines word and char level modification, similarly to TextBugger [36]. We expected the attack to produce adversarial examples of better quality than attacks implementing USE [39] and to be faster because of SPE.

In the end, this assumption was correct, our attack was indeed the fastest, produced the most accurate results, and had more successful attacks overall across all three datasets when evaluated automatically or by humans. The only downside is that more modifications were needed in our attack when compared to others.

For further improvement of the adversarial attack, the idea to modify SPE to only use one classifier as mentioned in the previous Section 7.1 also applies here, because decreasing the time complexity of SPE would also lead to a faster attack. Another suggestion could be to add some more complex modifications to the text, such as word deletion or merging of multiple words.

Chapter 8

Conclusion

The goal of this thesis was to develop a new similarity metric, which will tackle the problem of false positive adversarial attacks, and build a new adversarial attack using this metric. I have extensively researched state-of-the-art text classifiers and their properties and identified problems in adversarial attacks. Building on that, I propose a new solution aimed at overcoming these problems.

In the first part of this work, the fundamentals of this topic are outlined, which are essential for the reader's understanding. Specifically, classification problems in NLP and neural network models are explained in detail. Then, sentence and word representation and various word and sentence embedding techniques are shown. Finally, an analysis and an overview of the existing adversarial attacks are presented.

In Chapter 4, I propose a new sentence embedding technique called SPE - Semantics Preserving Encoder, which is designed to tackle the problem of false positive adversarial examples, by producing better vectors in the latent space. This technique is then used in the similarity metric as a substitution for USE [39]. I also describe our new adversarial attack, which combines both character and word level modifications, similar to [35].

Once the concept of the proposed metric using SPE and the adversarial attack are explained, various experiments are constructed in the following Chapter 5 to evaluate the performance of our techniques with others. To ensure that the metric evaluation is performed on a diverse set of data, I have developed 3 new datasets (DCUS) containing more than 500 sentence pairs that are used for this evaluation. This chapter also includes information about the hyper-parameters and setup that were used throughout and an in-depth explanation of both the automated and human evaluation of the adversarial attack.

In the following Chapter 6, I describe the results achieved throughout the

training of over 30 fastText classifiers [65], out of which the 7 best were implemented in SPE. Training results were the determining factor for which and how many of these classifiers to use in our SPE, which is described in the fine-tuning section. Then, the overall results of the similarity metric evaluation are presented. Compared with USE, which is commonly used in the similarity metric for adversarial attacks, the performance of our approach is significantly better both in time complexity and in accuracy on various datasets for the given cosine threshold. Moreover, our sentence embeddings are also only 10 dimensional, whereas USE uses 512-dimensional vectors.

The second half of the results focuses on the attack evaluation. First, we have performed an automated evaluation. Compared to SOTA attacks like TextFooler [40] and DeepWordBug [35], we achieve a higher attack success rate and much less time needed to perform the attack. However, our modification rate is higher by a few percentages.

Then, the evaluation of the produced adversarial examples was performed by human judges. This has shown that some of these adversarial examples are still false positives, however, our adversarial attack ranks the best, maintaining the highest attack success rate.

I believe that the proposed concept has great potential and shows promising results. In the future, it would be interesting to explore more diverse classifiers, which could further increase the accuracy of SPE. Also, the idea proposed in Chapter 7 could be implemented to decrease the calculation time of the sentence embedding.

In summary, the new metric and adversarial attack have been proven successful in various tasks. They are on par with existing solutions, and in some cases, even outperform them. I believe that I have fully met the thesis objectives and also contributed to the machine learning community. Thanks to the promising results, I plan to summarize my achievements and findings in a scientific paper.

Appendix A

Bibliography

[1] Marchand A, Marx P. Automated product recommendations with preference-based explanations. J Retail. 2020;96(3):328–43.

[2] Bhowmick, A., Hazarika, S. M. (2016). Machine learning for e-mail spam filtering: review, techniques and trends. arXiv preprint arXiv:1606.01042.

[3] Chiu C-C, Sainath TN, Wu Y, Prabhavalkar R, Nguyen P, Chen Z, Kannan A, Weiss RJ, Rao K, Gonina E, et al. State-of-the-art speech recognition with sequence-to-sequence models. In: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018 pages 4774–4778. IEEE .

[4] Ahmad, T., Truscan, D., Vain, J., Porres, I. (2022). Early Detection of Network Attacks Using Deep Learning. arXiv preprint arXiv:2201.11628.

[5] Fujiyoshi H, Hirakawa T, Yamashita T. Deep learning-based image recognition for autonomous driving. IATSS Res. 2019;43(4):244–52.

[6] Eykholt, Kevin, et al. "Robust physical-world attacks on deep learning visual classification." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

[7] Yang G, Li M, Fang X, Zhang J, Liang X. Generating adversarial examples without specifying a target model. PeerJ Comput Sci. 2021;7:e702. Published 2021 Sep 13. doi:10.7717/peerj-cs.702

[8] Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." arXiv preprint arXiv:1412.6572 (2014).

[9] Morris, John X., et al. "Reevaluating adversarial examples in natural language." arXiv preprint arXiv:2004.14174 (2020).

[10] Charu C. Aggarwal and ChengXiang Zhai. Mining Text Data. Springer New York, 2012. ISBN 9781461432234.

[11] Miner, G., Delen, D., Elder, J., Fast, A., Hill, T., Nisbet, R. A. (2012). Chapter 17 - Summary To Practical Text Mining and Statistical Analysis for Non-structured Text Data Applications (1007–1016). doi:10.1016/B978-0-12-386979-1.00045-1

[12] McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics. 5 (4): 115–133. doi:10.1007/BF02478259

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. MIT Press. http://www.deeplearningbook.org

[14] Bottou, Léon (1998). "Online Algorithms and Stochastic Approximations". Online Learning and Neural Networks. Cambridge University Press. ISBN 978-0-521-65263-6.

[15] Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function Mathematics of Control, Signals, and Systems, 2(4), 303–314.

[16] Rosenblatt, Frank. x. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books, Washington DC, 1961

[17] Elman, Jeffrey L. (1990). "Finding Structure in Time". Cognitive Science. 14 (2): 179–211. doi:10.1016/0364-0213(90)90002-E

[18] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2), 157–166.

[19] Mikolov, T. (2012). Statistical Language Models based on Neural Networks. Ph.D. thesis, Brno University of Technology.

[20] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.

[21] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

[22] Harris, Zellig (1954). "Distributional Structure". Word. 10 (2/3): 146–62. doi:10.1080/00437956.1954.11659520. And this stock of combinations of elements becomes a factor in the way later choices are made ... for language is not merely a bag of words but a tool with particular properties which have been fashioned in the course of its use

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics

[24] Raffel, Colin, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." arXiv preprint arXiv:1910.10683 (2019).

[25] Brown, Tom, et al. "Language models are few-shot learners." Advances in neural information processing systems 33 (2020): 1877-1901.

[26] Jurafsky, Daniel; H. James, Martin (2000). Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition. Upper Saddle River, N.J.: Prentice Hall. ISBN 978-0-13-095069-7.

[27] Tomáš Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In Adv. NIPS.

[28] Greene, William H. (2012). Econometric Analysis (Seventh ed.). Boston: Pearson Education. pp. 803–806. ISBN 978-0-273-75356-8.

[29] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daume III. 2015. Deep unordered compo- sition rivals syntactic methods for text classification. In Proceedings of ACL/IJCNLP.

[30] D. E. Rumelhart, G. E. Hinton, R. J. Williams. Learning internal representations by back-propagating errors. Nature, 323:533.536, 1986.

[31] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[32] Bowman, Samuel R., et al. "A large annotated corpus for learning natural language inference." arXiv preprint arXiv:1508.05326 (2015).

[33] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. 2019. Adversarial examples: Attacks and defenses for deep learning. IEEE transactions on neural networks and learning systems, 30(9):2805–2824.

[34] Wei Emma Zhang, Quan Z Sheng, Ahoud Alhazmi, and Chenliang Li. 2020. Adversarial attacks on deep-learning models in natural language processing: A survey. ACM Transactions on Intelligent Systems and Technology (TIST), 11(3):1–41.

[35] Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers. In 2018 IEEE Security and Privacy Workshops (SPW), pages 50–56. IEEE.

[36] Li, Jinfeng, et al. "Textbugger: Generating adversarial text against real-world applications." arXiv preprint arXiv:1812.05271 (2018).

[37] Steffen Eger, Gozde G ¨ ul S¸ ahin, Andreas R ¨ uckl ¨ e, Ji´Ung Lee, Claudia Schulz, Mohsen Mesgar, Krishnkant Swarnkar, Edwin Simpson, and Iryna Gurevych. 2019. Text processing like humans do: Visually attacking and shielding NLP systems. In Proceedings of NAACL-HLT

[38] Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020. Bert-attack: Adversarial attack against bert using bert. In Proc. of EMNLP

[39] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. arXiv preprint arXiv:1803.11175

[40] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is BERT really robust? A strong baseline for natural language attack on text classification and entailment. In The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, pages 8018–8025. AAAI Press

[41] Nikola Mrkšic, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašic, Lina M. Rojas-Barahona, Pei Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016. Counter-fitting word vectors to linguistic constraints. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 142–148, San Diego, California. Association for Computational Linguistics.

[42] Dianqi Li, Yizhe Zhang, Hao Peng, Liqun Chen, Chris Brockett, Ming-Ting Sun, and Bill Dolan. 2021. Contextualized perturbation for textual adversarial attack. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 5053–5069, Online. Association for Computational Linguistics.

[43] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

[44] Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 1875–1885, New Orleans, Louisiana. Association for Computational Linguistics

[45] Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In Proceedings of EMNLP.

[46] Zhengli Zhao, Dheeru Dua, and Sameer Singh. 2018. Generating natural adversarial examples. In Proceedings of ICLR

[47] Siddhant Garg and Goutham Ramakrishnan. 2020. Bae: Bert-based adversarial examples for text classification. In Proc. of EMNLP.

[48] John Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. TextAttack: A framework for adversarial attacks, data augmentation, and adversarial training in NLP. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 119–126, Online. Association for Computational Linguistics.

[49] Guoyang Zeng, Fanchao Qi, Qianrui Zhou, Tingji Zhang, Zixian Ma, Bairu Hou, Yuan Zang, Zhiyuan Liu, and Maosong Sun. 2021. OpenAttack: An open-source textual adversarial attack toolkit. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations, pages 363–371, Online. Association for Computational Linguistics.

[50] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In Proceedings of ICLR.

[51] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016a. The limitations of deep learning in adversarial settings. In 2016 IEEE European symposium on security and privacy (EuroS&P). IEEE.

[52] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary,Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating natural language adversarial examples. In Proceedings of the EMNLP.

[53] Danish Pruthi, Bhuwan Dhingra, and Zachary C. Lipton. 2019. Combating adversarial misspellings with robust word recognition. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 5582–5591, Florence, Italy. Association for Computational Linguistics

[54] Aminul Islam and Diana Inkpen. 2009. Real-word spelling correction using Google Web 1T 3-grams. In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, pages 1241–1249, Singapore. Association for Computational Linguistics

[55] Keisuke Sakaguchi, Matt Post, and Benjamin Van Durme. 2017. Grammatical error correction with neural reinforcement learning. In Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers), pages 366–372, Taipei, Taiwan. Asian Federation of Natural Language Processing

[56] Xinshuai Dong, Hong Liu, Rongrong Ji, and Anh Tuan Luu. 2021. Towards robustness against natural language word substitutions. In International Conference on Learning Representations (ICLR).

[57] Chenglei Si, Zhengyan Zhang, Fanchao Qi, Zhiyuan Liu, Yasheng Wang, Qun Liu, and Maosong Sun. 2020. Better robustness by more coverage: Adversarial training with mixup augmentation for robust fine-tuning. arXiv preprint arXiv:2012.15699.

[58] Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. 2020. Freelb: Enhanced adversarial training for natural language understanding. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020

[59] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In Advances in neural information processing systems, pages 649–657.

[60] Pang, B., & Lee, L. (2005). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. arXiv preprint cs/0506075.

[61] McAuley, Julian, and Jure Leskovec. "Hidden factors and hidden topics: understanding rating dimensions with review text." In Proceedings of the 7th ACM conference on Recommender systems, pp. 165-172. 2013.

[62] Bowman, Samuel R., et al. "A large annotated corpus for learning natural language inference." arXiv preprint arXiv:1508.05326 (2015).

[63] Cer, Daniel, et al. "Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation." arXiv preprint arXiv:1708.00055 (2017).

[64] Thomas Wolf , LysandreDebut, VictorSanh, JulienChaumond, Clement-Delangue, AnthonyMoi, PierricCistac, TimRault, R'emiLouf, Morgan-Funtowicz, and JamieBrew. 2019. HuggingFace's Transformers: State-of-the-art natural language processing. arXiv preprint arXiv:1910.03771.

[65] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. arXiv preprint 1607.01759, 2016.

[66] Bojanowski, Piotr; Grave, Edouard; Joulin, Armand; Mikolov, Tomas (2017-06-19). "Enriching Word Vectors with Subword Information". arXiv:1607.04606

[67] G. Rawlinson, "The significance of letter position in word recognition," IEEE Aerospace and Electronic Systems Magazine, vol. 22, no. 1, pp. 26–27, 2007.

[68] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in EMNLP, 2014, pp. 1532–1543.

[69] Wang, Alex, et al. "GLUE: A multi-task benchmark and analysis platform for natural language understanding." arXiv preprint arXiv:1804.07461 (2018).

[70] Saravia, Elvis et al. “CARER: Contextualized Affect Representations for Emotion Recognition.” EMNLP (2018).

[71] Annamoradnejad, I., Habibi, J. & Fazli, M. Multi-view approach to suggest moderation actions in community question answering sites. *Information Sciences*. **600** pp. 144-154 (2022), https://www.sciencedirect.com/science/article/pii/S0020025522003127

[72] Marelli, M., Menini, S., Baroni, M., Bentivogli, L., Bernardi, R., and Zamparelli, R. (2014). A SICK cure for the evaluation of compositional distributional semantic models. In Proceedings of LREC.

[73] Basile, V., Bosco, C., Fersini, E., Nozza, D., Patti, V., Rangel Pardo, F., Rosso, P. & Sanguinetti, M. SemEval-2019 Task 5: Multilingual Detection of Hate Speech Against Immigrants and Women in Twitter. *Proceedings Of The 13th International Workshop On Semantic Evaluation*. pp. 54-63 (2019), https://www.aclweb.org/anthology/S19-2007

[74] Zampieri, M., Malmasi, S., Nakov, P., Rosenthal, S., Farra, N. & Kumar, R. SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media (OffensEval). *Proceedings Of The 13th International Workshop On Semantic Evaluation*. pp. 75-86 (2019)

[75] Joulin, Armand, et al. "Fasttext. zip: Compressing text classification models." arXiv preprint arXiv:1612.03651 (2016).

Appendix B

Shortcuts

ADAM: Adaptive Moment Estimation
BERT: Bidirectional Encoder Representations from Transformers
BoW: Bag of Words
BPTT: Backpropagation through time
CBOW: Continuous bag of words
CV: Computer Vision
DAN: Deep Averaging Network
DCUS: Datasets with contradictory and unrelated sentence pairs
DNN: Deep Neural Networks
GPT-3: third generation Generative Pre-trained Transformer
LSTM: Long-short term memory
MLM: Masked Language Model
NN: Neural Network
NLI: Natural Language Inference
NLP: Natural Language Processing
NSP: Next Sentence Prediction
RNN: Recurrent Neural Network
SGD: Stochastic Gradient Descent
SOTA: state-of-the-art
SPE: Semantic-Preserving-Encoder
T5: Text-to-Text Transfer Transformer
USE: Universal-Sentence-Encoder

Appendix C

DCUS Examples with Similarity Scores

I would like to show some examples of the sentences from our DCUS datasets and how our proposed similarity metric using SPE can better distinguish between the contradictory and similar sentences than the metric with USE [39].

If the semantics is preserved, the cosine distance between the original and perturbed sentences should be close to high. If semantics is not preserved then the cosine distance should be low.

DCUS-RT

original: "bears is even worse than i imagined a movie ever could be ." *perturbed:* "bears is even strongest than i imagined a movie ever could be ."	*semantics:* preserved *our_cos_dist* = 0.92 *use_dan_cos_dist* = 0.57 *use_trans_cos_dist* = 0.83
original: "more trifle than triumph" *perturbed:* "more trifle than victoria"	*semantics:* preserved *our_cos_dist* = 0.99 *use_dan_cos_dist* = 0.61 *use_trans_cos_dist* = 0.61
original: "one of the worst movies of the year....watching it was painful" *perturbed:* "one of the strongest panorama of the enjoyment....watching it was poignant"	*semantics:* not preserved *our_cos_dist* = 0.23 *use_dan_cos_dist* = 0.35 *use_trans_cos_dist* = 0.28

DCUS-Offensive

original: "sex with strangers is fascinating . . ." *perturbed:* "sex with strangers is entrancing . . ."	*semantics:* preserved *our_cos_dist* = 0.93 *use_dan_cos_dist* = 0.88 *use_trans_cos_dist* = 0.93

original: "worst potus ever!" *perturbed:* "greatest potus ever!"	*semantics:* not preserved *our_cos_dist* = 0.63 *use_dan_cos_dist* = 0.77 *use_trans_cos_dist* = 0.82
original: "@user Well I mean by all means do you but not all men are like that!!!!" *perturbed:* "@user Nicer I wherewithal by all wherewithal making you but not all virile are lover that!!!!"	*semantics:* not preserved *our_cos_dist* = 0.37 *use_dan_cos_dist* = 0.59 *use_trans_cos_dist* = 0.58

DCUS-H

original: "The awkward moment when Lexus is showing you how much crack fits in the cup holders..." *perturbed:* "Both embarrassing dating when Lexus is proves you how much crack fits in the trophy landlady..."	*semantics:* not preserved *our_cos_dist* = 0.36 *use_dan_cos_dist* = 0.67 *use_trans_cos_dist* = 0.67
original: "She's a b***h... No less than a w***e." *perturbed:* "She's a vixen... Sans fewer than a nympho."	*semantics:* not preserved *our_cos_dist* = 0.25 *use_dan_cos_dist* = 0.53 *use_trans_cos_dist* = 0.57
original: "@user hell no!!!!!" *perturbed:* "@user f**k no!!!!!"	*semantics:* preserved *our_cos_dist* = 0.92 *use_dan_cos_dist* = 0.86 *use_trans_cos_dist* = 0.96