UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 2323

ADVERSARIAL ATTACKS IN NATURAL LANGUAGE PROCESSING

Josip Jukić

Zagreb, June 2020

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 2323

ADVERSARIAL ATTACKS IN NATURAL LANGUAGE PROCESSING

Josip Jukić

Zagreb, June 2020

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Zagreb, 13 March 2020

MASTER THESIS ASSIGNMENT No. 2323

Student:	**Josip Jukić (0036491111)**
Study:	Computing
Profile:	Computer Science
Mentor:	prof. Domagoj Jakobović

Title: **Adversarial Attacks in Natural Language Processing**

Description:

Describe the problem of adversarial attacks on machine learning models given the model type and capabilities of the attacker. Study the possibilities of an attack based on adversarial examples with particular emphasis on natural language processing domain. List possible attack scenarios with different levels of knowledge of the observed system. Devote special attention to the approach with limited information about the model. Define attacker's task as an optimization problem to enable the possibility of applying optimization algorithms in the attacks. Compare the effectiveness of various optimization algorithms with respect to available information, application domain and model type. Include the source codes, the obtained results with necessary explanations and the used literature with the thesis.

Submission date: 30 June 2020

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

DIPLOMSKI ZADATAK br. 2323

Pristupnik:	**Josip Jukić (0036491111)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	prof. dr. sc. Domagoj Jakobović

Zadatak: **Suparnički napadi u obradi prirodnog jezika**

Opis zadatka:

Opisati problematiku suparničkih napada na modele strojnog učenja s obzirom na vrste modela i mogućnosti napadača. Proučiti mogućnosti napada temeljenih na neprijateljskim primjerima s posebnim naglaskom na domenu obrade prirodnog jezika. Navesti moguće scenarije napada s različitim razinama poznavanja promatranog sustava. Posebnu pažnju posvetiti pristupu s ograničenim informacijama o modelima. Definirati zadatak napadača u obliku optimizacijskog problema, uz otvaranje mogućnosti primjene optimizacijskih algoritama u napadima. Usporediti učinkovitost različitih algoritama za optmizaciju s obzirom na dostupne informacije, domenu primjene i vrstu modela. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Rok za predaju rada: 30. lipnja 2020.

CONTENTS

List of Figures

LIST OF TABLES

1. Introduction

Thanks to the rapid development of machine learning algorithms and their unbelievable effectiveness, they have become prevalent among many computing systems and applications. Even though deep learning models are almost ubiquitous, there is a worrying lack of interest in defence mechanisms and security of those models. Awareness in general computer security seems to be on the rise, but deep learning continues to be the black spot. Because of the novelty and recentness of mentioned algorithms, the emphasis is put on results and models' performances rather than their robustness[1] and security.

It is quite difficult to cover all of the bases for a quality defence of deep learning models. Weaknesses sometimes appear in the most unexpected places. To grasp the scope of potential threats, studying attack techniques comes as a natural approach for that problem. By exploring invasive schemes, it is possible to pinpoint vulnerable areas and patch them up.

Due to the popularization of social networks and massive collections of forums, there has been a surge in textual data available on the Internet. This trend has been accompanied by the development of copious machine learning models targeted for text-based tasks. Large quantities of text demand more processing and analysis. Users require support for various problems regarding natural language. Many solutions are provided online in the form of application programming interfaces or just plain applications. Most of them run some kind of machine learning model in the background to solve specific tasks.

Machine translation, sentiment analysis, information extraction, question answering - these are only a few of the many problems that require the power of deep learning for automated and effective answers. Mentioned applications often handle sensitive user data. They can also make important and critical decisions based on their underlying models. That leaves those applications susceptible to security breaches and various malicious exploitations. It is essential to invest in model's protection, as well as in its robustness. Security requirements are often neglected in these products. More significant thought has been put into constructing attacks on models that predominantly work with images. The interest has been a product of

[1] characterization of how effective the model is on a new independent dataset or slightly altered data

academic curiosity rather than a real attempt to fortify the models. Development of attacks that could lead to better defences has only scratched the surface in natural language problems. It is generally harder to design attacks on textual models than on image-based ones. The reason lies in the unstructured nature of text. Consequently, attacks are also harder to detect and prevent.

Malicious techniques can seriously jeopardize deep learning models. They can be manipulated into providing specific answers for certain questions or queries. To mitigate this problem, it is necessary to analyze attacking approaches and use them as an advantage in building defence mechanisms.

Attacks don't only provide insights about security, but they can also be the key to unlocking a better understanding of used models. One of the glaring problems in deep learning is poor interpretability and black-box treatment. This means that human users know very little of the model's behaviour and they often view it as a series of inputs and outputs. What happens internally and what is the underlying logic of a certain model, remains a complete mystery. Identifying what affects model the most is a first step in revealing some of the answers. Changes in data for targeted attacks can be connected with the model's performances and thus, it can lead to a better internal image.

The main goal of this thesis is to explore the possibility of attacks in the natural language domain, with deep learning models as the most prominent targets. This focus is based on their impressive performances and increasing employment. Furthermore, extensive analysis of many attacks in varying situations is offered in order to gradually develop strong countermeasures. The research is concentrated on realistic set-ups where not much is known about the targeted models. As an additional important task, the correlation between human perception of text and word representation in natural language is also investigated. The whole process ultimately leads to developing novel attacks, mostly based on the adversarial approach. By accumulating all of the constructed attacks, it is possible to provide an assessment of the model's security and robustness. As the climax of this thesis, comes the fifth chapter with plentiful experiments and extensive analysis. Last chapters are dedicated to defence mechanisms and ultimate conclusions.

2. Natural Language Processing

Natural language processing (NLP) and computational linguistics (CL) are two areas of computational study of human language. NLP strives to construct methods for solving practical problems involving language, while CL employs computational methods to explore properties of human language (Rao and McMahan, 2019). How do we understand language? That question seems to be one of the greatest motivators in dealing with the described problems.

In the past decade, NLP has become an essential part of our daily lives. With the increase of shared information, natural language applications thrive. They are based on a combination of several areas such as algorithms, linguistics, logic and statistics (Eisenstein, 2019).

There are plentiful natural language tasks, but a deeper dive is taken into text classification and natural language inference problems. Those problem set-ups are widespread and can be utilized for plenty of specific tasks. In their essence, all of the approaches share a common logic. Models transform textual inputs into vector space to recognize as many patterns as possible. The chosen task then instructs a model to come up with answers in an appropriate form. While the bases of the models are similar, additional effort is invested to ensure that models' outputs satisfy desired needs. Additions are usually built on top of models.

Solving a natural language problem requires a breakdown into smaller steps. They are combined to make a coherent sequence called pipeline. Typical NLP pipeline is consisted of several steps:

1. language detection,
2. text cleanup (boilerplate removal, normalization),
3. **sentence segmentation**,
4. **tokenization** - breaking a text up into tokens[1],

[1] words and other meaningful elements

5. **stemming** - reduction of word-forms to stems[2],
6. **part-of-speech tagging** - determining the grammatical categories,
7. **lemmatization** - transformation of words into linguistically valid base forms called lemmas[3],
8. parsing.

Items marked in bold are considered as the basic part of the pipeline. After the execution of chosen steps, higher-level tasks can be introduced. An illustration of an NLP pipeline is shown in Figure 2.1. The process starts with unstructured texts and ends up with categorized and structured documents.

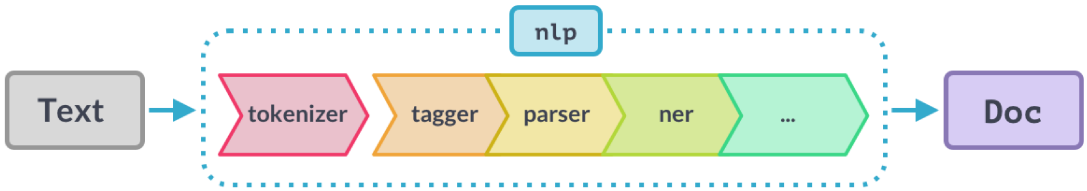

Figure 2.1: Basic NLP pipeline

2.1. Text Classification

Text classification or text categorization is a task of assigning a set of predefined categories to unstructured text. Text classifiers can be used to structure and organize various textual data. For instance, articles can be categorized by topics, electronic mail can be separated to wanted and unwanted, sentiment can be assigned to certain comments or queries, and many more.

Considering the fact that a very large portion of text available on the web is unstructured, this leaves a strong need for automated text organization. Because of it, classification is often utilized as a prerequisite for other types of problems. Many tasks require some form of categorization, and part-of-speech tagging is an example of that. It is also known as grammatical tagging or word-category disambiguation. Essentially, it represents the process of marking up a word in a text which corresponds to a particular part of speech, based on both its context and its definition. These categories are later used in other problems to achieve far

[2]form of a word before any inflectional affixes are added
[3]words in canonical or dictionary form

better results. The last example illustrates the power of text classification, how it disperses and makes an impact in many natural language tasks.

Text classification can be achieved manually or in an automated fashion. Since the former method requires human annotators and is very time-consuming as well as expensive, the latter is embraced more often. As in any other domain, there are several types of classification problems that can be distinguished:

- **binary** - there are only two distinct categories and an instance can only be assigned to one of them,
- **multi-class** - instance is classified into one of three or more classes,
- **multi-label** - multiple categories may be assigned to each instance.

The standard set-up of text classification involves unstructured textual data as input. Text is being numericalized[4] and forwarded to the model via NLP pipeline. The model produces the output which represents the prediction of categories assignment. This procedure can be put into the context of sentiment analysis, topic categorization, spam detection, etc.

2.2. Natural Language Inference

Natural language inference (NLI) or textual entailment aims to determine directional relations between text fragments. In a typical NLI framework, the entailing and entailed texts are termed *premise* (p) and *hypothesis* (h), respectively. Textual entailment is not equivalent to pure logical entailment, as a more relaxed definition is assigned to it. Typically, "t entails h" ($t \Rightarrow h$) is considered true if a human reading t would be justified to infer that h is true.

There are three distinct relations which can be assigned to a pair of texts. Relation is determined by the truthfulness of hypothesis as follows:

- **entailment** - hypothesis is true,
- **contradiction** - hypothesis is false,
- **neutral** - it cannot be determined.

Premises are considered to be fixed texts, while any number of hypotheses can be assigned to them. Each pair is observed as a separate example. Despite the loose definition, the judgement must be derived only if the facts in the premise necessarily imply all of the facts in the hypothesis. For example, if the hypothesis is a conjunction of several statements,

[4] quantified or represented with numbers

every single statement must be implied from its premise for it to be true. To acquire familiarity with textual entailment, some examples from Bowman et al. (2015) are displayed in Table 2.1. An instance of the described situation with the hypothesis in a conjunction form can be seen in the second example.

Table 2.1: Textual entailment examples

Premise	Judgement	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	**contradiction**	The man is sleeping.
An older and younger man smiling.	**neutral**	Two men are smiling and laughing at the cats playing on the floor.
A black race car starts up in front of a crowd of people.	**contradiction**	A man is driving down a lonely road.
A soccer game with multiple males playing.	**entailment**	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	**neutral**	A happy woman in a fairy costume holds an umbrella.

3. Attacks in Machine Learning

Algorithms and models in deep learning live off data since the ultimate goal is to find patterns in learning examples. Judging by the current situation, data sources seem to be virtually inexhaustible, thus only amplifying the need for their usage. Thriving in those trends, machine learning has become an integral part of large systems which completely automate the process of decision-making. Endangering those systems can lead to disastrous consequences. Computer-aided diagnostic medicine is vulnerable to this kind of attacks, as well as autonomous vehicles. This poses a serious threat in mentioned cases and other critical, life dependent decisions made by machines.

Attacks often target data that are used in the learning process. By compromising the data, the whole scheme is brought into question. Suddenly, the model can become corrupted with suspicious examples that are designed to trick the model.

The attacking algorithm is called the *threat model*, and the target is referred to as the *victim model*.

3.1. Attack Types

On a high level of abstraction, attacks on machine learning models can be distributed to three main subgroups that are listed hereafter.

- **Adversarial inputs** are specially designed to trick the model while making a decision. In a classification task, their goal is to cause a false prediction without alarming the system. Adversarial examples have to be stealthy and they cannot deviate too much from the usual data. This type of attack can be often seen in action: malicious documents for avoiding antivirus programs, electronic mail that seeks to bypass filters of unwanted messages, etc.
- **Data poisoning** is based on the technique of adversarial inputs, but it attacks the model during its learning phase, providing the examples to tear down the model's

confidence. Inserting poisoned data into a train set results in shifting the decision boundary, ultimately leading to a completely useless model. Poisoning attacks can focus on two main purposes: denial of service by harming the availability or destroying the model's integrity. The former is achieved by infiltrating large quantities of corrupted data, while the latter uses more sophisticated methods. Some of the approaches consider inserting a backdoor[1] which the model's architect is not aware of. For instance, the attacker can instruct a model for detecting malicious documents to declare a file harmless if it contains a specific string. This leaves a possibility to dump malicious code along with the chosen all-powerful string.

- **Model stealing** represents duplicating a model or reconstruction of a train set (if one is kept covert). This approach is often exercised by black-box probing. Stolen models can be used directly or as proxies in other attacks, with them providing guidance in the optimization process for attack execution.

Figures 3.1 and 3.2 illustrate the difference between adversarial examples and data poisoning. In a classical adversarial attack, an instance from the test set is being directly modified. On the other hand, poisoning interferes with the model's test set, making it alter its decision boundary. In summary, the adversarial approach provides a simple way to the bypass model's defence mechanisms. The downfall for the attacker is the requirement of owning the test data. Conversely, data poisoning requires owning the training data but enables a broader spectrum of attacks.

It is possible to combine the two discussed approaches. Infiltrating in data during the model's training stages allows even simpler crafting of adversarial inputs later on.

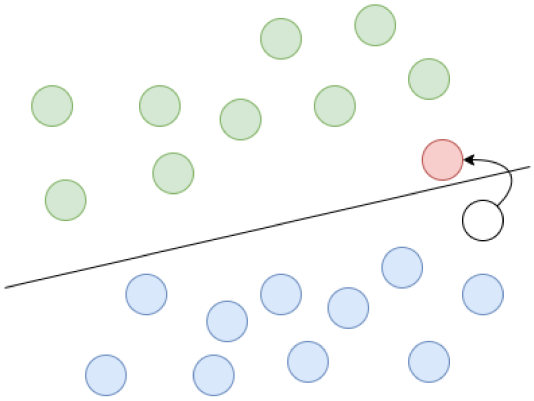

Figure 3.1: Adversarial example

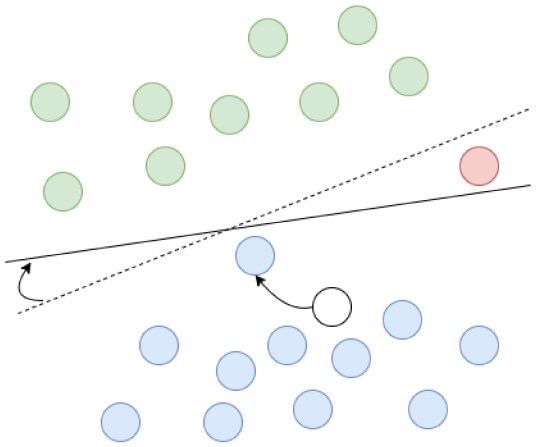

Figure 3.2: Data poisoning

[1] covert method of bypassing normal authentication or encryption in a system

3.2. Attack Settings

There are several different viewpoints that can be used to categorize attack methods (Zhang et al., 2019):

- **model access** - refers to the knowledge of attacked model,
- **target mode** - represents whether the goal of the attack is enforcing incorrect prediction (undirected) or targeting specific results (directed),
- **model type** - considers the model's architecture and optimization process (e.g. feed-forward networks, recurrent networks, convolutional networks, reinforcement learning models).

Model access strongly affects the attacking process. Two polar approaches are distinguished: *black-box* attack where the information is withheld from the attacker and *white-box* with opposite assumptions. The white box represents transparency and the full access to model's information, including architecture, parameters, loss functions, activation function, inputs and outputs. Even though this type of attack is usually very effective, it is not a realistic set-up. Chances are small for the model to be completely open-sourced with every detail known. The black-box scenario is much more sensible in real situations. That is the main reason for setting focus to black-box attacks in this thesis. There is also a middle ground, where some information is known, but still not all of it. This concept is appropriately called *grey-box* scenario.

The levels of adversarial access are another important aspect of the attacker's potentials. They are as follows:

- logic corruption,
- data manipulation,
- data injection,
- transfer learning.

The threat level of listed items is decreasing, from the most to the least dangerous scenario. Logic corruption refers to the attacker's possibility of changing the algorithm and learning method. In this case, absolute control can be achieved.

The second scenario, data manipulation, considers the possibility of altering the data examples. Similarly, data injection is a more restricted version of the former method where manipulation is allowed only in the form of adding new instances. Transfer learning is

considered as the lowest level of interference since it refers to framing a model that had previously been trained on adversarial data. The victim uses the corrupted model in the learning process for other tasks. The impact of adversarial examples fades as correct instances are newly added.

3.3. Adversarial Attacks on Textual Models

Generating adversarial examples is generally motivated by two goals: attack and defence mechanism. The attack's purpose is to evaluate the robustness of targeted models, while the defence takes the process a bit further by exploiting adversarial inputs to make models more robust.

There are certain viewpoints specific to the natural language domain when considering adversarial attacks. One of the most important is the semantic granularity. Since the input for textual models is almost exclusively unstructured text in the form of sentences, the question of granulation arises. The approach depends on the model's embedding[2] level. Embeddings can be based on the following constructs:

- character,
- word,
- sentence.

The attacking strategies are largely influenced by embedding levels.

Altering text without being noticed is a very delicate task. There are several obstacles on the way. Firstly, it is difficult to preserve the text's semantics when employing an attack. Additionally, it must be kept in mind how much has the adversarial example changed compared to the original data instance. This is described with the perturbation measure. The perturbations themselves are intently created noises added to the original input. Usually, the size of the perturbation is calculated as the distance between clean data and its adversarial example (Zhang et al., 2019). The ultimate goal is to make small changes, imperceptible to human, just enough to fool the model.

Adversarial example $\mathbf{x}'$ can be regarded as artificially fabricated data. Given the original data $\mathbf{x}$ and the set of all inputs $\mathbf{X}$, it can formally be defined as:

[2] distributed representation for text in numerical form

$$\mathbf{x}' = \mathbf{x} + \boldsymbol{\delta},\ \ F(\mathbf{x}) = \mathbf{y},\ \ \mathbf{x} \in \mathbf{X}, \tag{3.1}$$

$$F(\mathbf{x}') \neq \mathbf{y},$$

where $\boldsymbol{\delta}$ is the perturbation vector, $\mathbf{y}$ is the target vector and F represents the classifier's prediction function. In this case, $\mathbf{y}$ encodes correct prediction and the model's prediction must differ from it, for the adversarial example to be effective. When the attack seeks to get a specific prediction from the victim model, meaning that it is in directed target mode, conditions are somewhat different. For the mentioned mode, they are as follows:

$$F(\mathbf{x}') = \mathbf{y}' \ \ \wedge \ \ \mathbf{y}' \neq \mathbf{y},$$

where $\mathbf{y}'$ is a specific target that has to be different from the correct one.

All of the mentioned categorizations of adversarial attacks, including the linguistic aspects, are displayed in Figure 3.3.

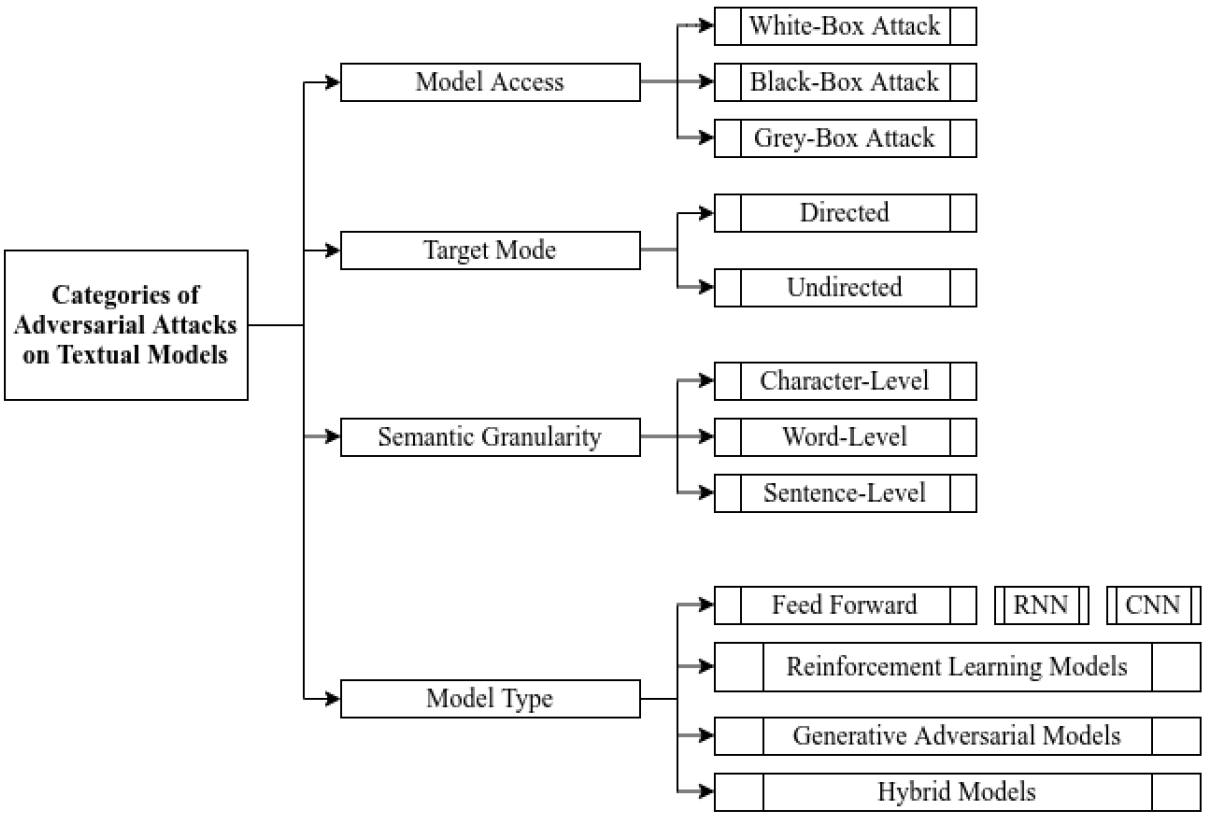

Figure 3.3: Summary of adversarial attack categories

3.4. Vulnerability of Deep Learning Models

Data dependency is an inherent vulnerability of deep learning models. In order to guarantee the trustworthiness of a model, all of the data, used both in training and testing phases, must be clean and uncompromised. That is extremely hard, almost impossible, to achieve.

Models have to train on adversarial examples in order to detect them later on. This presents a problem because of the vast number of different perturbations. Model robustness should increase with the number and variety of adversarial inputs. Nevertheless, that approach is flawed because those examples are designed specifically to shift the decision boundary. If the model sees and trains on too many of them, it can ruin its internal logic and alter the boundary beyond repair.

Training on adversarial data is risky business because a balance has to be achieved. A simplified example of an unsuccessful one is shown in Figure 3.4, where abbreviation AE stands for adversarial example. Model boundary collapses to an almost straight line. Nearly linear boundary won't suffice for correct predictions, and there are many errors as a consequence. The model has become virtually useless. The process of making models more resilient must be done with great caution, in order to avoid counterproductive results.

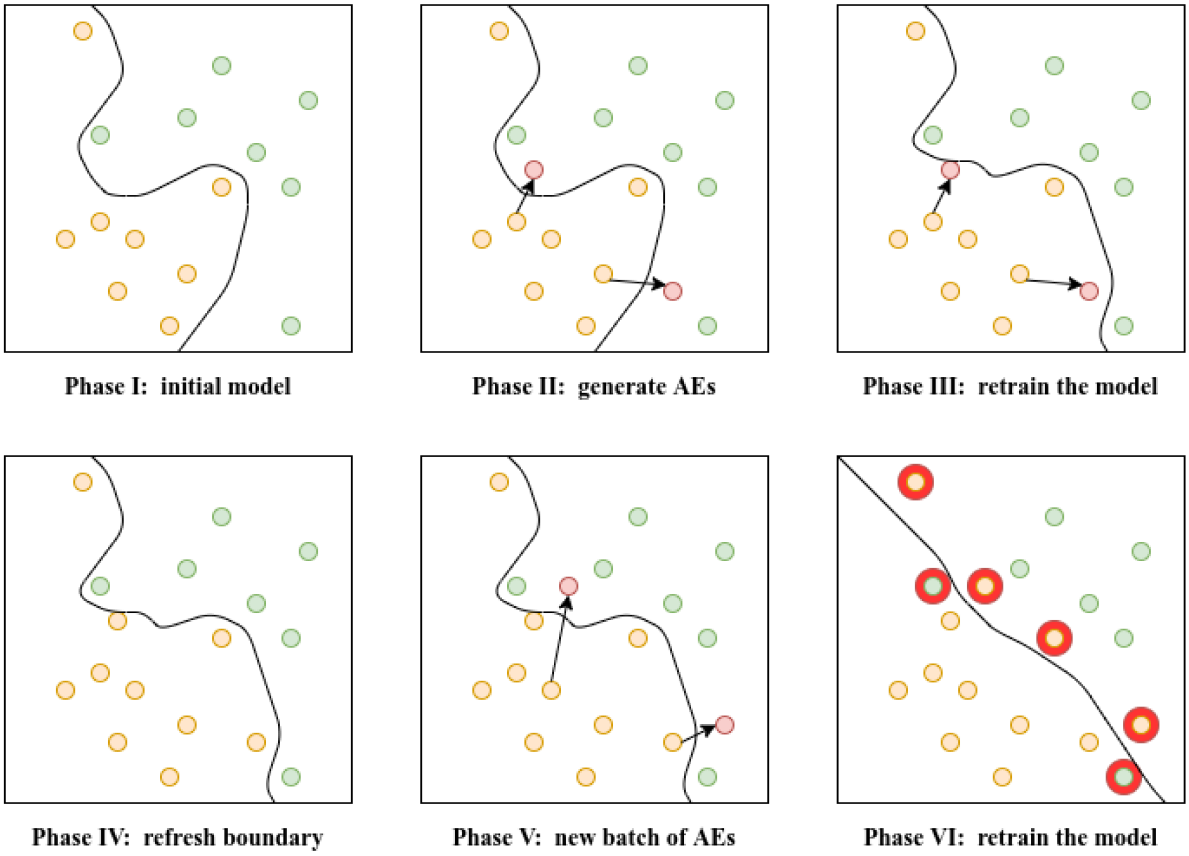

Figure 3.4: Example of an unsuccessful adversarial training. The model becomes oversimplified in the learning process, which is visible in Phase VI.

4. Adversarial Arsenal

Considering the factors of frequency and realistic set-up, the majority of the research has been directed towards *black-box* attacks. The methods are gradually developed to be more covert and semantically appropriate - meaning that a human reader cannot determine whether a text has been perturbed or not.

Attack set-up is shown in Figure 4.1, where it is highlighted that the threat model uses only the probability vector produced as output by the victim model. In the first step, legitimate input is perturbed to become adversarial. Whether the attack was successful or not, gets checked in the second step. This is an example of a directed attack, so the newly predicted label has to match the target label. If that isn't the case, the process can be repeated, but this time already altered input can be plugged back into step I.

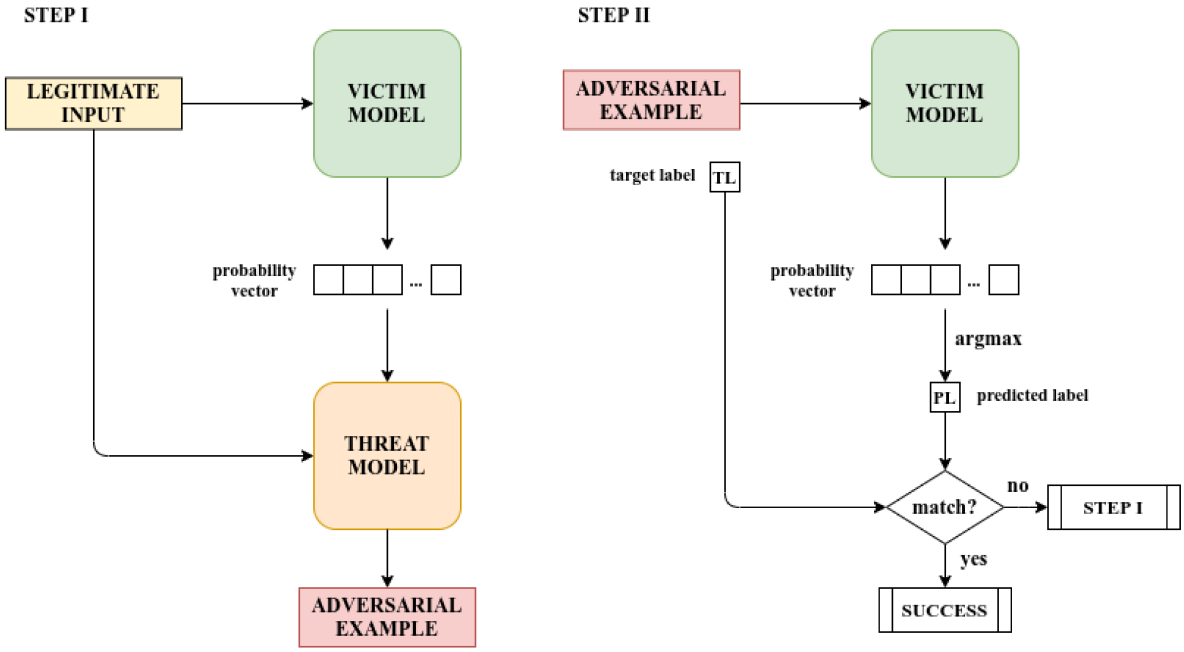

Figure 4.1: Adversarial attack flow. The threat model alters the original input based on the victim model's output score.

4.1. Why Not Gradient?

Studies on adversarial attacks in image domain are much more prolific than ones in texts. Because of that, researchers try to transfer approaches in image domain to text and hope to achieve better results (Wang et al., 2019). This type of attack has shown great promise for images. That has not been a surprise, considering that these attacks exploit the essence of the training procedure. The learning process is pushed in the opposite direction, aiming to achieve the worst possible loss, which makes it so successful. The downfall is the prerequisite of knowing the model's internal logic - loss function, optimization algorithm and model's complete structure (architecture, parameters, activation functions), in order to be able to accurately calculate information necessary for the attack. This essentially puts the attack in a white-box scenario.

The basic idea of any gradient-based attack is to find perturbation as a value proportional to the gradient of model's loss function with respect to its inputs. For instance, in fast gradient sign method (Goodfellow et al., 2014), perturbation is calculated as:

$$\boldsymbol{\delta} = \epsilon sgn(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, \mathbf{y})), \tag{4.1}$$

where J is model's loss function, $\boldsymbol{\theta}$ are its parameters and ϵ is a small positive constant (e.g. $\epsilon = 0.25$). Signum function takes the sign of the calculated gradient. It is defined simply as:

$$sgn = \begin{cases} -1 & \text{if } \mathbf{x} < 0, \\ 0 & \text{if } \mathbf{x} = 0, \\ 1 & \text{if } \mathbf{x} > 0. \end{cases} \tag{4.2}$$

Function is applied for each component of the vector (i.e. element-wise).

The described idea works great for images, but it stumbles upon problems in text. Adding adversarial noise to images produces a new image with slightly altered pixels. On the other hand, coarse-grained space of words in text isn't convenient for small shifts, since it is very unlikely to perfectly match another word when noise is added. This problem can be circumvented by searching for nearest neighbours in the vector space, which can be challenging to do without ruining the semantic properties of text.

4.2. Targeting Words with Inner Bias

Deep learning models search for patterns. In textual data, that can mean the identification of certain words important for prediction. For instance, in sentiment analysis of movie reviews,

the model has to focus on words that contribute to the impression of sentiment. Here is an example of a movie review with negative sentiment:

*This movie had **horrendous** acting, **disappointing** plot, and **terrible** choice of actors.*

And here is an example of a positive movie review:

*The movie was absolutely **stunning**, I unpacked a decade worth of **admiration**.*

The words in bold are the sentiment indicators, without them, it would be impossible to determine the overall sentiment. A good model should have a preference for them when making a prediction. This leaves room for the attacks to exploit the model's *inner bias* for those words.

4.2.1. Embeddings and Vocabulary

Text has to be converted to some form of numerical information, for the model to be able to understand it. Representing discrete types (e.g. words) as dense vectors is the core of deep learning's successes in NLP (Rao and McMahan, 2019). The term *embedding* refers to learning the mapping from a discrete type to a point in the vector space.

A finite set of words used in a certain NLP problem is called *vocabulary*. It is typically collected as all of the words that appeared in a dataset. To reduce the size of the vocabulary, it is possible to filter out the words by their frequency (dismiss low-frequent words) or some other measurement.

Distributed representations have been on a rise for some time. These low-dimensional dense vectors have multiple benefits over the one-hot and count-based encodings (e.g. term frequency - inverse document frequency): they are more computationally efficient, tend to reduce the redundancy of information and can be fine-tuned for task-specific data. As another positive side effect, the phenomenon called the *curse of dimensionality*[1] can be mitigated. Embeddings are the chains that link actual words with their numerical representations. They represent an important part in any NLP model and that is why they are a key figure in some of the following attacks.

A special type of embedding that will be important later on, is the one that maps unseen words or words that are not in the vocabulary. This embedding is usually marked as *unknown* or *UNK* for short. The underlying consequence is that the model will encode every word that

[1] various phenomena that arise when analyzing and organizing data in high-dimensional spaces that do not occur in low-dimensional settings

is not in the vocabulary with the same embedding, typically with all of the components set to zero.

4.2.2. Word Bug

In a black-box scenario, the only available information is the model's prediction vector. Let the classifier be denoted as F. It maps an input text sequence to an output vector. A data example is a pair $(\mathbf{x}, \mathbf{y})$, where $\mathbf{x} = x_1x_2x_3...x_n$ (x_i represents an embedded token). The label $\mathbf{y}$ is encoded as one-hot or multi-hot (if it is a multi-class problem) vector. It is required that the model produces probabilities for each class: $F(\mathbf{x}) = \mathbf{y}'$, where $\mathbf{y}'$ is a probability vector whose elements sum up to one.

A good starting point for an attack is to find important words in text for a given task. Without the guidance of gradients, it is necessary to target words with another scoring system (Gao et al., 2018). It has to be independent of the model's parameters since they are not observable. When those words are identified, the threat model needs to transform target words, keeping in mind the edit distance - how much and in which way will the word be changed. This approach is called *Word Bug* attack.

To employ the attack, two steps have to be considered. Firstly, using the scoring system, most relevant words (generally tokens) are selected. This includes ranking the tokens by their produced score. After that, a transformation algorithm is utilized to alter the words.

Scoring system is a function that uses the model's prediction for determining word importance. It should follow these guidelines (Gao et al., 2018):

1. Scoring function should appropriately reflect the importance of words for a specific task.
2. Knowledge of the model's parameters is considered unavailable.
3. Score calculation should be fairly efficient, because it has to be done for every word in a given text, to make a complete picture.

Gao et al. (2018) propose several scoring functions: temporal score, temporal tail score and their combination.

Temporal Score

How to measure importance of the i-th word of an input sequence $\mathbf{x} = x_1x_2x_3...x_n$? One way is to determine its impact on the classification prediction by comparing model's outputs with and without it in a temporal context.

Let $\boldsymbol{\delta}_i$ denote the perturbation of the i-th token. In the continuous domain (e.g. image), it is possible to alter the original input's i-th element by making a superposition of the corresponding token with $\boldsymbol{\delta}_i$. With x_i' as the newly created token, the resulting change of prediction output $\Delta_i F(\mathbf{x})$ can be approximated using the partial derivative of the i-th feature (Gao et al., 2018):

$$\Delta_i F(\mathbf{x}) \approx (x_i' - x_i)\nabla_{x_i} F(\mathbf{x}) \ . \tag{4.3}$$

Considering a black-box setting, the term $\nabla_{x_i} F(\mathbf{x})$ cannot be calculated because of the lack of needed information about the model. Furthermore, it is difficult to calculate the difference $x_i' - x_i$ since words in text are discrete by their nature. To bypass the posing problem, $\Delta_i F(\mathbf{x})$ can be measured directly by removing the i-th token. Comparing the model's prediction with and without the observed token reflects the word's impact on the output. Since most of the NLP models process input in a sequential manner, the token is observed in temporal context. The temporal score S_t of the i-th element in a sequence is defined as:

$$S_t(x_i) = F(x_1x_2...x_{i-1}x_i) - F(x_1x_2...x_{i-1}) \ . \tag{4.4}$$

The described temporal score of every token in an input x can be calculated inexpensively by one forward pass of the used model.

Temporal Tail Score

The previously described temporal score can be considered myopic since it only takes preceding words of a sequence into account. Following tokens cannot simply be ignored if expected to have full insight in token's significance for the prediction.

An another extreme approach would be to observe only the targeted word's following tokens. New measurement, the temporal tail score S_{tt} should describe whether the word influences the final prediction when coupled with tokens that follow it. The score of the i-th token in a sequence of length n, of which only $n - i + 1$ elements are actively observed, is defined as:

$$S_{tt}(x_i) = F(x_ix_{i+1}...x_{n-1}x_n) - F(x_{i+1}...x_{n-1}x_n) \ . \tag{4.5}$$

Combined Score

Temporal and temporal tail scores are two extreme approaches and they can be viewed as mutually exclusive. This leaves room for their combination. When both directions are taken into account, a clearer picture can be achieved. The combined score S_c can simply be constructed as:

$$S_c(x_i) = S_t(x_i) + \lambda S_{tt}(x_i), \tag{4.6}$$

where λ is hyperparameter used for adjusting the relevance of used measures.

Token Transformer

How to change the most important words selected by some scoring criterion? The main goal is to confuse the model as much as possible during its prediction phase. Small changes are preferable, some transformation that can have controlled edit distance. It would be sufficient to alter the targeted token just enough for it to be unrecognizable, which means that it does not exist in the model's vocabulary.

One way of achieving the described goal is by using a simple transformation to deliberately create misspelt words. Since vocabulary knowledge of an NLP system is crammed into a finite set of possible words, it is a relatively easy task to convert a token into *unknown*.

Modifying the words with one of the following methods keep the Levenshtein distance at low value:

1. **substitution** of a character in a word with a random letter,
2. **insertion** of a random letter,
3. **deletion** of a random character in the targeted token,
4. **swap** two adjacent characters in the word.

First three methods result with the edit distance equal to one while swapping makes it two. The generated adversarial inputs should appear visually and morphologically similar to human observers (Gao et al., 2018).

A somewhat different transformation function would be replacement of a character with its *homoglyph* - another character that has an identical or very similar shape, but it is not essentially the same character (i.e. it possesses a different encoding).

A few transformation examples are shown in Table 4.1. Homoglyph column contains instances with visually distinctive letters, for demonstration purposes. In a real attack, homoglyphs can be so similar that is nearly impossible to tell them apart.

Table 4.1: Word transformations

Original	*Substitute*	*Insert*	*Delete*	*Swap*	*Homoglyph*
keyboard	kezboard	keuyboard	keboard	kebyoard	keẙboard
astonishing	astonizhing	astoniishing	astonshing	astonihsing	astonişhing
horrible	horriboe	horriblse	horrible	horribl	horriblé

4.2.3. Word Drop

By studying the model's behaviour with input as a temporal sequence, it is difficult to keep the whole rest of the text when eliminating a single word. Removal is necessary for comparison which leads to a conclusion about the token's importance. *Word Bug* variations described in the subsection 4.2.2 can only partially view the input sequence. The most informative variation is combined score, but that approach is also flawed since it looks at the context before and after the targeted word one at a time. This obstacle can be bypassed if the sequences' lengths are fixed. They also must be equal in size. A novel approach of estimating word importance, called *Word Drop* is based on that idea without significant losses in computational performance.

Sequence Packing

When training a deep learning model, input data usually come in batches. To ensure that model can accept every instance in a batch, they must all be the same length. This can typically be achieved by padding the sequences to the length of the longest one (e.g. fill them up with zeros until they reach the wanted length). The downside of that is the unnecessary calculation of pad tokens which will result in zero either way.

Circumvention can be done with sequence packing. Let the following be the input examples (i.e. batch consisting of three sequences):

Long	input	sequence	with	many	words.
Short	sentence.				
Medium	is	the	best.		

With <*pad*> representing the pad token, vocabulary that only contains words appearing in the provided batch is a set:

{<*pad*>, *best*, *input*, *is*, *long*, *many*, *medium*, *sentence*, *sequence*, *short*, *the*, *with*, *words*}.

For the process of sequence padding to work, sequences need to sort in descending order by their lengths. After conversion of tokens into indices of corresponding vocabulary elements and sorting by length, the batch may look like this:

$$\begin{matrix}[4 & 2 & 8 & 11 & 5 & 12] \\ [9 & 7 & 0 & 0 & 0 & 0] \\ [6 & 3 & 10 & 1 & 0 & 0]\end{matrix} \xrightarrow{descending\ \ sort} \begin{matrix}[4 & 2 & 8 & 11 & 5 & 12] \\ [6 & 3 & 10 & 1 & 0 & 0] \\ [9 & 7 & 0 & 0 & 0 & 0]\end{matrix}$$

Next step is packing - token by token is taken from each sequence, until it runs out of non-pad tokens. This procedure with randomly initialized five-dimensional embeddings is illustrated in two phases. The first phase considers the input as follows:

[	2.1511	0.1019	−0.3452	1.2028	0.7754	] long
[	−0.4189	0.6132	0.2892	−0.3298	−1.5918	] input
[	0.0982	2.9812	1.1096	1.4319	−1.8320	] sequence
[	0.4260	−1.3618	0.5322	0.9163	0.9982	] with
[	0.1999	0.3618	−0.1872	−0.5123	0.2221	] many
[	−0.7612	−1.0899	0.1991	0.2214	−0.0210	] words
[	3.1209	0.1129	0.6426	−0.9102	0.5713	] medium
[	0.1120	0.3214	−1.2106	0.5317	−0.3109	] is
[	0.8823	−1.5561	−3.2210	1.2020	−1.8744	] the
[	−0.2291	0.2718	0.4672	−0.9910	0.0781	] best
[	0.4261	−1.3618	0.5322	0.9163	0.9982	] <*pad*>
[	0.4261	−1.3618	0.5322	0.9163	0.9982	] <*pad*>
[	0.8111	0.2199	−1.0439	−1.3451	−0.7189	] short
[	2.4012	−0.9901	−2.3310	0.5512	0.0916	] sentence
[	0.4261	−1.3618	0.5322	0.9163	0.9982	] <*pad*>
[	0.4261	−1.3618	0.5322	0.9163	0.9982	] <*pad*>
[	0.4261	−1.3618	0.5322	0.9163	0.9982	] <*pad*>
[	0.4261	−1.3618	0.5322	0.9163	0.9982	] <*pad*>

The shape of the input tensor is currently $3 \times 6 \times 5$ ($batch\ size \times maximum\ length \times embedding\ size$). Pad token removal is left to be done. After token reorganization, the result tensor should look like:

[	2.1511	0.1019	−0.3452	1.2028	0.7754	] long
[	3.1209	0.1129	0.6426	−0.9102	0.5713	] medium
[	0.8111	0.2199	−1.0439	−1.3451	−0.7189	] short
[	−0.4189	0.6132	0.2892	−0.3298	−1.5918	] input
[	0.1120	0.3214	−1.2106	0.5317	−0.3109	] is
[	2.4012	−0.9901	−2.3310	0.5512	0.0916	] sentence
[	0.0982	2.9812	1.1096	1.4319	−1.8320	] sequence
[	0.8823	−1.5561	−3.2210	1.2020	−1.8744	] the
[	0.4260	−1.3618	0.5322	0.9163	0.9982	] with
[	−0.2291	0.2718	0.4672	−0.9910	0.0781	] best
[	0.1999	0.3618	−0.1872	−0.5123	0.2221	] many
[	−0.7612	−1.0899	0.1991	0.2214	−0.0210	] words

The input tensor shape is now 12×5, which is considerable smaller and more computationally efficient. Sequence packing is a methodical prerequisite for estimating word importance. The illustration of the described process is displayed in Figure 4.2.

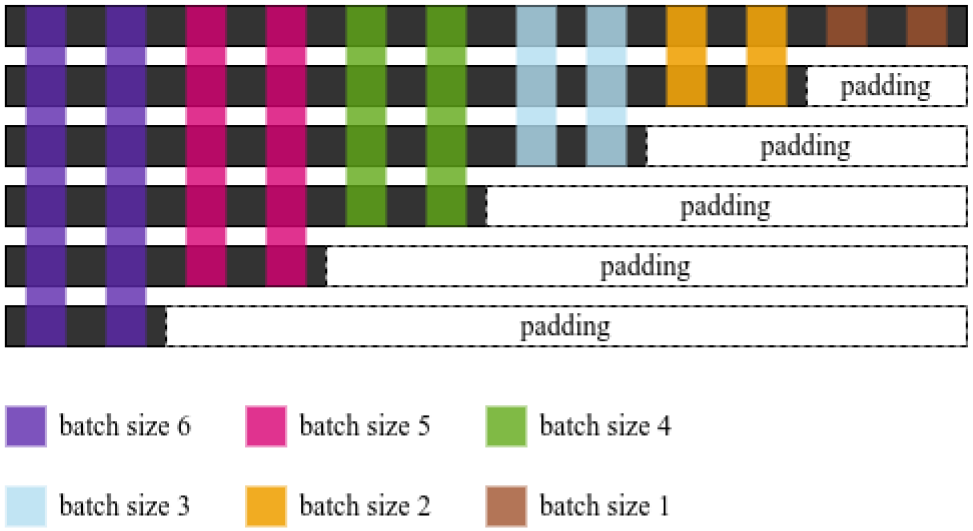

Figure 4.2: Sequence packing

Word Drop in Action

The omission of words can be done simply by removing tokens on targeted positions. The beauty of the constructed packed sequence set-up is the possibility of efficient computing in batches. The algorithm can calculate the influence of a token with a particular index for multiple texts. *Word Drop* score S_{wd} of the i-th token x_i is defined as follows:

$$S_{wd}(x_i) = F(x_1 x_2 ... x_{i-1} x_i x_{i+1} ... x_n) - F(x_1 x_2 ... x_{i-1} x_{i+1} ... x_n) \ . \tag{4.7}$$

When structure for input examples is prepared, the process can be executed similarly to any *Word Bug* attack variation. For a series of texts formed as packed sequences, one by one word is dropped, but the rest of the sequence is preserved. Thanks to the input structure, the computation can be executed in parallel. The intensity of the attack can be regulated with the attack power. That is essentially the number of total words that can be changed,

i.e. transformed with a chosen transformation function. The described process is shown in discrete steps in Algorithm 1.

The method *word_drop_scores* presents the batch version of described score calculation. Relevance for each word in every sequence is processed at once. The packing process is executed in *pack_sequence* method, while *sort_by_sequence_length* sorts inputs in decreasing order.

Algorithm 1 Word Drop Attack

Arguments:

input sequences	$\mathbf{X} = \mathbf{x_1 x_2 ... x_{n-1} x_n}$,
model	classification algorithm,
vocab	model's vocabulary,
attack power	$P :=$ maximum number of changed tokens,
iterator	object for batch navigation,
transform	function for adversarial token transformation

Returns:

Set of newly created adversarial examples.

adversarial_examples $= \varnothing$
for batch **in** iterator.*generate_batches*($\mathbf{X}$) **do**
  token_indices $=$ vocab.*convert_tokens_to_indices*(batch)
  token_indices, lengths $=$ *sort_by_sequence_length*(token_indices)
  packed_batch $=$ *pack_sequences*(token_indices, lengths)
  y_preds $=$ model.*predict*(packed_batch) // prediction vectors
  scores $=$ *word_drop_scores*(model, batch, y_preds)
  indices $=$ *argsort*(scores) // sorted argument indices of every sequence
  top_indices $=$ indices$[:, 0 : P]$ // take only top P indices for every sequence
  for $i = 0$ .. *row_count*(top_indices) **do**
    adv_ex $=$ *transform*(top_indices$[i]$) // single adversarial example
    adversarial_examples $+=$ adv_ex
  end for
end for
return adversarial_examples

4.3. Lexical Substitution

Lexical substitution is an NLP task of identifying replacements for a word in the context of a clause. For instance, given the text:

"*After the **match**, athletes go through recovery phase.*",

word ***game*** can be a satisfying substitute for the word ***match***.

This task is closely related to word sense disambiguation (WSD), in a way that both aim to determine the meaning of words. While WSD deals with the automated assignment of sense from a fixed set of option inventory, lexical substitution does not impose any limitations on which word substitution to use. Ideally, the provided substitutes should be synonyms of targeted words. Moreover, the predicted synonym must semantically fit into a sentence context.

Word substitution will be the key component in the upcoming attacks. The main flaw of previously described methods *Word Bug* and *Word Drop* is the lack of semantic integrity. With adversarial word transformation, the model can no longer interpret the altered token. Those methods are relatively easy to discover by setting up an automated process of word integrity - checking for consistent misspells or unfamiliar character encodings.

Language Model

Generally, a language model represents a statistical probability distribution over sequences of words. It provides context to distinguish between parts of sequences that can sound similar. Nevertheless, language models can be utilized in lexical substitution tasks.

One such model is BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers). It is called a mask language model because it is possible to mask one or more words in a sentence and have the model predict those masked words given the corresponding context. These kinds of models are usually pre-trained on an extremely large text corpus.

The simplest way of using BERT for substitution is taking the words with the highest probability within a given context. An example of BERT prediction in language model context is shown in Figure 4.3. The [*CLS*] token is used to mark the sequence beginning and [*MASK*] denotes the masked word - model will produce the most likely options to fill in the spot. If the best answer corresponds to the actual word (i.e. the word that is masked), the substitute can be the next best prediction.

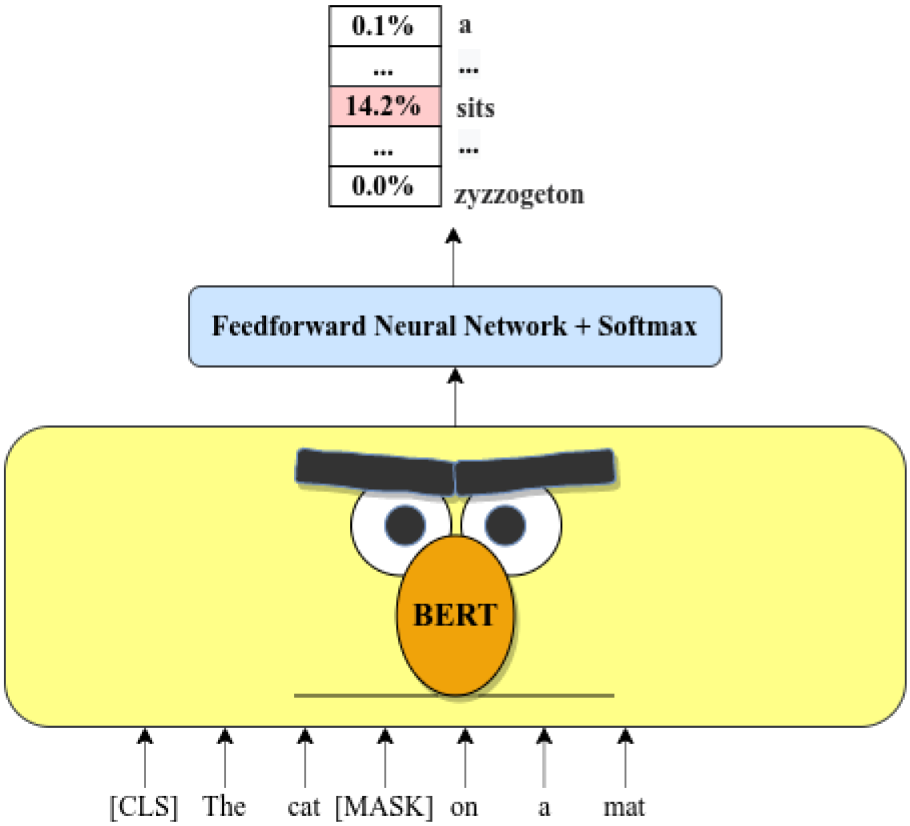

Figure 4.3: BERT as a masked language model. A feedforward neural network along with the softmax output layer is stacked on top of pre-trained embeddings. The model predicts appropriate substitutes for masked tokens considering the surrounding context.

For the example sentence: "*The cat ___ on a mat.*", the model produces word **sits** as the best fit. In this way, substitution words can now be trivially extracted. The only prerequisite is to convert the targeted word to mask token.

A more advanced approach considers applying embedding dropout to partially mask the target word. This implies an additional, dropout layer after the embeddings. Individual embeddings are either dropped out from the calculation with a probability $1 - p$, or kept with probability p, where p is the probability of dropping a single embedding. It is argued that complete masking can often result in substitutes that are semantically different, although they fit into context (Zhou et al., 2019). The embedding dropout is proposed to mitigate the problem.

Counter-Fitted Vectors

Language models perform relatively poorly when it comes to finding word synonyms. They return words that are often found in a similar context as the target word. This behaviour

is based on the *distributional hypothesis* which assumes that semantically similar or related words appear in similar contexts (Harris, 1954). However, learning word embeddings from co-occurrence information in text corpora can lead to coalescence of two similar but different notions: semantic similarity and conceptual association (Hill et al., 2015).

Another way of lexical substitution relies heavily on vector space. Encoded words should be closer together if they have similar meanings or are connected somehow. Since training word embeddings is connected to training language models, it is required to find a way to correctly link synonyms. Mrksic et al. (2016) propose a way to ensure linguistic constraints by utilizing a counter-fitting procedure. It seeks to bring the word vectors of known synonymous word pair closer together in the vector space.

The core of the idea is to use an optimization method (e.g. stochastic gradient descent) to squeeze distance between synonyms close to zero. As as starting point, an indexed set of word vectors $V = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n\}$ is collected. Newly constructed word vectors, into which semantic relations will be injected, are denoted as $V' = \{\mathbf{v'}_1, \mathbf{v'}_2, ..., \mathbf{v'}_n\}$. A set of constraints S is consisted of pairs of indices such that the corresponding words from vocabulary are synonyms. Formally, the procedure SA, called **synonym-attract**, can be defined as (Mrksic et al., 2016):

$$SA(V') = \sum_{(u,w)\in S} d(v'_u, v'_w), \tag{4.8}$$

where d is an arbitrary distance measure (e.g. Euclidean distance).

A few examples are shown in Table 4.2, taken from Mrksic et al. (2016), that show the difference between pre-trained *GloVe*[2] vectors and counter-fitted ones.

[2]Global Vectors for Word Representations: https://nlp.stanford.edu/projects/glove/

Table 4.2: Regular vs. counter-fitted vectors

	east	**expensive**	**British**
Regular	west	pricey	American
	north	cheaper	Australian
	south	costly	Britain
	southeast	overpriced	European
	northeast	inexpensive	England
Counter-Fitted	eastward	costly	Brits
	eastern	pricy	London
	easterly	overpriced	BBC
	-	pricey	UK
	-	afford	Britain

4.4. Reinforced Genetic Attack

To avoid limitations of gradient-based attacks and employ constructed mechanism for lexical substitution, it is necessary to design an optimization procedure that meets all of the constraints. It has already been demonstrated that genetic algorithm copes well with this type of problem (Alzantot et al., 2018). However, the algorithm can be refined and enhanced to achieve even better results.

4.4.1. Threat Model

The threat model is designed to work in a black-box scenario. It can only query the victim model to obtain the probability vector of its output. Any information about the model's parameters, architecture, loss and activation functions are off the limits.

There are a few different options, regarding the attack goal. Adversarial examples can be directed - used to force the model to predict a specific class, or they can be undirected where the threat model only aims for the victim model to misclassify the example.

4.4.2. Chromosome Design

One of the crucial components of the attack is chromosome representation. The algorithm should be able to operate with compact structures. Given that the threat model works with textual data, the input sequences are already encoded with vectors assigned to each token. For easier manipulation with chromosomes, the actual structure contains only indices of representational vectors that are stored in the embedding matrix E. The size of a chromosome depends on the number of its tokens. For instance, if a text is broken into n tokens, the corresponding chromosome would be an array consisting of n indices. In the process of crossover and mutation, the word substitutions are always conducted as one-to-one mapping. Thus, the chromosome's size remains fixed even after applying genetic operators.

To put things into context, let the example of a vocabulary with mapped indices be defined as:

$$\{\ \textit{a}\colon 0,\ \textit{captivating}\colon 1,\ \textit{character}\colon 2,\ \textit{compelling}\colon 3,\ \textit{engrossing}\colon 4,$$
$$\textit{is}\colon 5,\ \textit{main}\colon 6,\ \textit{movie}\colon 7,\ \textit{of}\colon 8,\ \textit{profile}\colon 9,\ \textit{psychological}\colon 10,$$
$$\textit{the}\colon 11,\ \textit{very}\colon 12,\ \textit{with}\colon 13\ \}.$$

As an example of a movie review with positive sentiment, the following sentence represents the initial textual sequence and its phenotype: "*The movie is very engrossing with a compelling psychological profile of the main character.*".
Once the mapping to corresponding indices is completed, the chromosome (i.e. genotype) should be represented as $[11\ 7\ 5\ 12\ 4\ 13\ 0\ 3\ 10\ 9\ 8\ 11\ 6\ 2]$. The indices may reoccur if the same word is repeated throughout the text. Furthermore, the token vector is easily accessible in a single operation via the embedding matrix. In particular, the embedding of the i-th token can be extracted in constant time simply by retrieving $E[i]$. If the fifth token "*engrossing*" is substituted with a synonym, the corresponding word index merely needs to be switched with the substitute's index. For instance, if the synonym in question is "*captivating*" with index equal to 1, the chromosome will undergo a minor transformation:

$$[11\ 7\ 5\ 12\ \underline{\mathbf{4}}\ 13\ 0\ 3\ 10\ 9\ 8\ 11\ 6\ 2] \rightarrow [11\ 7\ 5\ 12\ \underline{\mathbf{1}}\ 13\ 0\ 3\ 10\ 9\ 8\ 11\ 6\ 2].$$

By using the vocabulary's inverse mapping, the chromosome can easily be reverted to its textual representation.

4.4.3. Components Connection

A lot of constraints have been imposed on the procedure. As few as possible words should be modified in the attack, but they have to maintain semantic similarity with the original ones, as well as the syntactic coherence. To achieve this task, a population-based genetic algorithm is employed (Alzantot et al., 2018).

There are several steps involved in the attack. Firstly, the lexical substitution model has to be chosen. As variations of previously described methods, the options are as follows:

- **Euclidean model** - employs counter-fitted vectors to calculate nearest neighbours using Euclidean distance,
- **WordNet model** - uses WordNet[3] database to retrieve synonyms for certain words,
- **Lin thesaurus model** - utilizes a powerful thesaurus that is a part of the NLTK[4] (Natural Language Toolkit) library,
- **BERT** - used as a masked language model that serves as a proxy in finding lexical substitutions.

Listed lexical models provide the possibility of preserving the semantic soundness of targeted text. They serve as the first component used repeatedly by the genetic algorithm during the optimization process. Furthermore, these models should be able to filter words that are supposed to be skipped during the attack. This often relates to stop words and punctuation tokens, which can easily ruin text integrity if substituted.

To ensure that a certain substitute word fits well, the substitutability measure is introduced. It calculates the word's affiliation towards other tokens in text, based on the context. This can be done by computing distances from each token and finding the mean value. Formally, the substitutability value V_{subst} of i-th word in a text can be defined as follows:

$$V_{subst}(v_i) = \frac{\sum_{j=0, j\neq i}^{N-1} d(v_i, v_j)}{N-1}, \tag{4.9}$$

where N is the number of tokens in the text, v_i corresponds to the embedding vector of the i-th word and d is arbitrarily chosen distance measure.

For an additional enhancement of the attacking procedure, the part-of-speech tagging can be utilized. Lexical databases such as WordNet often benefit from grammatical categories when delivering an answer to a query. That is a simple way of integrating word sense in synonym search.

[3]WordNet website: https://wordnet.princeton.edu/
[4]NLTK platform: https://www.nltk.org/

As a second step, genetic perturbation must be carefully designed. This method will eventually be used in mutation, as well as in the creation of the initial population. The *perturb* procedure uses lexical substitution model to retrieve top n candidates, where n is defined as the magnitude of search space. With the increase of this parameter, it is more likely for the semantic correctness to be disrupted, but the search space is larger so it's easier to find an adversarial model that would fool the *victim model*.

Finally, the implemented steps are used in the full-blown genetic algorithm to slightly perturb the given text in order to achieve false prediction. The population initialization is done by perturbing a random word with the lexical model. If the wanted population size is N, the process is repeated N times. There are several options for crossover:

- uniform crossover,
- k-point crossover,
- segmented crossover.

The segmented crossover presented with the best behaviour, when considering how believable were the adversarial examples - meaning how hard it is to detect that text was meddled with. In the used variation of segmented crossover, the procedure traverses through elements of parents. It points to one parent at a time. If the focus is on the first parent in the i-th step, the i-th element in the child chromosome will be from the first parent and similarly for the second parent. In each step transition, there is a small probability that the focus will switch to the other parent. The process is repeated until all of the elements are exhausted.

With the described components put together, the only thing left to do is used one of the word-targeted attacks to reinforce the procedure, such as *Word Drop* described in 4.2.3. This secondary attack evaluates the importance of each word in a given text, which can be converted to weighted probability. Newly constructed probability vector is now used in selection processes by the genetic algorithm. For instance, in the population initialization and mutation methods, the vector is utilized to achieve *roulette wheel*[5] selection on the token level.

The whole attack procedure is summarized in Algorithm 2. Lexical substitution model retrieve top results with *get_neighbours* function. The used criterion is specific to each model, as previously described. It is crucial for computational benefits to convert the population of adversarial examples (in numerical format) to a compact batch. Prediction results can now be effectively calculated for all instances. Selection, crossover and mutation operations are all done in bulk - for the whole population. The genetic algorithm ensures that elitism is

[5] fitness proportionate selection

applied by always adding the best example to the next generation. Regarding the termination conditions, the halt function simply compares predicted and targeted classes. If the attack is directed and the predictions match, the procedure is successful and it returns the best adversarial example. On the other hand, in undirected attacks, mismatch of predictions indicate the favourable outcome.

Algorithm 2 Reinforced Genetic Attack

Arguments:

original text	$\mathbf{x}_{orig}$,
weights	$\mathbf{w}$:= reinforcement weights from the secondary attack,
directed	boolean flag that is true if the attack is directed,
target_class	target class can be directed or undirected,
generation count	G := maximum number of generations,
lexical substitution model	M_{LS},
victim model	M_V,
candidate count	n_{cand} := number of top results taken into consideration by M_{LS},
substitute count	n_{subs} := number of best substitutes extracted from candidates,
pop_size	wanted population size.
halt_fn	function that determines if the attack is successful.

Returns:

Best adversarial example based on the success in fooling the *victim model*.

neighbours, lengths = M_{LS}.*get_neighbours*($\mathbf{x}_{orig}, n_{cand}, n_{subs}$)
pop = *generate_pop*($\mathbf{x}_{orig}$, neighbours, weights, target, pop_size)
for $i = 1 .. G$ **do**
  batch = *prepare_batch*(pop)
  pop_preds = M_V.*predict_proba*($batch$)
  pop_scores = pop_preds[:, target]
  if ¬directed **then**
    pop_scores = **1** − pop_scores
  end if

  top_attack = *argmax*(pop_scores)
  select_probs = *softmax*(pop_scores)
  prediction = *argmax*(pop_preds[top_attack, :])
  best = pop[top_attack]
  if *halt_fn*(prediction, target) **then**
    return best
  end if
  parent_pairs = *select_parents*(pop, select_probs, pop_size − 1)
  children = *crossover*(parent_pairs)
  children = *perturb*(children) // equivalent to mutation
  pop = *concat*(best, children) // apply elitism
end for
return best

5. Experiments and Results Analysis

A battery of experiments was conducted to properly evaluate the efficiency and characteristics of certain attacks. A wide spectrum of models was adjusted to different datasets, achieving greater diversity in experiments. Extensive analysis that followed afterwards, aimed to dive in different aspects of the observed attacks, as well as their pros and cons. Because of the abundance of data examples throughout the used datasets, the corresponding split proportions are as follows:

- **train set** - 35%,
- **validation set** - 15%,
- **test set** - 50%.

Large test sets help in providing more reliable evaluations.

Most of the functionality was implemented in *Python* programming language. For the construction of various models, the powerful *PyTorch* machine learning framework was used along with *Torchtext* library, designed for NLP experiments.

5.1. Datasets

Since experiments are conducted on two types of tasks, text classification and natural language inference, there are two corresponding sets of datasets. For attacks in regular classification, the following datasets are used:

- Internet Movie Database (IMDB) reviews,
- Stanford Sentiment Treebank (SST),
- Yahoo questions,
- AG news corpus.

Textual entailment experiments are conducted on these datasets:

- Stanford Natural Language Inference (SNLI),
- Multi-Genre Natural Language Inference (MultiNLI).

IMDB

IMDB movie reviews dataset (Maas et al., 2011) consists of $50,000$ examples. Each instance can belong to one of two categories: *positive* sentiment or *negative* sentiment. The reviews are informal and mostly polar regarding their sentiment. For better performances, a few preprocessing steps were executed before running any experiments. Texts were collected in HTML (Hypertext Markup Language) format, so the corresponding tags and variables were stripped along with the special characters that bear no significance for sentiment analysis. Inclusion of other standard steps, like stop words removal and stemming or lemmatization is based on the goals of certain experiments. Thus, they are carried through only in some of the microstudies.

SST

The Stanford Sentiment Treebank includes fine-grained sentiment labels for $215,154$ phrases in the parse trees of $11,855$ sentences (Socher et al., 2013). Initially, there had been 25 different sentiment values, with 1 being the most negative and 25 the most positive. Dataset labels were reduced to three separate classes: *positive*, *negative* and *neutral* sentiment, similar to IMDB reviews dataset with the addition of neutral sentiment.

Yahoo Questions

Large corpus of questions used both for question answering and topic classification tasks is gathered from the Yahoo search engine. Regarding the possible topics of user's queries, there are ten distinct classes:

1. Society & Culture,
2. Science & Mathematics,
3. Health,
4. Education & Reference,

5. Computers & Internet,
6. Sports,
7. Business & Finance,
8. Entertainment & Music,
9. Family & Relationships,
10. Politics & Government.

The dataset is quite large, so for the experiment purposes, a total of 120.000 examples had been randomly taken from it.

AG News

AG news dataset is a collection of more than one million news articles. Texts are collected from more than $2,000$ sources. Each article is assigned one of the four following topics:

1. World,
2. Sports,
3. Business,
4. Science & Technology.

SNLI

The SNLI corpus is is a collection of $570,000$ human-written English sentence pairs manually labelled for balanced classification with the annotations: *entailment*, *contradiction*, and *neutral*, supporting the task of natural language inference (Bowman et al., 2015), described in section 2.2. Each pair contains its premise and hypothesis, between which relation is questioned.

MultiNLI

Similarly to the SNLI dataset, the MultiNLI corpus is a crowd-sourced collection of $433,000$ sentence pairs labeled with textual entailment information. The corpus is modelled on the

SNLI corpus but differs in that it covers a range of genres of spoken and written text, and supports a distinctive cross-genre generalization evaluation (Williams et al., 2018).

5.2. Models

As mentioned at the beginning of the chapter, deep learning algorithms were implemented in *PyTorch* ecosystem. All of them extend an abstract base class, which defines several methods that every model has to implement:

- *forward* - represents a forward pass resulting with logits which are non-normalized raw predictions generated by the model,
- *predict_proba* - outputs a probability estimation vector, typically calls the *forward* method with inputs and then applies *sigmoid* function for binary classification or *softmax* for multi-class data,
- *predict* - retrieves predicted classes for the provided input, it essentially the *argmax* value of *predict_proba* function's output.

For standard classification, most of the models are built on top of a recurrent neural network (RNN). Diversity is achieved by changing the RNN cell type with the following three options:

1. plain RNN,
2. long short-term memory (LSTM),
3. gated recurrent unit (GRU).

Sequential models can process inputs unidirectionally or from both ends. This is left as an option since the victim model's bidirectionality can be significant for the attack behaviour. Finally, every model also has two working modes: normal and packed. Sequence packing, described in subsection 4.2.3, is a useful tool because it improves models' performances and it ensures the feasibility of the *Word Drop* attack.

Finally, a transformer model used for text classification experiments is a BERT classifier, whose base was introduced in section 4.3 where BERT is used as a foundation for a lexical substitution model. Implementation is relied on the *Hugging Face*[1] library. The model itself

[1]Hugging Face website: https://huggingface.co/

is based on *attention* mechanism that learns relations between words. A transformer includes two separate entities - an encoder that reads the textual input and a decoder that estimates a prediction for the given problem.

The models for textual entailment require a somewhat different approach. It is necessary to provide components for both the premise and hypothesis part (depicted in section 2.2). In addition to RNN-based layers, there are several fully connected linear layers to deepen the model. Two separate parts are processed in parallel and concatenated at the end to form a single tensor output. As in standard classification models, it is possible to choose one cell type among plain RNN, LSTM and GRU. Model's directionality and activation functions are adjustable as well.

All of the implemented deep learning models are parameterized so that they can have arbitrary architectures. It is possible to adjust the number of layers and dimensions of each layer, excluding the input and output layer which are dependent on data. Furthermore, there is an option of applying dropout throughout the models' layers. For the sake of comparability, every model uses pre-trained *GloVe* embeddings.

5.3. Attacking Efficiency

The attacks are executed under the assumption that the victim model is at least better than a random prediction. Otherwise, the model would be useless and the attack could only increase the model's performance by changing its decisions which contradicts the attack's purpose. If the actual labels of original data are omitted, then the process of forging adversarial examples is led strictly by the model's predictions. For instance, if the targeted model predicts class y for the provided input, the adversarial example y' should be such that $y' \neq y$, regardless of the input's true label (i.e. gold label). Some experiments are also conducted with known gold labels, whose results are displayed later on in this section.

Tables with results are divided by dataset pairs. There are three pairs in total, where each dataset is coupled with one that is the most similar to it. The outcome of coupling is:

- (IMDB, SST) - sentiment analysis,
- (Yahoo questions, AG news) - topic categorization,
- (SNLI, MultiNLI) - textual entailment datasets.

The RNN-based models listed within tables are implemented with sequence packing, two recurrent layers and dropout probability set to 0.5. Hidden layer size is 256 and embedding

dimension is 100. Models are trained for 20 epochs with the possibility of early stopping after 3 epochs with a consecutive decrease in validation performance. The batch size is set to 64. *Adam* optimizer (Kingma and Ba, 2014) is used in combination with cross-entropy loss for the training process.

Results denoted with "*regular*" present models' performances without applying any attack. The first baseline (i.e. baseline I) is derived by taking random words and changing them to *unknown* token, modelled by *Word Bug* and *Word Drop* attacks. On the other hand, the *greedy swap* attack (i.e. baseline II) is designed to take only the most important words in a text and swapping them with a first available synonym (statically mapped in *WordNet* database). These models are primarily developed to achieve a more nuanced comparison between other attacks and them. All of the other adversarial methods are described in detail in chapter 4, along with ***RGA*** denoting the *Reinforced Genetic Attack* which is defined in section 4.4.

Results marked with † in superscript are statistically better than other listed attacks on the same dataset with the same models. Statistical tests are conducted with significance level $\alpha = 0.01$, as $\mathbf{10}$-fold $\mathbf{t}$-tests. Numbers displayed in bold represent the best scores, i.e. the most effective attacks in their respective categories. Maximum allowed perturbation is $\mathbf{10\%}$ (equation 3.1). First set of attacks, including *Word Bug* variations and *Word Drop* always perturb the maximum permissible amount of text. Greedy swap and RGA may result in a maximum perturbation or lower.

Firstly, the accuracy measures on IMDB and SST datasets with varying models and attacks are shown in Table 5.1. *RGA* is a clear winner, certified by statistical tests. Attacking efficiency seems to be correlated with the quality of a model, with attacks having a greater relative impact on better models rather than on poorer ones. This can be explained by the higher ratio of incorrect labels provided by the poor models. Nevertheless, attacks seem to be effective in either case. Lower accuracy of certain models corresponds to more effective attacks.

Table 5.2 contains results for RNN-based models on Yahoo questions and AG news topic categorization. All of the attacks outperform their corresponding baselines, but *Word Drop* and *RGA* tend to be the most effective ones.

Judging by the results of *temporal* and *tail* attacks, it can be hypothesized that in many cases the context before is more important than the context after for understanding the text. Moreover, *temporal* score consistently outperforms *tail* score on IMDB, SST, Yahoo and AG datasets, which is supported by statistical t-test with significance level $\alpha = 0.01$.

Table 5.1: Accuracy of victim models on IMDB & SST datasets

Dataset	**IMDB**			**SST**		
Model	bi-RNN	bi-LSTM	bi-GRU	bi-RNN	bi-LSTM	bi-GRU
Regular	0.8966	0.9153	0.9056	0.6220	0.6381	0.6197
Random (Baseline I)	0.8648	0.8821	0.8617	0.6114	0.6211	0.6093
Word Bug - Tail	0.6747	0.6754	0.6727	0.5229	0.5213	0.5319
Word Bug - Temporal	0.6132	0.6121	0.6097	0.4892	0.4831	0.4956
Word Bug - Combined	0.4914	0.5178	0.5097	0.4274	0.4301	0.4194
Word Drop	0.2772	0.3188	0.2957	0.1674	0.1761	**0.1719**
Greedy Swap (Baseline II)	0.7921	0.7984	0.7899	0.5372	0.5415	0.5399
RGA	**0.2127**†	**0.2099**†	**0.2879**	**0.1612**	**0.1455**†	0.1821

Table 5.2: Accuracy of victim models on Yahoo Questions & AG News

Dataset	**Yahoo Questions**			**AG News**		
Model	bi-RNN	bi-LSTM	bi-GRU	bi-RNN	bi-LSTM	bi-GRU
Regular	0.6106	0.6238	0.6211	0.8550	0.8691	0.8458
Random (Baseline I)	0.5913	0.5942	0.5891	0.8152	0.8275	0.8177
Word Bug - Tail	0.5220	0.5421	0.5439	0.7057	0.7344	0.7391
Word Bug - Temporal	0.5007	0.5121	0.5094	0.6742	0.6811	0.6796
Word Bug - Combined	0.4821	0.4312	0.4111	0.6132	0.6227	0.6308
Word Drop	0.4314	0.4101	0.4121	0.4703	**0.4501**†	**0.4819**
Greedy Swap (Baseline II)	0.5775	0.5813	0.5644	0.7710	0.7927	0.7699
RGA	**0.1971**†	**0.1877**†	**0.1902**†	**0.4613**	0.4911	0.4864

A drop in performance under *RGA* is even more expressed in NLI tasks. It is the undisputed winner for every model type on both SNLI and MultiNLI datasets.

Table 5.3: Accuracy of victim models on SNLI & MultiNLI datasets

Dataset	**SNLI**			**MultiNLI**		
Model	bi-RNN	bi-LSTM	bi-GRU	bi-RNN	bi-LSTM	bi-GRU
Regular	0.7795	0.7814	0.7803	0.6370	0.6419	0.6283
Random (Baseline I)	0.7511	0.7529	0.7689	0.6298	0.6324	0.6210
Word Bug - Tail	0.6157	0.6098	0.6209	0.5114	0.5017	0.5045
Word Bug - Temporal	0.6014	0.6122	0.6137	0.4988	0.5007	0.4834
Word Bug - Combined	0.5422	0.5326	0.5307	0.4327	0.4201	0.4339
Word Drop	0.4188	0.2928	0.3219	0.2978	0.3779	0.3135
Greedy Swap (Baseline II)	0.6987	0.6981	0.7270	0.6107	0.6094	0.6113
RGA	$\mathbf{0.2201}^{\dagger}$	$\mathbf{0.1135}^{\dagger}$	$\mathbf{0.1577}^{\dagger}$	$\mathbf{0.2177}^{\dagger}$	$\mathbf{0.0954}^{\dagger}$	$\mathbf{0.1225}^{\dagger}$

Due to longer training time, experiments with BERT classifier are conducted only under *Word Drop* attack and *RGA*, along with both baselines. Despite the fact that BERT achieves better results on most datasets, it tends to be more susceptible to attacks.

Table 5.4: Accuracy of BERT classifier under attacks

Dataset	IMDB	SST	Yahoo Questions	AG News
Regular	0.9078	0.6412	0.6297	0.8812
Random (Baseline I)	0.8801	0.6214	0.5914	0.8271
Word Drop	0.1847	$\mathbf{0.0751}^{\dagger}$	0.2101	0.1458
Greedy Swap (Baseline II)	0.8014	0.5783	0.4922	0.7541
RGA	$\mathbf{0.1424}^{\dagger}$	0.1233	$\mathbf{0.1215}^{\dagger}$	**0.1403**

The real power of attacks lies in their success with low levels of perturbation. It is easy to fool models if adversarial examples are extremely altered, but that approach is easily de-

tectable. How do attacks behave with different limitations? Some insight can be extracted from Figure 5.1. It displays dependency of perturbation and attack success. Data had been accumulated from all of the six datasets used in experiments, and the corresponding results were averaged. Each of the three shown attacks: *RGA*, *Word Drop* and *Word Bug*, presents with a surge in success at the early stages of perturbation level. This can be explained by the minimum required alteration of original examples for the adversarial ones to take effect on the model's output. After a sudden rise, *RGA* quickly reaches a high plateau where the curve stagnates at nearly 100% success rate. *Word Drop* and *Word Bug* possess different behaviour, reaching peak performance at 60% perturbation level, with some decrease in success towards the completely perturbed example. At the endpoint, the text is represented only with unknown tokens, which leaves the model to practically take a random guess.

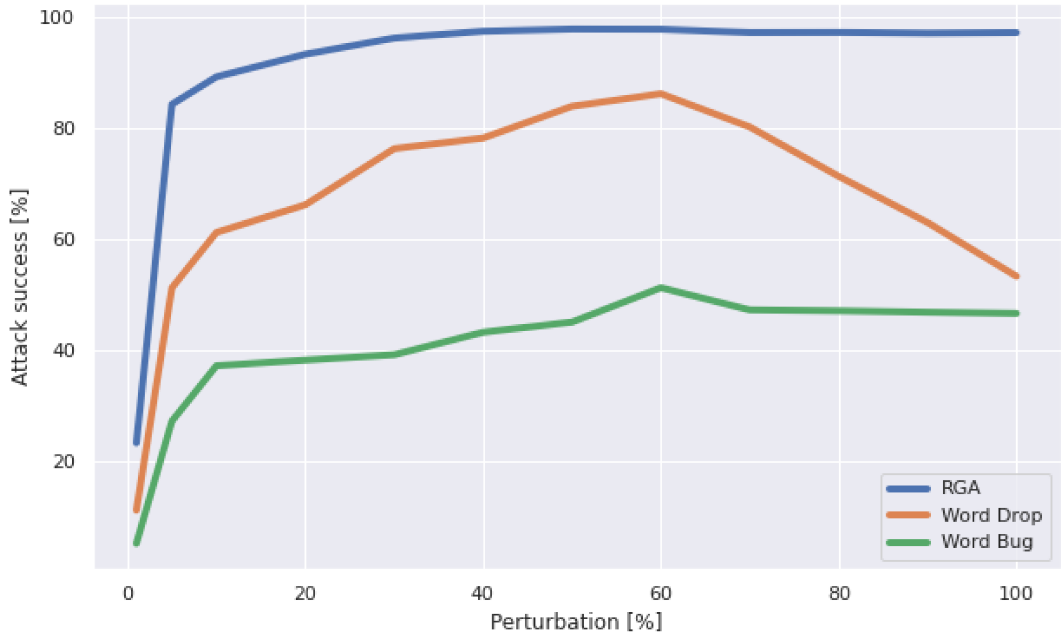

Figure 5.1: Perturbation vs. attack success. The figure displays averaged results on multiple datasets. Attack success represents the ratio of adversarial examples that fool the model to total number of targeted instances.

5.4. Reinforced Genetic Attack Analysis

RGA has shown the most promising results in the analysis of attacking efficiency in the previous section. To truly uncover the strengths and weaknesses of this attack, it is necessary to dive in the specifics. The general success of the attack is quite solid and the next important metric is semantic integrity. Several other measures can be significant when generating

adversarial examples. Three criteria have been picked as a priority:

1. **perturbation level** - the measure of the difference between the original and the adversarial example,
2. **target loss** - how far-off is the attack from switching the model's prediction,
3. **human impression loss** - how likely it is for a human to detect anomalies in adversarial text.

The first two metrics are straightforward to express numerically. Counting the different words at the matching positions serves as perturbation level, whereas target loss has two scenarios. Firstly, in targeted attacks the target loss is measured as $1 - P(Y = y')$, where y' is the targeted class. Secondly, in undirected attacks, where the goal is for the model to predict any class but the current one, the loss is simply $P(Y = y)$, where y is the original prediction. The last metric, human impression loss, requires a more advanced technique. The tool of choice is a BERT-based masked language model described in section 4.3. The main idea is for the model to predict a loss for a given input, where the lower loss means that the words are more likely to appear together in the provided context. Metric is supposed to mimic the human judgement because it implicitly provides an evaluation for structural and semantic soundness.

In Figure 5.2, the chromosomes of targeted *RGA* on IMDB dataset are displayed. Population size is set to 50 and the maximum number of generations is 20. The green circles represent the adversarial examples that managed to change the model's prediction and the dark red crosses are the ones that failed. Pareto front is shown as a red line. Two groups are distinguishable, located in the left and right corner. The left group with fewer examples presents the medium perturbed examples with high semantic integrity and human impression. On the other hand, the right group is dispersed on different levels of perturbation, whereas their human impression loss is quite high. The gap can be explained by the difficulty of achieving fine-grained perturbation with adjustable integrity. In most cases the result is binary, either the text is semantically sound or it contains discerning holes in integrity.

Figure 5.3 represents an attempt of showing the dependency of target loss and perturbation level. The set-up is the same as in Figure 5.2 with the corresponding 50 chromosomes, but with different scores. The successful adversarial examples are quite dispersed when it comes to the amount of alteration. If the human impression is not a concern, there is a diverse spectrum of dominant solutions that can be picked.

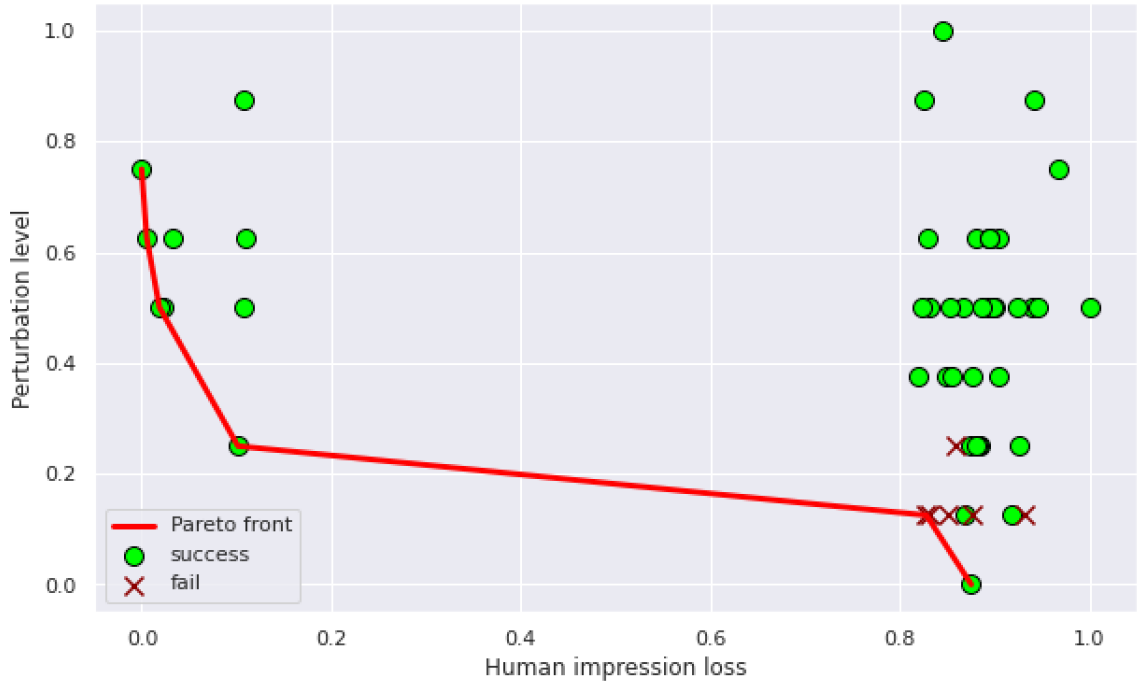

Figure 5.2: Human impression loss vs. perturbation level. Results are shown for *RGA* on IMDB dataset with Pareto front indicated in red color.

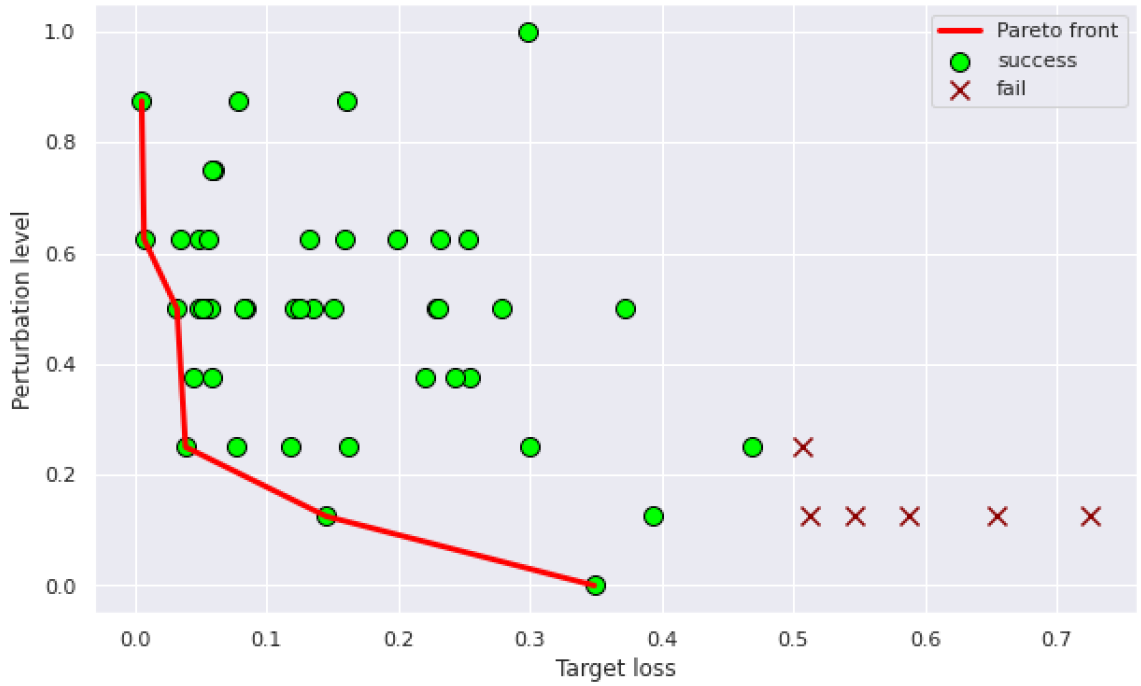

Figure 5.3: Target loss vs. perturbation level. Pareto front is shown as a red line. Target loss represents the objective function that needs to be minimized. As target loss decreases, the victim model's prediction is deeper in the subspace of targeted class.

An essential part of the attack is its lexical substitution model. The boxplot in Figure 5.4 displays average target losses of the threat model using four different models: BERT language model (blue), Lin Thesaurus (orange), Euclidean (green) and WordNet model (red). Results are gathered by running *RGA* 20 times per example and taking the average loss, with $10,000$ instances in total from all of the six datasets described in 5.1.

When it comes to median loss, WordNet model is the winner, while the Lin Thesaurus model shows the lowest dispersion in target loss. Euclidean model also provides solid performance, especially if its computational efficiency is taken into consideration. Unlike Euclidean, the WordNet requires part-of-speech tagging which takes a toll on the algorithm's time complexity.

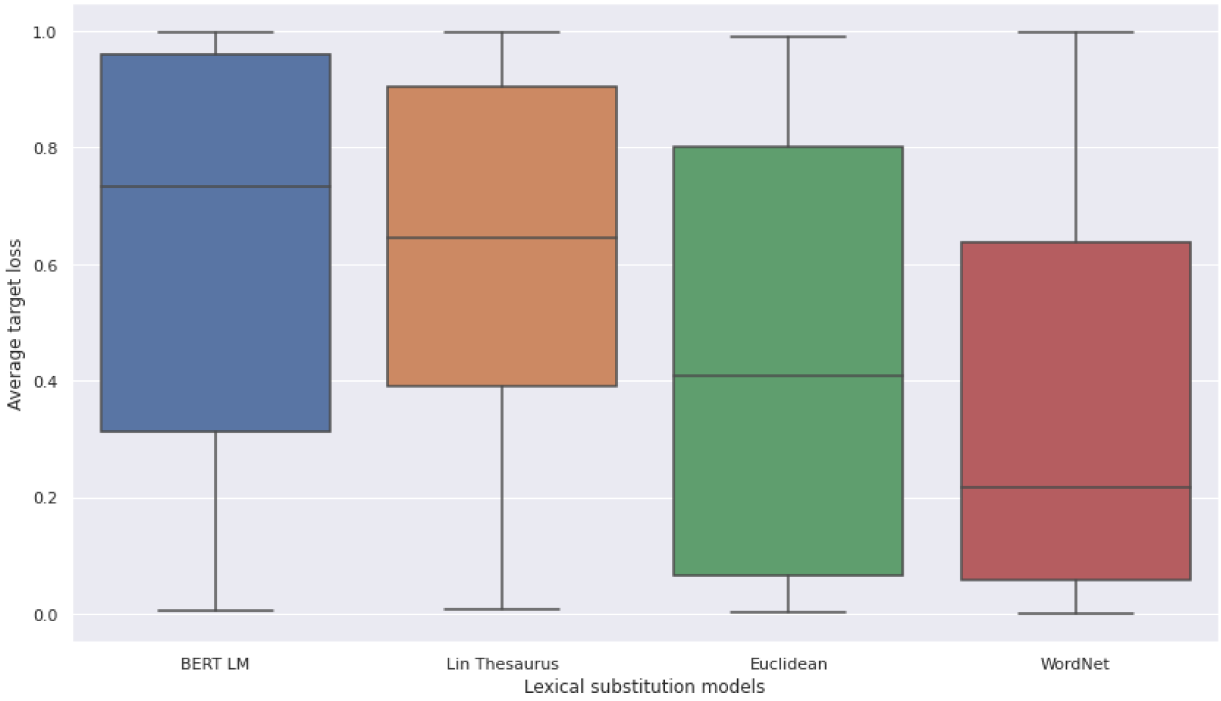

Figure 5.4: Boxplot depicting *RGA*'s target loss with different lexical substitution models.

Furthermore, to see how the models behave in action, concrete cases are provided. In a series of examples, it is shown how does the attack cope with directed attacks with specific target classes and the undirected ones.

The first example is shown in Table 5.5, which is a non-targeted attack. The replacement words are indicated in red colour, and it can be seen that they correspond to sentiment carriers.

Table 5.5: Non-targeted adversarial example (IMDB)

Original & adversarial texts	Prediction
Amazing movie. Script writing is quite good. Beautiful scenery and great acting.	*positive* 0.95
Astonishing movie. Script writing is quite good. Nice-looking scenery and wonderful acting.	negative 0.84

The next set of examples shows the directed version of the attack. Table 5.6 displays the original text from the Yahoo questions dataset and several examples of adversarial text with different target classes. Most of the cases provide quality solutions, but sometimes there are some obstacles in the way. For instance, the first adversarial example in the first block has made change **accounts** → **accountants**, in order to switch the model prediction from *Entertainment & Music* to *Business & Finance* category. The substitution word is not an actual synonym, but it is morphologically similar. Other examples successfully executed the perturbation, preserving the original semantics and tricking the model as well.

Table 5.6: Targeted adversarial examples (Yahoo questions)

Original & adversarial texts	Prediction
If you guys want to ask more questions why don't you just create multiple accounts?	*Entertainment & Music* 0.77
If you guys want to ask more questions why don't you just get multiple accountants?	Business & Finance 0.82
If you guys want to pose more subjects why don't you just generate multiple accounts?	Education & Reference 0.68
I am trying to get along with my fiance but I don't know how to settle our difference.	*Family & Relationships* 0.85
I am trying to get along with my fiance but I don't understand how to solve our differential.	Science & Mathematics 0.76
I am trying to get along with my fiance but I don't know how to address our variance.	Computer & Internet 0.59

Lastly, there are some examples from the SNLI dataset in Table 5.7. Most of the hypotheses are short and the attack has to carefully pick one or two words to appropriately substitute.

The relation between the premise and its hypothesis must also be taken into consideration.

Table 5.7: Adversarial examples in NLI domain (SNLI)

Premise	Original & adversarial hypotheses	Prediction
Two dogs in a grassy field.	*Two dogs are outside.*	*entailment* 0.98
	Two canines are out.	contradiction 0.63
A man is giving a presentation.	*The man is not talking.*	*contradiction* 0.94
	The human is not speaking.	entailment 0.71
A blond woman in a white shirt demonstrates her talents to a crowd.	*The woman is a street performer.*	*entailment* 0.92
	The woman is a road entertainer.	contradiction 0.79
	The lady is a street performer.	neutral 0.82

5.5. Attack Transferability

The transferability of an adversarial attack represents its ability to be effective on models that haven't been used in the process of adversarial examples generation (Szegedy et al., 2013). Table 5.8 shows transferred accuracy of four different models: bi-RNN, bi-LSTM, bi-GRU and the BERT classifier. Each block contains results for the models which the adversarial examples are generated on - generator models (denoted with $\bullet$ in superscript), and their corresponding transferred performances (for other models). For each attack and base model, the next best accuracy is displayed in bold.

The most surprising results are indicated in green colour and they appear in cases where the BERT classifier is the generator model. Unexpected fact is that the models LSTM and GRU achieve better results when using transferred adversarial examples.

Table 5.8: Accuracy of transferred attacks (AG news)

Attack \ *Model*	bi-RNN	bi-LSTM	bi-GRU	BERT
Regular	0.8550	0.8691	0.8458	0.8812
Word Bug - Combined	0.6132•	**0.6352**	0.6374	0.6471
Word Drop	0.4703•	**0.5122**	0.5231	0.5549
Greedy Swap	0.7710•	**0.8245**	0.8297	0.8566
RGA	0.4613•	0.5334	**0.5013**	0.5578
Word Bug - Combined	0.6521	0.6227•	**0.6309**	0.7119
Word Drop	0.4933	0.4501•	**0.4680**	0.5123
Greedy Swap	0.8306	0.7927•	**0.7982**	0.8231
RGA	0.9078	0.4911•	**0.6297**	0.8812
Word Bug - Combined	0.6892	**0.6436**	0.6308•	0.6998
Word Drop	0.5216	**0.4907**	0.4819•	0.5564
Greedy Swap	0.8122	**0.7725**	0.7699•	0.8308
RGA	0.5062	**0.4913**	0.4864•	0.5411
Word Bug - Combined	**0.6320**	0.6455	0.6495	0.4921•
Word Drop	0.5211	**0.4726**	0.5147	0.1458•
Greedy Swap	0.8144	0.8221	**0.8019**	0.7541•
RGA	0.4702	**0.3741**	0.4201	0.1403•

5.6. From Adversarial Examples to Data Poisoning

In a scenario where restrictions are more relaxed and there is a possibility to influence training set, the generated adversarial examples can be used to poison the learning data pool. The idea is to provide incorrect labels in the learning process, in order to simplify the decision function. The model becomes too generic and its predictions are virtually useless.

Although the adversarial examples aren't crafted to undermine the learning process, they still show some potential in poisoning attacks. Experiments are conducted on IMDB dataset with an LSTM model (pre-trained LSTM) that has already been trained on regular data and also an LSTM model (LSTM zero) that had no prior training. Original training set size of the pre-trained LSTM is the same as the poisoned data size. Thus, the half of the training

set consists of regular data and the other half are adversarial examples (i.e. poisoned data). Accuracy on test data is calculated in every set-up. Results are shown in Table 5.9 for *RGA* and *Word Drop* poisoning, as well as for normal training. If the model is trained only on poisoned data, it serves no further purpose. In the case of an already trained model, its performance is significantly dropped.

Table 5.9: Accuracy on poisoned data (IMDB)

Model	pre-trained LSTM	LSTM zero
Normal training	0.9153	0.5079
Word Drop poisoning	0.6329	0.0139
RGA poisoning	0.1424	0.1233

6. Defence Mechanisms

How to properly protect a model from attacks, especially from adversarial examples? There isn't a definitive answer and there are very few methods that achieve any success in defence. The main problem is the sheer variety of adversarial data. Plenty of angles can be used to approach the attack. The consequence is that there is no silver bullet. The model has to be fortified with multiple defence methods to stand any chance against attacks.

There are some simple approaches to mitigate the danger from simple attacks. Models can be protected with:

- text sanitization,
- checking if there is a concerning number of unknown tokens,
- checking for frequent misspellings.

When it comes to more sophisticated attacking methods, a more elaborate approach to security measures is required.

6.1. Adversarial Training

One of the straightforward ideas is adversarial training as a defence mechanism. Training the model on adversarial examples, but with correct labels, is the essence of it. There are some inherent problems with it, as described previously in section 3.4. The main point is that too much of adversarial training can oversimplify the model and make it useless.

The adversarial training has to be combined with training on regular data. Models can achieve better generalization with small batches of adversarial examples to train on. To explore the possibilities, an LSTM model was used on IMDB data. Initially, the model had been trained on $25,000$ regular examples. It was evaluated on an adversarial test set, consisting of combined data generated by *Word Drop* attack and *RGA*. Furthermore, five different instances had been trained on other adversarial examples with different sizes. The first model

instance was evaluated immediately after its standard training, and the other four used, respectively, 10%, 20%, 50% and 100% of the other adversarial set with $25,000$ instances in total. The results are shown in Table 6.1. Initially, there is a bump in accuracy at 10% of examples, but later on, a decrease can be identified when using 20%, 50% or 100% of the generated adversarial data. The best result is indicated in bold.

Table 6.1: Influence of adversarial training on LSTM's accuracy (IMDB)

Adversarial pool size	0	$2,500$	$5,000$	$12,500$	$25,000$
Accuracy	0.2099	$\mathbf{0.3891}$	0.3511	0.3112	0.1203

6.2. Attract-Repel Embeddings

Adversarial training turned out to be somewhat limiting when it comes to defence, especially the amount of adversarial data that a certain model can withstand to train on. In a search for more robust defences, certain types of word embeddings transpired to work well. The idea is based on the counter-fitted embeddings that are used in some of the attacks described previously.

In order to make a model resistant to synonym replacement, it is necessary to set certain linguistical constraints on word vectors. The attract-repel algorithm uses synonymy and antonymy constraints taken from lexical resources to tune word vector spaces (Mrkšić et al., 2017). This leads to similar vectors (close in space) of synonym pairs, which effectively means that the model's prediction won't differ much if a certain word is replaced with its synonym. Furthermore, the antonym pairs are further apart in space, ideally orthogonal. The method works to some extent, surpassing the capabilities of plain adversarial training. Using the same experimental set-up as in the previous section, evaluating the accuracy of an LSTM model on the adversarial test set, some improvements are shown. The accuracy has risen from 0.2099 (model without defences) to $\mathbf{0.4521}$ when the attract-repel algorithm is applied.

7. Conclusion

Adversarial attacks are not yet fully explored, especially in the NLP domain. When looking back at today's trends in machine learning and deep learning, it is very important to invest attention in models' security.

Attacks on text classification or natural language inference models can be achieved with simple ideas, such as *Word Bug* variations and *Word Drop* attack. They have limitations regarding semantic integrity, but they are quite successful in tearing down models' performances. On the other hand, these attacks are easily detectable, by both humans and computers. More sophisticated methods are required to fool the model and to generate semantically and structurally sound adversarial models. Those advanced attacks are based on lexical substitution, searching for synonyms of targeted words in vector space.

Reinforced Genetic Attack has shown good properties. It can generate hard adversarial examples with preserved text integrity, which is supported by the conducted experiments. Furthermore, the attack is versatile, functioning well on different kinds of text. The threat model still has some issues, which are influenced by the imperfection of word embeddings. Nevertheless, it copes well with many challenges.

Attack is the best form of defence, so many of the developed methods can be also used as defence mechanisms. Adversarial training is a functioning, but limited technique. The proposed attract-repel defence is a more general approach and it produces decent results, but there is still plenty of room for improvement.

In conclusion, there will be more and more machine learning models and more security flaws that can be exploited. By studying the attacks, insights can be retrieved which can be used in building quality defences. Security should always be respected and held in the back of mind. Unquestionable responsibility comes with creating broadly used systems and applications, especially ones powered by machine learning.

BIBLIOGRAPHY

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. pages 2890–2896, 2018. doi: 10.18653/v1/D18-1316. URL https://www.aclweb.org/anthology/D18-1316/.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. *CoRR*, abs/1508.05326, 2015. URL https://www.aclweb.org/anthology/D15-1075/.

Jacob Eisenstein. *Introduction to Natural Language Processing*. The MIT Press, 1st edition, 2019.

J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi. Black-box generation of adversarial text sequences to evade deep learning classifiers. pages 50–56, 2018. URL https://arxiv.org/abs/1801.04354/.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. 2014. URL https://arxiv.org/abs/1412.6572/.

Zellig S. Harris. Distributional structure. *WORD*, 10(2-3):146–162, 1954.

Felix Hill, Roi Reichart, and Anna Korhonen. SimLex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695, December 2015. URL https://www.aclweb.org/anthology/J15-4004/.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014. URL https://arxiv.org/abs/1412.6980/.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. pages 142–150, June 2011. URL https://www.aclweb.org/anthology/P11-1015/.

Nikola Mrksic, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gasic, Lina Maria Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. Counter-

fitting word vectors to linguistic constraints. *CoRR*, abs/1603.00892, 2016. URL https://arxiv.org/abs/1603.00892.

Nikola Mrkšić, Ivan Vulić, Diarmuid Ó Séaghdha, Ira Leviant, Roi Reichart, Milica Gašić, Anna Korhonen, and Steve Young. Semantic specialisation of distributional word vector spaces using monolingual and cross-lingual constraints. 2017. URL https://arxiv.org/abs/1706.00374/.

Delip Rao and Brian McMahan. *Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning*. O'Reilly Media, Inc., 1st edition, 2019.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. pages 1631–1642, October 2013. URL https://www.aclweb.org/anthology/D13-1170/.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. 2013. URL https://arxiv.org/abs/1312.6199/.

Wenqi Wang, Benxiao Tang, Run Wang, Lina Wang, and Aoshuang Ye. A survey on adversarial attacks and defenses in text. *CoRR*, abs/1902.07285, 2019. URL https://arxiv.org/abs/1902.07285/.

Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. pages 1112–1122, 2018. URL https://aclweb.org/anthology/N18-1101/.

Wei Emma Zhang, Quan Z. Sheng, Ahoud Alhazmi, and Chenliang Li. Adversarial attacks on deep learning models in natural language processing: A survey. 2019. URL https://arxiv.org/abs/1901.06796/.

Wangchunshu Zhou, Tao Ge, Ke Xu, Furu Wei, and Ming Zhou. BERT-based lexical substitution. pages 3368–3373, July 2019. doi: 10.18653/v1/P19-1328. URL https://www.aclweb.org/anthology/P19-1328/.

Appendix A
Dataset Sources

IMDB movie reviews	https://ai.stanford.edu/ amaas/data/sentiment/
SST	https://nlp.stanford.edu/sentiment/index.html/
Yahoo questions	https://sourceforge.net/projects/yahoodataset/
AG news	https://www.kaggle.com/amananandrai/ag-news-classification-dataset/
SNLI	https://nlp.stanford.edu/projects/snli/
MultiNLI	https://cims.nyu.edu/ sbowman/multinli/

Appendix B
Image Sources and Tools

2.1 Basic NLP pipeline	source: https://spacy.io/usage/processing-pipelinest/
3.1 Adversarial example	tool: *draw.io* (https://app.diagrams.net/)
3.2 Data poisoning	tool: *draw.io* (https://app.diagrams.net/)
3.3 Summary of adversarial attack categories	tool: *draw.io* (https://app.diagrams.net/)
3.4 Adversarial training gone wrong	tool: *draw.io* (https://app.diagrams.net/)
4.1 Adversarial attack flow	tool: *draw.io* (https://app.diagrams.net/)
4.2 Sequence packing	tool: *draw.io* (https://app.diagrams.net/)
5.1 Perturbation vs. attack success	tool: *Seaborn* (Python library)
5.2 Human impression loss vs. perturbation level	tool: *Seaborn* (Python library)
5.3 Target loss vs. perturbation level	tool: *Seaborn* (Python library)
5.4 Boxplot: RGA's target loss with different lexical models	tool: *Seaborn* (Python library)

Adversarial Attacks in Natural Language Processing

Abstract

Motivation for this assignment, crafting adversarial examples for textual machine learning models, is presented in the introductory chapter. Natural language processing and adversarial attack concepts, important for understanding the experiments, are briefly described. Main topic of this assignment is development of various attacks in the natural language domain. Emphasis is put on effective attacks and methods that can preserve structural and semantic integrity of text. Novel approaches have been designed. They rely on evaluating importance of certain words in text, along with the heavy use of lexical substitution models and custom-crafted word embeddings. A series of experiments had been conducted in order to evaluate and compare different adversarial attacks. The thesis is concluded with short overview of possible defence mechanisms and insights gathered from numerous results.

Keywords: adversarial, attack, text, natural language, classification, entailment, machine learning, deep learning

Suparnički napadi u obradi prirodnog jezika

Sažetak

Motivacija za problem izgradnje suparničkih primjera, opisana je u uvodnom poglavlju. Obrada prirodnog jezika i koncepti suparničkih napada ukratko su opisani. Glavni cilj rada je razvoj raznolikih napada u području prirodnog jezika. Naglasak je stavljen na učinkovite napade i na metode koje mogu očuvati strukturalnu i semantičku cjelovitost teksta. Razvijeni su novi pristupi napada. Oslanjaju se na procjenu značajnosti pojedinih riječi u tekstu, kao i na intenzivno korištenje modela za leksičku zamjenu te ručno rađenih reprezentacija riječi. Provedeni su brojni eksperimenti kako bi se procijenile i usporedile izvedbe pojedinih suparničkih napada. Rad je zaključen kratkih pregledom mogućih obrambenih mehanizama i uvidima koji su stečeni na temelju dobivenih rezultata.

Ključne riječi: suparnički, napad, tekst, prirodni jezik, klasifikacija, zaključivanje, strojno učenje, duboko učenje