Towards Deployable Robust Text Classifiers

by

Lei Xu

B.E., Tsinghua University (2017)
S.M., Massachusetts Institute of Technology (2020)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2022

Author .....
Department of Electrical Engineering and Computer Science
August 15, 2022

Certified by .....
Kalyan Veeramachaneni
Principal Research Scientist
Laboratory for Information and Decision Systems
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Towards Deployable Robust Text Classifiers

by

Lei Xu

Submitted to the Department of Electrical Engineering and Computer Science
on August 15, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

Text classification has been studied for decades as a fundamental task in natural language processing. Deploying classifiers enables more efficient information processing in various applications. However, the deployment of classifiers also presents long-standing and quite challenging problems. As they are increasingly used in critical applications, the complicated cyberspace and the increasing impact of classifiers on decision-making contributes to higher expectations for classifier robustness, fairness, accuracy on scarce training data, and so forth.

In this dissertation, we aim to develop more deployable and robust text classifiers, with a main focus on improving classifier robustness against adversarial attacks by developing both attack and defense approaches. Adversarial attacks are a security concern for text classifiers in which a malicious user can take a sentence and perturb it slightly to manipulate the classifier's output. To design better attack methods, we focus on improving adversarial sentence quality and reducing computation. For the first of these, existing methods prioritize misclassification and ignore the criteria of sentence similarity and fluency. We thus propose synthesizing these three criteria into a combined critique score and outline a rewrite and rollback framework for optimizing this score and achieving state-of-the-art attack success rates while improving similarity and fluency. For the second concern, existing methods typically use combinatorial search to find adversarial examples that alter multiple words, so they are inefficient and require plenty of queries to the classifier. We overcome this problem by proposing a single-word adversarial perturbation attack. This attack only needs to replace a single word in the original sentence with a high-adversarial-capacity word, which significantly improves its efficiency and makes its attack success rate similar to existing methods.

The most common approach for defending against attacks is training classifiers using adversarial examples as data augmentation, even though such methods are limited by the inefficiency of attack methods. We explore data augmentation with our efficient single-word perturbation attack and show that it can also improve the robustness of the classifier on other existing attack methods. We also design *in situ* data augmentation to counteract adversarial perturbations in the classifier input.

We use the gradient norm to identify keywords for classification and a pre-trained language model to replace them. Our *in situ* augmentation can effectively improve the robustness and does not require tuning the classifier.

Finally, we explore the vulnerability of a very recent text classification architecture – prompt-based classifiers, and find them to be vulnerable to attacks as well. We also develop a library called Fibber to facilitate adversarial robustness research.

Thesis Supervisor: Kalyan Veeramachaneni
Title: Principal Research Scientist, Laboratory for Information and Decision Systems

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the contemporary proliferation of artificial intelligence, text classifiers are increasingly being deployed across plenty of systems. Classifiers are so widespread, in fact, that in recent years, most people have interacted with them almost every day without knowing. For example, when we use voice to control our smartphones, the intentions of our utterances are recognized by a classifier [Ren and Xue, 2020]. Companies use classifiers to automatically filter resumes and speed up hiring [Ali et al., 2022, Chen et al., 2018]. Academic digital libraries use classifiers to automatically categorize papers [Li et al., 2019b]. Social media such as Twitter or Reddit uses classifiers not only to identify user sentiment [Colbaugh and Glass, 2010], but also to detect misinformation such as toxic speech [Risch and Krestel, 2020] and fake news [Shu et al., 2017] and to filter annoying spam [Guzella and Caminhas, 2009]. These classifiers being developed and deployed thus have an extensive impact on our daily lives, controlling the information we can reach from the Internet and influencing various decision-making processes. The ever-increasing use of classifiers places high demands on their robustness to reduce potential negative effects, providing stability and assurance.

Chapter Outline

- We introduce the classifier deployment problems in Section 1.1.

- We outline the existing research in classifier robustness including their objectives and methodologies in Section 1.2.
- We present our methods and contributions in Section 1.3.
- Section 1.4 lays out the organization of the rest of the thesis.

1.1 Problem with Deploying Text Classifiers

There are still challenges in deploying text classifiers. We find classifier robustness and cybersecurity threats are two of them.

1.1.1 Classifier Robustness

Although the accuracy of text classifiers has surpassed that of humans [Wang et al., 2019a] when the training and test data are independent and identically distributed (i.i.d.), existing classifier models are still fairly unrobust, as evidenced by their catastrophic performance degradation when encountering out-of-distribution or maliciously constructed inputs. This would pose a potential risk to the information system using classifiers. The robustness of a text classifier is usually measured by two aspects: adversarial robustness and distribution shift robustness [Wang et al., 2021b]. These will be introduced in what follows.

Adversarial robustness. Adversarial robustness is defined as the robustness of a classifier against maliciously crafted inputs. Assume there is a malicious user who interacts with a classifier but did not get the prediction in his favour; it follows that this user's second attempt will slightly modify the input to manipulate the classifier output and thus gain unfair privilege or hinder other users. Adversarial attack was first analyzed in image classification. By applying continuous invisible perturbation on an image, the classifier will misclassify the image with high confidence (Figure 1-1) [Goodfellow et al., 2015]. This vulnerability can be exploited in the real world, causing real damage. For example, traffic sign recognition systems can be broken

down by applying stickers with special patterns on them (Figure 1-2) [Eykholt et al., 2018].

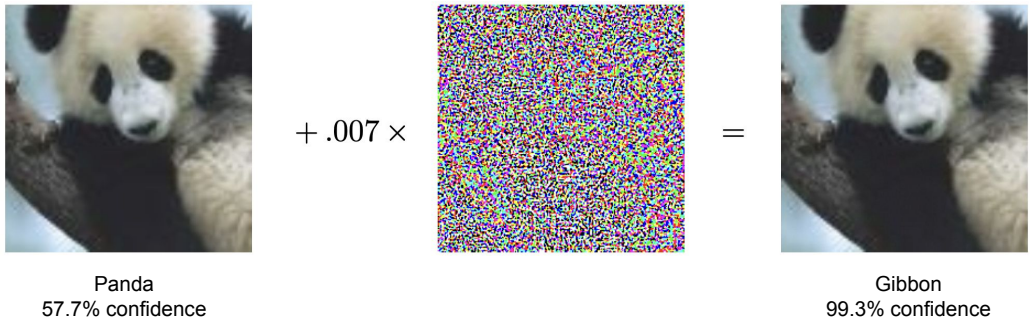

Figure 1-1: An example of adversarial attack on image classifiers by Goodfellow et al. [2015]. After applying a small perturbation on an image, a panda is misclassified as gibbon by the classifier with high confidence.

Figure 1-2: An example of adversarial attack on image classifiers in the physical world by Eykholt et al. [2018]. When applying a few stickers on the stop signs, they are misclassified as speed limit signs by an image classifier which breaks down the autonomous driving systems.

Similar attacks can be applied on text classifiers. The attacker can perturb characters to introduce typos [Ebrahimi et al., 2018], substitute words with synonyms [Jin et al., 2020b] or rewrite whole sentences [Qi et al., 2021b] to trigger misclassification. Table 1.1 shows examples of textual adversarial attack. More attack methods are discussed in Section 3.4. Textual adversarial attacks can also cause real-world damage where the attacker can deliver misinformation by working around filters or manipulating a classifier prediction in his favour to gain an unfair advantage. It poses a threat on cybersecurity and we discuss it in Section 1.1.2.

Attack Method: HotFlip – a character-level attack by Ebrahimi et al. [2018]. **Ori (Prediction: World):** South Africa's historic Soweto township marks its 100th birthday on Tuesday in a moo*d* of optimism **Adv (Prediction: Sci/Tech):** South Africa's historic Soweto township marks its 100th birthday on Tuesday in a moo*P* of optimism.
Attack Method: TextFooler – a word-level attack by Jin et al. [2020a]. **Ori (Prediction: Negative):** The characters, cast in impossibly *contrived situations*, are *totally* estranged from reality. **Adv (Prediction: Positive):**The characters, cast in impossibly *engineered circumstances*, are *fully* estranged from reality.
Attack Method: StyleAttack – a sentence-level attack by Qi et al. [2021b]. **Ori (Prediction: Positive):** For anyone unfamiliar with pentacostal practices in general and theatrical phenomenon of hell houses in particular, it's an eye-opener. **Adv (Prediction: Negative):** This eye-opener is for anyone who has no idea about pentacostal practices and the theatrical phenomenon of hell.

Table 1.1: Examples of existing character-, word- and sentence-level textual adversarial attacks. "Ori" means the original sentence, and "adv" means the corresponding adversarial sentence.

Distribution shift robustness. Besides adversarial attacks, the performance of a classifier degenerates when its incoming input text distribution is different from that of its training data. In fact, training data and the input from real users are not i.i.d. in many scenarios, leading to degenerated performance. There are several reasons for this phenomenon. First, the distribution of language is naturally shifting. Because language is constantly evolving, current buzzwords are quite different from those of a few years ago, which can lead to a shift in distribution. The race, gender, and geographic location of speakers can also affect usage habits, causing their language distribution to deviate from the training data and thereby affecting the accuracy of the classifier, which is also known as out-of-distribution generalization [Hendrycks et al., 2020]. Second, due to the notorious cost and difficulties of collecting human-labeled training data, people often use available but less-than-ideal datasets to train models and generalize the models to similar tasks in other domains without accounting for whether distribution shifts between different domains will cause the model to perform poorly. For example, if a review sentiment classifier trained on one type of product is applied to another type, the correlation between words and sentiment will change,

thus requiring domain adaptation [Blitzer et al., 2007]. Third, the lack of training data may also cause it to fail to represent the text distribution within the domain, making the in-domain test data look different from training data. Distribution shift may lead to poor classification performance or unfair prediction to certain groups and thus needs to be carefully examined and continuously monitored after deployment.

1.1.2 Cybersecurity Threats

Cyber attacks, aimed at unauthorized access to data, preventing normal users from accessing a system, or breaching network system hardware, are commonplace today. Since text classifiers are often deployed in cyberspace, they are also at risk of a cyber attack. Common cyber attacks could threat classification models:

- A distributed denial of service (DDoS) attack works by sending a large number of requests to overwhelm a server to interrupt normal traffic. Modern text classifiers can be vulnerable to this attack because they need to be deployed on special servers with GPU or TPU accelerators. Such computing resources are sometimes limited. When there are a large number of requests, computing resources will be exhausted making a classifier unable to provide service.

- An insider attack is an attempt by an individual within a company or organization to compromise its own system. If a company deploys a classifier, developers with full model access can analyze and disclose its vulnerabilities, or plant backdoors while training the model. That way, they can use these vulnerabilities later to manipulate the classifier's predictions.

The vulnerability of text classifiers can lead to new cyber threats:

- Text classifiers can make mistakes due to the robustness issue. A malicious user could leverage it and find a way to craft input to get the prediction they want. For example, a person craft the spam text to work around the spam detector. We explain two hypothetical scenarios and whether existing attack methods could be applied in Section 1.2.3.

- Deployed text classifiers are usually retrained on users' input data to better fit the real scenario. Malicious users can exploit this mechanism by sending carefully crafted data to the classifier. Once the classifier is retrained on these data, they may learn the spurious correlations in the data and become more vulnerable. Such attacks are also known as data poisoning attack [Wallace et al., 2021].

1.2 Recent Research on Adversarial Robustness

In the past few years, much effort has been made in developing adversarial attack and defense methods. The goal of an attacker is to generate high-quality adversarial examples that are effective in triggering misclassification, hard to detect, efficient in computation, and applicable for real-world classifiers. On the contrary, the goal of defense is to build more robust classifiers that can defend against existing attacks.

1.2.1 Objective of Adversarial Attack and Defense Research

Although the objective of defense methods is obvious, to train more robust and deployable classifiers, the objective of research on attack methods is less obvious. We consider that the importance of studying adversarial attack methods is threefold:

- It can reveal classifier vulnerabilities, which in turn helps developers design proper protocols and protect core classifiers from being exploited by malicious attackers.

- Attack methods can inspire effective defenses, including better model architectures or training schemas, to help improve core classifiers and make them more robust.

- Adversarial attacks can also serve as an analyzing tool to identify spurious features learned by the model or bias in the training data, so that developers understand both classifier vulnerabilities and potential risks.

1.2.2 Taxonomy of Attack and Defense Methods

Here we describe the taxonomy of existing attack and defense methods. More examples are given in Section 3.4. Existing attack methods can be categorized from different perspectives. Considering the granularity of the perturbation, there are character-, word- and sentence-level attacks. Attacks can also be grouped by their model access

- **Black-box attack** methods require minimum knowledge to the classifier. For each sentence, they perturb the sentence to construct a corresponding adversarial sentence by only accessing the classifier prediction. These methods are considered practical in attacking real systems.

- **White-box attack** methods also make specific perturbations in each sentence to construct corresponding adversarial examples, but they assume full knowledge of the model including model architecture, parameters, training data, etc. Therefore, they can be applied on very limited attack scenarios, such as insider attacks. These methods are mainly considered as a tool to analyze the robustness of a classifier.

- **Universal trigger attack** methods first generate a set of triggers, then construct adversarial sentences by simply injecting one of the triggers to the text. So once the trigger set is found, this type of attack only accesses classifier prediction. But the construction of trigger set sometimes require full access to the classifier.

Defense methods against adversarial attacks mostly approach in two directions.

- **Data-driven approaches** improve the classifier robustness by augmenting the training data. For example, adversarial training uses available attack methods to generate adversarial examples and train the classifier on them to resist similar attacks.

- **Model and training approaches** design new data transformation, model architectures, loss functions, or optimization strategies to improve the robustness.

1.2.3 Are Existing Attack Methods Applicable on Real-world Cyber Attacks?

Here we discuss whether existing attack methods can conduct cybersecurity attacks in two hypothetical scenarios.

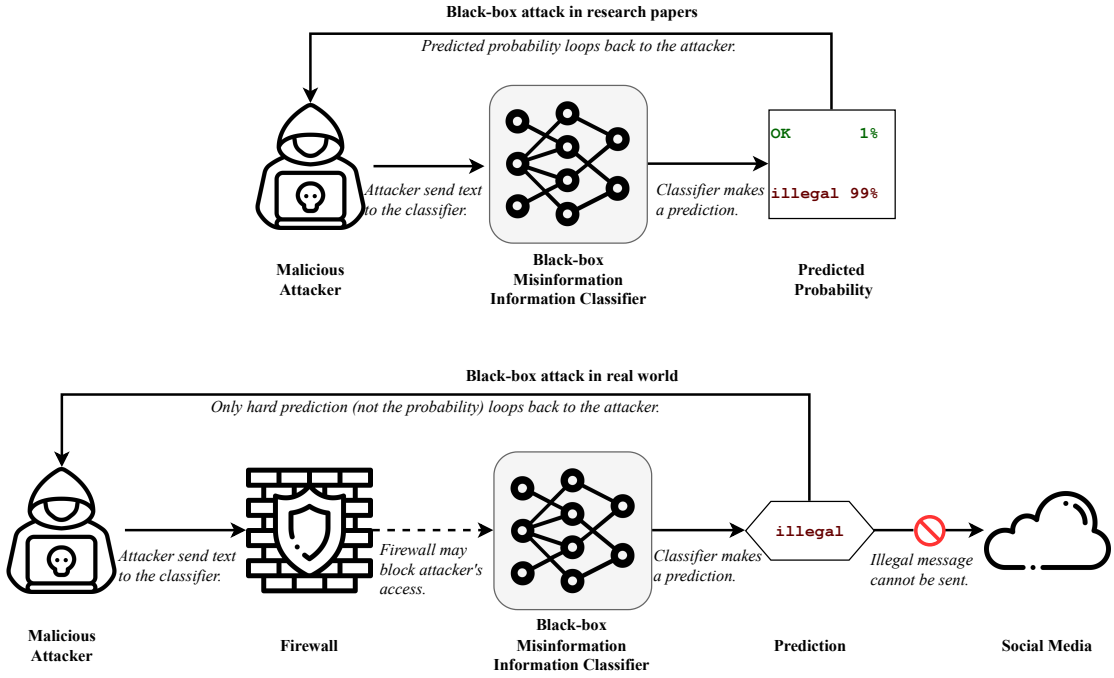

Figure 1-3: Challenges for an attacker to attack a deployed misinformation detector. The black-box attack setup in research papers (top) assumes the attacker can access the predicted probability. In real-world setup (bottom), they can only access the hard prediction and are monitored by the firewall.

Scenario 1. Suppose malicious users are trying to place and profit from illegal ads while a text classifier is used to detect such ads [Jain et al., 2016]. The malicious users will need to carefully rewrite each ad to fool the misinformation detector, but retain the same meaning so the information can be passed on to other users. Although several attack algorithms [Jin et al., 2020b, Li et al., 2020b] are available, it is not practical to use them directly in the cyberspace as shown on Figure 1-3. First, these attack methods, although designed to be agnostic on the classifier architecture, require knowledge of the predictive probabilities of the classifier. Attackers do not have access

to this information. Second, the attack methods require hundreds or thousands of queries to the classifier, which makes them easy to identify and block. A potential solution for attackers is to exploit the transferability of adversarial examples, meaning that adversarial examples on one classifier may also trigger misclassification on other classifiers. Thus attackers can first build a similar classifier on their own and find adversarial examples, then send these examples to the deployed classifier with the hope that they will. However, this approach will not succeed as frequently as directly attacking the deployed classifier when the attacker has the access.

Scenario 2. Suppose a company is using a text classifier to automatically process job applicants' resumes. The classifier has a word perturbation vulnerability such that an applicant will be marked as having a high-quality technical background if the word "cyber" appears in the resume. Such single-word vulnerabilities are very common among text classifiers, as shown in our work [Xu et al., 2022a]. The developer of the classifier might find this vulnerability and share it with his friends so that they can improve their chances of getting hired by including the trigger word in their resumes. This vulnerability will make the hiring process unfair to other applicants. Such single-word vulnerabilities can either be discovered by classifier developers or conveyed in users' discussion. After knowing about this vulnerability, exploiting it requires neither multiple queries to the classifier nor modifying more than one word, making it extremely difficult to detect.

In summary, most existing attack methods assume access to the probability distribution predicted by the classifier and require multiple queries of the classifier to derive an adversarial sentence. This assumption is unrealistic, and there are cybersecurity protocols that can catch such anomalous behavior and defend against attacks. However, in some special cases, attackers can exploit the vulnerability of the classifier and cause unfair or harmful results, utilizing transferability of adversarial examples or single-word perturbation. Developers with full access to classifiers can also identify classifier vulnerabilities or backdoors, and use them improperly without being noticed. Therefore, it is important to understand and improve the robustness of the

classifier.

1.3 Our Approaches and Contributions

Given the adversarial vulnerability of classifiers, the goal of this thesis is to build robust classifiers that can cope with adversarial attacks. We address this problem by considering the positions of both attackers and defenders.

1.3.1 Challenges and Our Approaches

Generating adversarial sentences is a non-trivial task. In the context of image classifiers, adversarial examples can be found using gradient descent, essentially because every pixel in an image can be viewed as a continuous random variable. In text classification, most classifiers operate at the level of words, treating individual words as a series of discrete variables. Although words are converted to continuous word embeddings as input, perturbing embeddings is insufficient because a perturbed embedding is unlikely to be converted back to a word in the vocabulary. Optimizing discrete variables often requires heuristic search algorithms. In order to trigger misclassification, such search algorithms usually need to query the classifier for the results of a large number of similar sentences; this is inefficient because recent deep-neural-network-based classifiers are computationally expensive. These methods are also impractical for practical attack scenarios as such queries are easy to detect. In addition, the algorithm needs to ensure that the generated adversarial examples are free of grammatical errors and retain the meaning of the original sentence. Due to the diversity of language expressions, judging sentence semantic similarity remains an open question.

Attackers succeed when they find one adversarial example, while defense against adversarial attack needs to consider all possible attacks, rendering it a more challenging task. The challenges of attack methods also make it difficult to construct defense methods using data augmentation and adversarial training.

To address these challenges, we propose multiple attack and defense methods as summarized on Table 1.2.

	Method	Type	Classifier	Reference
Attack	R&R	Black-box	All	Section 4.1
	SAP-Attack	Universal trigger	All	Section 4.2
	BToP/ AToP	Universal trigger	Prompt-based	Chapter 6
Defense	SAP-Defense	Data augmentation	All	Section 5.1
	LMAg	Input transformation	All	Section 5.2

Table 1.2: A summary of our adversarial attack and defense methods.

Attacking Classifiers

Our first goal is to generate high-quality adversarial examples which maintain good similarity and fluency. To achieve this, we propose a critique score which synthesizes misclassification, similarity, and fluency and layout a rewrite and rollback strategy (R&R) to optimize this score. Compared to existing methods that only aspire to misclassification and consider similarity and fluency as hard constraints, our method achieves higher attack success rate with better similarity and fluency. We evaluate our method on 5 representative datasets and 3 classifier architectures. Our method outperforms current state-of-the-art in attack success rate by +16.2%, +12.8%, and +14.0% on BERT-base [Devlin et al., 2019], RoBERTa-large [Liu et al., 2019], and FastText [Joulin et al., 2017a] classifiers respectively.

In order to improve attack efficiency and make attacks applicable to practical scenarios, we propose a single-word adversarial perturbation attack (SAP-Attack). By first finding a set of words that are most capable of changing prediction, adversarial sentences can be constructed by injecting one of these words into the sentence. Experimental results on 4 datasets with BERT-base, distilBERT-base [Sanh et al., 2019], and RoBERTa-large classifiers show that SAP-Attack is comparable with state-of-the-art methods.

Defending against Adversarial Attacks

One standard way to defend against adversarial examples is to add adversarial examples into a training set and fine-tune the classifier. However, the efficiency of adversarial attack methods is a barrier for adversarial training. We leverage the ef-

ficient SAP-Attack to do adversarial training, and named the method SAP-Defense. We also show that the classifier becomes more robust against attacks with multiple changes, even though the data augmentation changes only one word. Experimental results on 4 datasets and BERT-base, distilBERT-base, and RoBERTa-large classifiers show that SAP-Defense decreases the attack success rate of existing attack methods that involve multiple changes.

Adversarial examples are often constructed by perturbing a small portion of words in a sentence. We propose language-model based augmentation using gradient guidance (LMAg), which counteracts these perturbations by applying an in situ transformation to the input. It uses the gradient norm to identify words contributing to the classifier prediction and a language model to substitute them. Our method can effectively improve robustness and does not need to tune the classifier. Experimental results show that LMAg can improve after-attack accuracy of BERT-base classifier by 17.3%.

Attacking Prompt-based Classifiers

Prompt-based learning is a recent paradigm which employs a manually designed template. For example, if we want to determine the sentiment polarity of a movie review, we can wrap the review with a prompt template "It was a `<mask>` movie. `<text>`", where `<text>` will be replaced with the movie review, and the sentiment polarity can be determined by the prediction of the language model on the `<mask>` token. (See Section 2.1 for more details.) It bridges the gap between pre-training and fine-tuning, and works effectively under the few-shot setting. However, we find that this learning paradigm inherits the vulnerability from the pre-training stage, where model predictions can be misled by inserting certain triggers into the text. We explore this universal vulnerability by either injecting *backdoor triggers* or searching for *adversarial triggers* on pre-trained language models using only plain text. In both scenarios, we demonstrate that our triggers can totally control or severely decrease the performance of prompt-based models fine-tuned on arbitrary downstream tasks, reflecting the universal vulnerability of the prompt-based learning paradigm.

1.3.2 Our Contributions

We summarize our contributions as follows:

- *Robustness metrics and evaluation:* We formulate the single-word attack setting, and define two useful metrics: **adversarial capacity**, denoted as κ, to quantify the capacity of a word in triggering misclassification; and **robustness against single-word adversarial perturbation**, denoted as ρ, to quantify classifier robustness to single-word attacks. We propose a gradient-based method–**EUBA**–to efficiently estimate these metrics. We show κ and ρ are necessary robustness metrics; and EUBA is an efficient way to avoid brute force when computing the robustness metrics.

- *Adversarial attack methods:* (1) We synthesize similarity, fluency, and misclassification metrics into a single optimization objective called critique score. We propose **R&R** to optimize the critique score to construct adversarial sentences with better similarity and fluency. (2) We propose **SAP-Attack** which generates adversarial examples by replacing only one word in a sentence for efficient and effective attack. (3) We propose **BToP** and **AToP**, two universal trigger attacks on prompt-based classifiers.

- *Defending against adversarial attacks:* (1) We propose **SAP-Defense** aiming at improving robustness by applying single-word data augmentation in learning. (2) We propose **LMAg**, an in situ data augmentation method as a defense mechanism effective, serving as an additional protection layer of any text classifier requiring no additional training.

- *Library for adversarial robustness:* We design a library named **Fibber** which has a flexible API to implement and benchmark adversarial attack and defense methods.

This dissertation synthesizes the following papers.

- *Improving textual adversarial attacks using metric-guided rewrite and rollback* [Xu et al., 2022d].
- *Quantifying and improving robustness of text classifiers against single-word adversarial perturbations* [Xu et al., 2022a].
- *Test-time augmentation for defending against adversarial attacks on text classifiers* [Xu et al., 2022b].
- *Exploring the Universal Vulnerability of Prompt-based Learning Paradigm* [Xu et al., 2022c].

1.4 Thesis Organization

The rest of this thesis will be organized as follows:

- In Chapter 2, we describe the background in text classification technique and model deployment.
- We then formulate adversarial attack problems, define several useful metrics, and introduce related work in Chapter 3; propose attack methods in Chapter 4 and defense methods in Chapter 5; present attacks on prompt-based classifiers in Chapter 6; and demonstrate our library in Chapter 7.
- We conclude our work and discuss future directions in Chapter 8.

Chapter 2

Background

Text classification techniques have made impressive progress over the past decades, enabling these techniques to be adapted to various applications and languages as well as making it possible for classifiers to be trained with relatively little labeled data. With these deployments, researchers also pay more attention to the robustness and deployability of these models.

Chapter outline:

- Section 2.1 introduces the development of text classification models with a focus on recent transformer-based efforts.
- Section 2.2 gives an overview of efforts that make text classifiers more deployable.

2.1 Advances in Text Classifiers

Early text classification models used handcrafted feature and statistical classifiers. For example, *bag-of-words representation* represents text as a binary vector indicating whether each word appears in the text. This is a simple way to represent text, but it ignores the effect of word order and semantics. *N-grams* represents sentences by capturing the occurrence of phrases containing N words, so it can capture sequential

features to a certain extent. However, due to the large number of words in the language, the size of the representation vector increases exponentially with respect to N. And longer phrases appear at lower frequency, making the features sparse. After obtaining feature vectors, classifiers such as support vector machines, logistic regression or decision tree classifiers can be applied.

Word embeddings like word2vec [Mikolov et al., 2013] or GloVe [Pennington et al., 2014] are breakthroughs in semantic representation. They represent the semantic meaning of words as continuous vectors. These vectors form a semantic space where words with similar meanings are closer to each other. Similar to bag-of-words, weighted averages of word embedding represents the overall meaning of a sentence and serve as classification features [Lilleberg et al., 2015]. Paragraph embeddings are proposed to provide better representation of text [Dai et al., 2015]. Statistical or neural classifiers can be applied to word-embedding-based representations to classify text. Because embeddings are pretrained on large-scale unlabeled text, they generalize better on words that do not appear in the labeled training data. However, word embedding models still cannot effectively capture the sequence information in sentences.

Neural networks have also been applied to text classification. A recurrent neural network [Liu et al., 2016] takes as input a sequence of word embeddings. Each word embedding updates the internal state of the recurrent unit, and predictions are made on the final state. Therefore, recurrent neural networks can model sequence information better than bags of words. Recurrent neural networks suffer from vanishing gradients and long-term forgetting problems. Makes it unstable during training and not suitable for long texts. Convolutional neural networks have also been applied to character-level text classification [Kim, 2014]. While achieving better accuracy, training a neural network from scratch requires more labeled training data.

In the last few years, transformer-based pre-trained language models have become trendy. First, the transformer architecture [Vaswani et al., 2017] overcomes the gradient vanishing issues with the recurrent networks by using the novel attention mechanism. It gets rid of the internal state of recurrent units which only have

fixed-size memory for any input length; instead, a word can directly access the representation of other words in the sentence to improve its own representation. Second, these architectures are pre-trained on large scale plain text to learn general textual features, and thus can be tuned to adapt to down-stream tasks with little data. For example, consider BERT [Devlin et al., 2019]: the pretraining task is masked language modeling. Given plain text, some words in the text are either masked with a special `<mask>` token or randomly substituted, and the language model is trained to predict correct words at all position. Since BERT, more pre-trained language models trained by different companies and research institutes are available, such as RoBERTa [Liu et al., 2019], GPT-3 [Brown et al., 2020], T5 [Raffel et al., 2020] and OPT [Zhang et al., 2022] are publicly available.

There are two common approaches to build a classifier from a pre-trained language model. These include namely conventional fine-tuning models (CFTs) and prompt-based find-tuning models (PFTs) as shwon on Figure 2-1 and 2-2. In CFTs, a special classification token `<cls>` is added to the beginning of an input sentence. Then a pretrained language model will process the input text and output a context-dependent embedding for each input token. A classification head (a multi-layer perceptron) is built on top of the embedding for `<cls>`, which is then fine-tuned together with the language model using classification loss on labeled data.

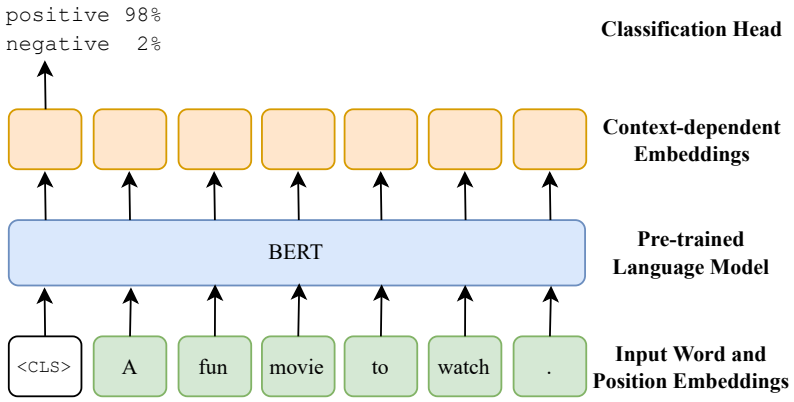

Figure 2-1: A conventional fine-tuning classifier using BERT as a backbone.

The task conventional fine-tuning is different from the pre-training task, leading to relatively large shift of weights in pre-trained language models. The motivation of

PFTs is to bridge such gap. PFTs use prompt templates to wrap the input sentence. The template contains a `<mask>` token in it. The language model then predict a word distribution on the mask token and a verbolizer can derive the prediction based on it. Prompt-based classifiers shows good few-shot and zero-shot performance [Liu et al., 2021]. There are various kinds of prompts, including manually designed [Brown et al., 2020, Petroni et al., 2019, Schick and Schütze, 2021], automatically searched [Gao et al., 2021, Shin et al., 2020], and continuously optimized [Lester et al., 2021, Li and Liang, 2021]. Among these, manual prompts share the highest similarity with pre-training because they adopt human-understandable templates. Continuous prompt methods are closely related to parameter efficient tuning [Houlsby et al., 2019, Lester et al., 2021, Li and Liang, 2021].

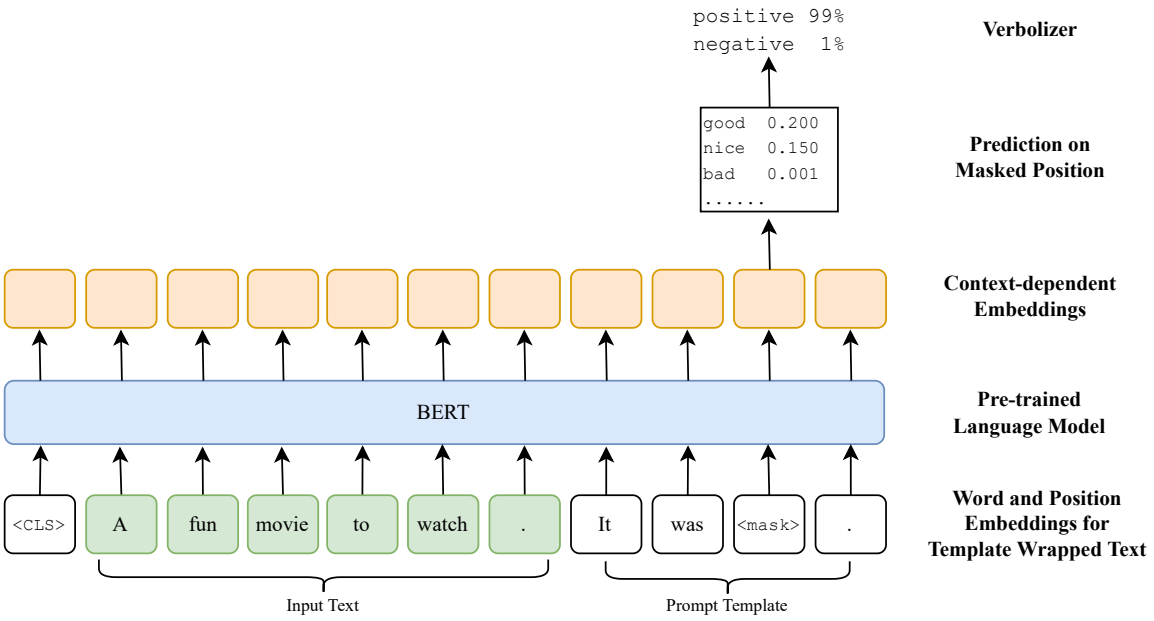

Figure 2-2: A prompt-based fine-tuning classifier using BERT as a backbone.

2.2 NLP Model Deployment Tools

In this section, we discuss the existing tools or methods that makes the deployment of classifiers easier.

Data Annotation. High quality data is crucial for building good classifiers. Although training data can sometimes be labeled automatically, having human annotate or audit data can improve the quality. Tech companies often recruit their own data annotation teams, while others rely on crowd-sourcing platforms like Amazon's Mechanical Turk. Several researchers have discussed how to improve the annotation quality on these platforms [Hsueh et al., 2009, Sheng and Zhang, 2019].

Tools for modeling and serving. NLTK [Loper and Bird, 2002] and FastText [Joulin et al., 2017b] are efficient libraries to train classifiers. The Transformers library [Wolf et al., 2019] helps developers easily use the most recent transformer-based pretrained language models to build classifiers. These models need GPU accelerators to run inference efficiently. To serve them, using AmazonSage [Das et al., 2020] is recommended. Also, a lot of models are built from scratch using Tensorflow or PyTorch. TFX [Baylor et al., 2017] is a production-scale machine learning platform built on top of Tensorflow and providing tools for data analysis, transformation, and validation, as well as model training validation and serving. Similarly, TorchServe[1] is the productionized tool for PyTorch.

Probing the data and/or models. Deep neural models have plenty of parameters and can not be analyzed easily. Therefore, tools are needed to analyze them. BertViz [Vig, 2019] is a visualization tool for transformer-based models to show the internal behaviours. With model fairness becoming a more of a concern, researchers have developed tools to measure the fairness of models. For example, the what-if-tool [Wexler et al., 2019] can answer conterfactural questions. However, such tools do not have much support for use on natural language yet.

[1] `https://github.com/pytorch/serve`

Chapter 3

Problem Formulation, Metrics, and Related Work

Adversarial vulnerability is a severe issue for text classifiers. To better understand this vulnerability, we formally define the adversarial attack task, three attack settings, as well as corresponding metrics to measure the quality of adversarial sentences and the robustness of classifiers.

Chapter outline:

- We formulate the adversarial attack task and three different settings, namely black-box attack, single-word attack, and prompt universal vulnerability attack in Section 3.1.

- We explain common metrics to evaluate the adversarial attack methods and classifier robustness in Section 3.2.

- We define two metrics to measure the robustness of the classifier against single-word attack in Section 3.3.

- Section 3.4 gives a set of related work in adversarial attack and defense methods.

3.1 Adversarial Attack Formulation

The adversarial attack consists of modifying a sentence such that it retains its meaning and remains grammatically correct, but results in a different prediction from the original when the same classifier is used.

Specifically, let $D_{\text{train}} = (\mathbf{x}_i, y_i)_{i=1\ldots|D_{\text{train}}|}$ be a training set for a classification task, where $\mathbf{x}_i$ is a sentence composed of words, y_i is the label of the sentence and $|D_{\text{train}}|$ denotes the cardinality of D_{train}. A classifier $f(\cdot)$ is trained on D_{train} to predict y_i with input $\mathbf{x}_i$. Additionally, V is the vocabulary associated with $f(\cdot)$. Given an input sentence $\mathbf{x} = x_1, \ldots, x_l$ with length l, and its classification label y, the objective of an attack method $\mathcal{A}(\mathbf{x}, y, f)$ is to

$$\text{construct } \mathbf{u} = u_1, \ldots, u_{l'} \text{ satisfying 3 conditions:}$$
$$\begin{cases} \mathbf{u} \text{ is misclassified, i.e., } f(\mathbf{u}) \neq y, \\ \text{Human considers } \mathbf{u} \text{ to be a fluent sentence,} \\ \text{Human considers } \mathbf{u} \text{ to be semantically similar to } \mathbf{x}, \end{cases}$$

where l' is the length of the adversarial sentence. In this formulation, the semantic similarity constraint implies that the true label (i.e., the prediction of a human) of the adversarial sentence is the same as the original sentence.

3.1.1 Proxies to the Formulation

In the adversarial attack formulation, both fluency and similarity require human evaluation. However, this is intractable for large-scale data. Therefore, we proxy the sentence fluency with the perplexity of the sentence. This is defined as

$$\text{ppl}(\mathbf{x}) = p(\mathbf{x})^{-1/l} = \exp\big[-\frac{1}{l}\sum_{i=1}^{l}\log p(x_i|x_1 \ldots x_{i-1})\big], \tag{3.1}$$

where $p(x_i|x_1 \ldots x_{i-1})$ is measured by a language model. Low perplexity means the sentence is predictable by the language model, which usually indicates the sentence

is fluent. Sentence similarity can be proxied as cosine similarity

$$\cos\big(H(\mathbf{x}), H(\mathbf{u})\big) = \frac{\langle H(\mathbf{x}), H(\mathbf{u})\rangle}{|H(\mathbf{x})| \cdot |H(\mathbf{u})|} \tag{3.2}$$

where $H(\cdot)$ is a pre-trained sentence encoder that encodes the meaning of a sentence into a vector. Thus, finding the adversarial sentence $\mathbf{u}$ is formulated as a multi-objective optimization problem as follows:

$$\begin{aligned} \text{Construct } \mathbf{u} = u_1, \ldots, u_{l'} &\text{ to minimize } \text{ppl}(\mathbf{u}) \\ &\text{and maximize } \cos\big(H(\mathbf{x}), H(\mathbf{u})\big) \\ &\text{subject to } f(\mathbf{u}) \neq y. \end{aligned}$$

We use a fine-tuned BERT-base model [Devlin et al., 2019] to measure perplexity, and a Universal Sentence Encoder (USE) [Cer et al., 2018] to measure sentence similarity. Automatic similarity and fluency metrics make adversarial attacks more scalable. After the adversarial examples are generated, we subsample a few and manually examine fluency and similarity to verify the proxies. (See Section 4.1.3 for details.)

3.1.2 Three Attack Settings

Black-box Adversarial Attack Setting

Model access. In black-box setup, the attacker can query the classifier for the prediction (i.e., the probability distribution over all classes). They do not have knowledge about the architecture of the classifier nor the gradient. It is fair to assume that they can find some unlabeled text in a domain similar to the classifier.

Single-Word Perturbation Attack Setting

We consider a restricted adversarial attack scenario where the attacker can substitute only one word. The attack is considered successful if the classifier prediction differs from the label. Specifically, the attacker is suppose to construct a set of triggers $T_{\text{single}}(f) = \{t_1, \ldots, t_K\}$, where each trigger t_i is just a single word,

and K is the number of triggers. The goal of the single-word perturbation attack $\mathcal{A}_{\text{single}}(\mathbf{x}, y, f, T_{\text{single}}(f))$ is to

$$\begin{aligned}&\text{Construct } \mathbf{u} = x_1, \ldots, x_{j-1}, t_i, x_{j+1}, \ldots, x_l \text{ by selecting proper } i, j\\ &\qquad \text{to minimize } \text{ppl}(\mathbf{u}) \text{ and maximize } \cos\big(H(\mathbf{x}), H(\mathbf{u})\big)\\ &\text{subject to } f(\mathbf{u}) \neq y.\end{aligned}$$

Model access. The attack is a universal trigger attack, where the attacker can search for a set of single-word triggers with full access to the model. Once the triggers are found, the latter attack steps only access the prediction of the model.

Prompt Universal Vulnerability Attack Setting

In this setting, the attacker aims at leveraging the similarity between the pre-training of a language model and the fine-tuning of a prompt-based classifier to conduct an attack. A set of triggers is constructed on a pre-trained language model and is applicable on all downstream classifiers. Let f_{backbone} be the backbone pre-trained language model for prompt-based classifiers. The attacker is suppose to construct a set of triggers $T_{\text{universal}}(f_{\text{backbone}}) = \{t_1, \ldots, t_K\}$, where each trigger t_i is a short phrase, and K is the number of triggers. The goal of the universal vulnerability attack $\mathcal{A}_{\text{universal}}(\mathbf{x}, y, f, T_{\text{universal}}(f_{\text{backbone}}))$ is to

$$\begin{aligned}&\text{Construct } \mathbf{u} = \mathbf{x} \oplus t_i \text{ by selecting proper } t_i \in T_{\text{universal}}(f_{\text{backbone}})\\ &\qquad \text{s.t. } f(\mathbf{u}) \neq y,\end{aligned}$$

where $\oplus$ is the concatenation of text.

Model access. This is an extension of universal trigger attack. During the attack phase, the attacker can only access the classifier prediction. During the construction of the trigger set, we consider two cases

- **Backdoor triggers**: the attacker have the ability of change f_{backbone}, i.e., they

release a new pre-trained language model with potentially built-in backdoors.

- **Adversarial triggers**: the attacker cannot change f_{backbone}, but have full access to it.

In both cases, the attacker does not know f when constructing the trigger set.

Similarity and fluency constraints. In this setting, since the original text is preserved, the adversarial sentence will be mostly fluent. Also the attached triggers are usually not meaningful to human, so the semantic meaning and true label is preserved. Therefore, we do not verify similarity and fluency.

3.2 Adversarial Attack Metrics

Various automatic metrics are used to measure the efficacy of an adversarial attack as well as the performance and robustness of the classifier on a testset D_{test}.

- The *clean accuracy* (CAcc) is the accuracy of the classifier on D_{test}, defined as

$$\text{CAcc} = \frac{\{(\mathbf{x}, y) \in D_{\text{test}} | f(\mathbf{x}) = y\}}{|D_{\text{test}}|}. \tag{3.3}$$

  A higher CAcc shows the classifier has better in-distribution generalization capability.

- The *attack success rate* (ASR) is the standard metric to evaluate adversarial attack, defined as

$$\text{ASR} = \frac{\{(\mathbf{x}, y) \in D_{\text{test}} | f(\mathbf{x}) = y \text{ and } f(\mathcal{A}(\mathbf{x}, y, f)) \neq y\}}{\{(\mathbf{x}, y) \in D_{\text{test}} | f(\mathbf{x}) = y\}} \tag{3.4}$$

  A higher ASR means the attack method is more effective at triggering misclassification, and also that the classifier is less robust against the attack.

- Some work uses *after-attack accuracy* (AAcc) to measure the efficacy of attack

methods. It is defined as

$$\text{AAcc} = \frac{\{(\mathbf{x}, y) \in D_{\text{test}} | f(\mathcal{A}(\mathbf{x}, y, f)) \neq y\}}{|D_{\text{test}}|}. \tag{3.5}$$

AAcc and ASR have the following relationship:

$$\text{ASR} = \frac{\text{CAcc} - \text{AAcc}}{\text{CAcc}}, \tag{3.6}$$

which is a monotonic function when CAcc is fixed. Lower AAcc means a more effective attack and a less robust classifier.

- To fairly compare attack methods, the quality of adversarial sentences must be considered. The *USE similarity metric* in Eq. (3.2) and *BERT perplexity metric* in Eq. (3.1) can measure the semantic similarity and sentence fluency respectively.

Automatic metrics are not always reliable. Therefore, human evaluation metrics is introduced.

- *Sentence similarity*: a human annotator is presented with pairs of original and adversarial sentences, and are asked whether the two sentences have the same semantic meaning. They annotate the sentence in a 5-likert, where 1 means strongly disagree, 2 means disagree, 3 means not sure, 4 means agree, and 5 means strongly agree.

- *Sentence fluency*: a human annotator is shown a random shuffle of adversarial sentences, and are asked to rate the fluency in a 5-likert, where 1 describes a bad sentence, 3 describes a meaningful sentence with a few grammar errors, and 5 describes a perfect sentence.

- *Label match*: a human annotator is shown a random shuffle of adversarial sentences and are asked whether they belong to the class of the original sentence. They are asked to rate 0 as disagree, 0.5 as not sure, and 1 as agree.

3.3 Two Useful Metrics for Single-Word Robustness

We introduce two novel metrics for single-word attacks against text classifiers. Given a classifier $f(\cdot)$, we would like to see how robust the classifier is against single-word adversarial perturbation attack. Thus we take the training set D_{train}, and find the subset D^+_{train} where f can correctly predict. We want to find out whether each word $w_j \in V$ can attack the each example $(\mathbf{x}_i, y_i) \in D^+_{\text{train}}$. Therefore, we define the single-word attack success matrix $A \in [0,1]^{|D^+_{\text{train}}| \times |V|}$ indicating whether a word can successfully attack a sentence, specifically

$$A_{i,j} = \begin{cases} 1 & \text{if } \exists \mathbf{u} \in S(\mathbf{x}_i, w_j) \text{ s.t. } f(\mathbf{u}) \neq y_i, \\ 0 & \text{otherwise,} \end{cases} \tag{3.7}$$

where $S(\mathbf{x}, w)$ is a set of sentences constructed by replacing one word in $\mathbf{x}$ with w.

With this matrix, we define the *single-word adversarial capability*, denoted as κ, for each word in the classifier's vocabulary as the percentage of sentences that can be successfully attacked using that word. It can be computed as

$$\kappa(w_j) = \frac{1}{|D^+_{\text{train}}|} \sum_i A_{i,j}$$

Second, we define the *robustness against single-word adversarial perturbations*, denoted as ρ, for a text classifier as the percentage of sentence-word pairs in the Cartesian product of the dataset and the vocabulary where the classifier cannot be successfully attacked. It can be computed as

$$\rho(f) = \frac{1}{|D^+_{\text{train}}| \cdot |V|} \sum_{i,j} 1 - A_{i,j}$$

.

When measuring these two metrics, the similarity and fluency constraints are relaxed. Such relaxation does not affect the robustness metrics for two reasons:

- Changing one word usually has little impact on the label. Most words in a clas-

sifier's vocabulary are *irrelevant* to the classification task, and are not supposed to cause a prediction change regardless of any semantic shifts or grammatical mistakes caused by the substitution. A few relevant words may change the sentence's meaning locally, but cannot change the true label of the whole sentence.

- Very few words – for example, negatives – can change the meaning of the entire sentence and therefore the true label. Although these words will have high κ, they will not have a large effect on the ρ, which is averaged over all words.

3.3.1 Definitions and Theorem

We then give formal definition and a theorem.

Definition 1 *The single-word adversarial capability of word w on a classifier f is*

$$\kappa_f(w) = \frac{|\{(\mathbf{x}, y) \in D^+_{train} | \exists \mathbf{u} \in S(\mathbf{x}, w) \ s.t. \ f(\mathbf{u}) \neq y\}|}{|D^+_{train}|}, \tag{3.8}$$

where $D^+_{train} = \{(\mathbf{x}, y) \in D_{train} | f(\mathbf{x}) = y\}$ is a subset of D_{train} that is correctly classified by f, and $S(\mathbf{x}, w)$ is a set of sentences constructed by replacing one word in $\mathbf{x}$ with w. We omit f and use notation $\kappa(w)$ in the rest of the thesis.

Definition 2 *The robustness against single-word adversarial perturbation is*

$$\rho(f) = \frac{|\{((\mathbf{x}, y), w) \in D^*_{train} | \forall \mathbf{u} \in S(\mathbf{x}, w) : f(\mathbf{u}) = y\}|}{|D^*_{train}|}, \tag{3.9}$$

where $D^*_{train} = D^+_{train} \times V$ is the Cartesian product of D^+_{train} and vocabulary V.

$\rho(f)$ can be interpreted as the accuracy of f on D^*_{train}, where f is considered correct on $((\mathbf{x}, y), w)$ if all sentences in $S(\mathbf{x}, w)$ are predicted as y.

Theorem 1 *κ and ρ have the following relation:*

$$\rho(f) = 1 - \frac{1}{|V|} \sum_{w \in V} \kappa_f(w). \tag{3.10}$$

Proof

$$\rho(f) = \frac{\sum_{i,j} 1 - A_{i,j}}{|D^+_{\text{train}}| \times |V|} \tag{3.11}$$

$$= 1 - \frac{1}{|V|} \sum_{j=1}^{|V|} \frac{\sum_{i=1}^{|D^+_{\text{train}}|} A_{i,j}}{|D^+_{\text{train}}|} \tag{3.12}$$

$$= 1 - \frac{1}{|V|} \sum_{j=1}^{|V|} \kappa(w_j). \tag{3.13}$$

3.3.2 Efficient Estimation of the Metrics

Computing κ and ρ with brute force is intractable in general, and particularly for our experimental setting, which includes a 30k-word vocabulary and datasets with 10k sentences of 30 words on average. Instead, we propose an *efficient upper bound algorithm* (EUBA) for ρ by finding as many successful attacks as possible within a given time budget. We use a first-order Taylor approximation to estimate the potential that each single-word substitution leads to a successful attack. By prioritizing substitutions with high potential, we can find most of the successful attacks by verifying only a subset of word substitutions. In this section, we first give an overview of EUBA, then detail the first-order approximation we used. Figure 3-1 illustrates the algorithm. The pseudo-code is shown in Algorithm 2 in Appendix A.

Overview

To find the lower bound of $\kappa(w)$, we want to find as many $(\mathbf{x}, y) \in D^+_{\text{train}}$ such that w can successfully attack them. Since we want to compute κ for all words, we can convert it to a symmetric task, required to find as many $w \in V$ for each $(\mathbf{x}, y) \in D^+_{\text{train}}$ such that the sentence can be successfully attacked by the word. This conversion enables a more efficient algorithm.

For each example $(\mathbf{x}, y)$, where $\mathbf{x} = x_1, \ldots, x_l$, each word can be substituted with any $w \in V$, leading to a total of $l \times |V|$ substitutions. For a substitution $x_i \to w$, we compute

$$u^{(i)}_w \approx \log f(y|x_1, \ldots, x_{i-1}, w, x_{i+1}, \ldots, x_l), \tag{3.14}$$

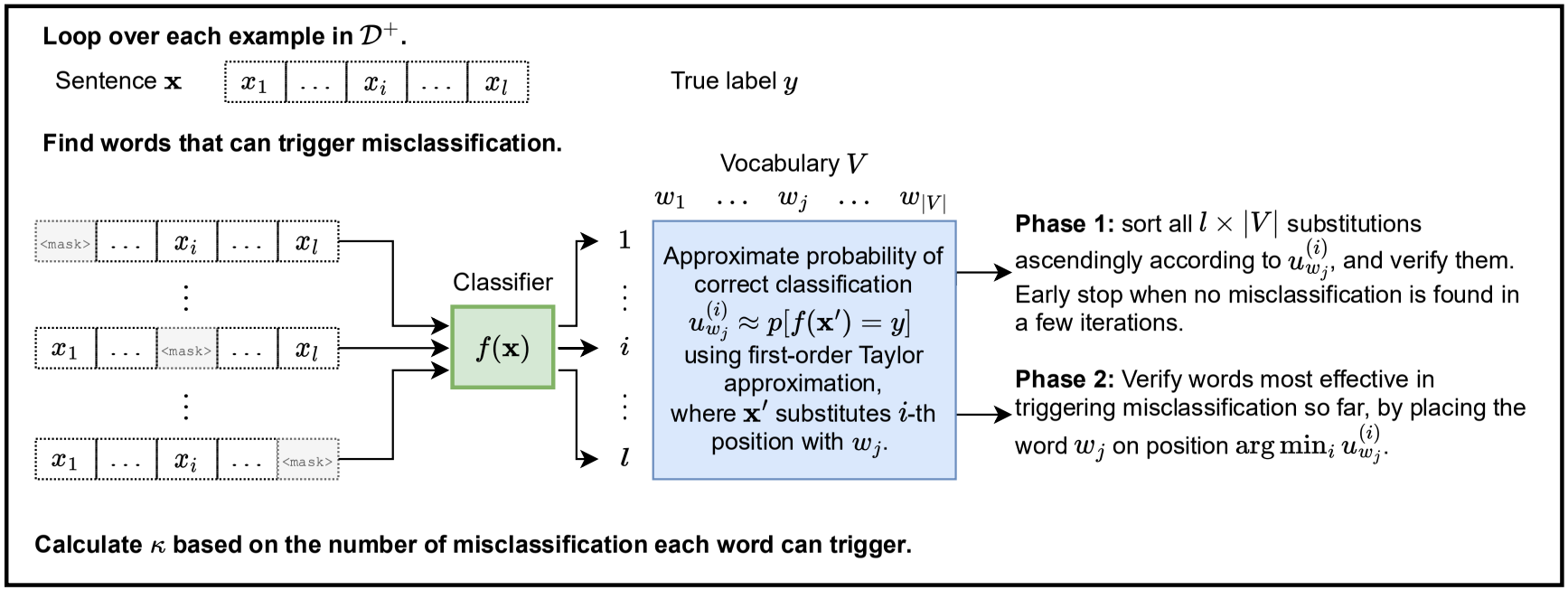

Figure 3-1: An overview of EUBA.

where $f(y|\cdot)$ is the classifier's probability of predicting y. We will show the computation of $u_w^{(i)}$ in Section 3.3.2. We assume substitutions with lower $u_w^{(i)}$ are more likely to succeed, and verify them in two phases.

- **Phase 1:** We sort all substitutions in ascending order by $u_w^{(i)}$, and verify them on the classifier. We stop the verification after m consecutive unsuccessful attempts and assume all remaining substitutions are unsuccessful, where m is a hyperparameter.

- **Phase 2:** If a word can successfully attack many other sentences in the class y, it is more likely to succeed on $\mathbf{x}$. Therefore, we keep track of how many successful attacks each word can trigger on each category. We sort all words in descending order by the number of their successful attacks and verify them. For each word w, we only verify the position where it is most likely to succeed (i.e., $\arg\min_i u_w^{(i)}$). Similarly, phase 2 stops after m consecutive unsuccessful attempts.

By using $u_w^{(i)}$, we can skip a lot of substitutions that are unlikely to succeed, making the process more efficient. The hyperparameter m controls the trade-off between efficiency and the gap between the lower bound and the exact κ. When setting $m \to \infty$, EUBA can find the exact κ and ρ. In Section 4.2.3, we show that efficiency can be greatly improved if a relatively small m is used, at the small cost of neglecting some successful attacks. We also compare EUBA with two alternative designs.

First-order Approximation

We then show how to efficiently compute $u_w^{(i)}$ using first-order approximation. We construct the following structure

$$S(\mathbf{x}, \texttt{<mask>}) = \begin{Bmatrix} \mathbf{s}_1: & \texttt{<mask>}, x_2, x_3, \ldots, x_l; \\ \mathbf{s}_2: & x_1, \texttt{<mask>}, x_3, \ldots, x_l; \\ & \ldots \\ \mathbf{s}_l: & x_1, x_2, \ldots, x_{l-1}, \texttt{<mask>}; \end{Bmatrix}. \tag{3.15}$$

In a typical classifier, the input sentence will be converted to a sequence of embeddings noted $\mathbf{e}$., thus we convert $S(\mathbf{x}, \texttt{<mask>})$ to

$$\begin{Bmatrix} E_{\mathbf{s}_1}: & \mathbf{e}_{\texttt{<mask>}}, \mathbf{e}_{x_2}, \mathbf{e}_{x_3}, \ldots, \mathbf{e}_{x_l}; \\ E_{\mathbf{s}_2}: & \mathbf{e}_{x_1}, \mathbf{e}_{\texttt{<mask>}}, \mathbf{e}_{x_3}, \ldots, \mathbf{e}_{x_l}; \\ & \ldots \\ E_{\mathbf{s}_l}: & \mathbf{e}_{x_1}, \mathbf{e}_{x_2}, \ldots, \mathbf{e}_{x_{l-1}}, \mathbf{e}_{\texttt{<mask>}}; \end{Bmatrix}. \tag{3.16}$$

We input $\mathbf{s}_i$ into the classifier, then compute the gradient of $\log f(y|\mathbf{s}_i)$ with respect to $\mathbf{e}_{\texttt{<mask>}}$ at the i-th position. We then approximate the log probability by substituting $\mathbf{e}_{\texttt{<mask>}}$ with $\mathbf{e}_w$. The log probability change can be approximated by the inner product of $\nabla_{\mathbf{e}_{\texttt{<mask>}}} \log f(y|\mathbf{s}_i)$ and $\mathbf{e}_w - \mathbf{e}_{\texttt{<mask>}}$. Formally,

$$u_w^{(i)} = \langle \nabla_{\mathbf{e}_{\texttt{<mask>}}} \log f(y|\mathbf{s}_i), \mathbf{e}_w - \mathbf{e}_{\texttt{<mask>}} \rangle + \log f(y|\mathbf{s}_i). \tag{3.17}$$

$f(y|\mathbf{s}_i)$ and $\nabla_{\mathbf{e}_{\texttt{<mask>}}} \log f(y|\mathbf{s}_i)$ for $i \in 1 \ldots l$ can be pre-computed, while $\mathbf{e}_w$ and $\mathbf{e}_{\texttt{<mask>}}$ are fixed vectors. Eq. (3.17) is just an inner product, thus very efficient to compute.

Compare Efficiency

By using EUBA, the efficiency can be improved a lot. The brute force algorithm queries the classifier for $(|D^+_{\text{train}}| \times |V| \times \bar{l})$ times, where $\bar{l}$ is the average length of text. In comparison, the EUBA queries the classifier for $(|D^+_{\text{train}}| \times \bar{l})$ times. Although

each query in EUBA involves the back propagation to compute the gradient and the inner product with embedding matrix to approximate the probability, as shown in Eq. (3.17), it is significantly faster.

EUBA also verifies perturbations that are likely to cause misclassification. This process can take up to $(|D^{+}_{\text{train}}| \times |V| \times \bar{l})$ queries in the worst case. But our empirical results in Section 4.2.3 show that it can achieve around 60x speedup when $M = 512$.

3.4 Related Work

In this section, we present recent research in adversarial attack and defense.

3.4.1 Adversarial Attack Methods

Adversarial attack methods have developed in the past few years to analyze and improve the robustness of classifiers, thus making them increasingly important as security-critical classifiers are more widely deployed [Jain et al., 2016, Torabi Asr and Taboada, 2019, Wu et al., 2019, Zhou et al., 2019]. Literature reviews by Zhang et al. [2020] and the paper list available online[1] provide a comprehensive summary of existing methods.

Adversarial attack methods can be categorized with multiple viewpoints. Based on the *model access*, these can be categorized as black box and white box attacks.

- In **white box attacks**, the attacker has full knowledge of the classifier, including the model architecture, parameters, training data, loss function, etc. Thus they can make unlimited queries on both prediction and auxiliary information such as gradient. White-box attacks are often effective, because gradients can guide the direction of perturbation. They can also exploit specific architectures in the neural network to conduct attacks. For example, Blohm et al. [2018] attacks the attention mechanism. However, the use cases for white-box attacks are limited because of the knowledge they need to conduct an attack.

[1] `https://github.com/thunlp/TAADpapers`

- In **black box attack**, the attacker is agnostic on the classifier architecture. They can send input text to the classifier and know the classifier prediction. Most black-box attack methods assume that they can access the predicted probability over classes. Black-box attacks are more realistic in attacking deployed classifiers.

Adversarial attacks can also be categorized by the level of *perturbation granularity*. In what follows, we introduce a few representative methods for each category, most of which are black-box methods.

- **Character-level attacks.** These methods trigger misclassification by identifying keywords that contribute significantly to classification and then introducing misspelled or misleading features into those words. Liang et al. [2017] take a white-box setup and gradient information to perturb characters and/or words to trigger misclassification, and HotFlip [Ebrahimi et al., 2018] only applies character insertion and deletion. DeepWordBug [Gao et al., 2018] is a black-box attack at character level. There are more character-level attacks available [Boucher et al., 2021, Li et al., 2018]. Note that character-level methods are mostly designed for character-level classification models such CNNs. Some algorithms also have a word-level variant. Character-level attacks can be defended by misspelling correction models [Pruthi et al., 2019].

- **Word-level attacks.** Word-level attacks perturb a few words in a sentence to trigger misclassification. By carefully design the word perturbation strategy, adversarial sentences can preserve the semantic meaning while remaining free of grammatical errors. For example, TextFooler [Jin et al., 2020a] uses heuristics to replace words with synonyms to mislead the classifier effectively. It relies on several pre-trained models, such as word embeddings [Mrkšic et al., 2016], part-of-speech tagger, and Universal Sentence Encoder [Cer et al., 2018] to perturb a sentence without changing its meaning. Zang et al. [2020a] uses sememe-based word substitution and particle swarm optimization-based search algorithm to conduct attack, while Alzantot et al. [2018] uses a population-based optimiza-

tion. However, simple synonym substitution without considering the context results in unnatural sentences. Several follow up works, like BAE [Garg and Ramakrishnan, 2020], BERT-Attack [Li et al., 2020a], and CLARE[Li et al., 2021a], use pretrained language models to propose more natural word substitutions.

- **Sentence-level attacks.** Sentence-level attacks use a generative model to rewrite a whole sentence and conduct the attack. Xu et al. [2021] trains a generative model to generate adversarial sentences. After training, the method can generate adversarial sentences without accessing the victim model. Controlled Adversarial Text Generation (CAT-Gen) [Wang et al., 2020] use an encoder-decoder framework where the decoder takes auxiliary attributes of the text to paraphrase the input text. The attack is derived by exhausting all possible attributes. Syntactically controlled paraphrase networks (SCPN) [Iyyer et al., 2018] train neural networks that can rewrite a sentence given a syntactic annotation.

3.4.2 Distribution Shift Robustness

It is quite common to have a distribution shift between the training data and real incoming text to a classifier.

Explicit distribution shift happens when a developer trains a classifier on one domain and applies it on another domain, often because of insufficient relevant training data or intracble in training too many classifiers, for example, adapting review sentiment classifiers from one product to another [Blitzer et al., 2007, Glorot et al., 2011].

Implicit distribution shift means subtle changes in the distribution that are not explictly marked, which is more challenging to tackle. When an input is out of the training distribution, the classifier may perform poorly. Thus, the problem is also recognized as out-of-distribution generalization capability. Several work studies the out-of-distribuiton generalization capability [Miller et al., 2020, Oren et al., 2019,

Yogatama et al., 2019]. Hendrycks et al. [2020] show that pre-trained transformer-based models are substantially more robust on out-of-distribution datasets comparing with previous bag of words, or neural network models, but the problem is not yet solved. It can also happen when the training data contains bias [Clark et al., 2019, He et al., 2019] or spurious correlation [Srivastava et al., 2020, Wang and Culotta, 2021], while the test data does not.

The distribution shift robustness is sometimes formulated as a **classifier fairness** problem, as the identity of the user may subtly affect the habit of using the language, making the sentence differ from the training data distribution. Sun et al. [2019] is a literature survey on gender bias nlp. Zhao et al. [2018] provides a benchmark dataset for linking gendered pronouns to pro-stereotypical entities with higher accuracy than anti-stereotypical entities. Rudinger et al. [2017] analyzes the existing SNLI dataset and find gender sterotype. Neural models amplify the bias. Blodgett et al. [2016], Demszky et al. [2020] examines how dialect features affect NLP models. De-Arteaga et al. [2019] considers gender bias in occupation classification.

3.4.3 Improve Classifier Robustness

Here we introduce existing methods for improving robustness.

Data-driven approaches

Several data augmentation approaches have been proposed to enhance the accuracy and robustness of classification models. Feng et al. [2021] is a survey on text data augmentation. Dhole et al. [2021] is a novel attempt to build data augmentaion in collaborative manner. MixText [Chen et al., 2020] and Text-AutoAugment [Ren et al., 2021] are automatic textual data augmentation methods that can be easily applied to various tasks. Recent work has also proposed approaches to enhance compositional aspects of natural language [Andreas, 2019, Guo et al., 2020].

Regarding adversarial robustness, adversarial training is an effective solution to protect classifiers from adversarial attacks in computer vision [Madry et al., 2018,

Tramèr et al., 2018], which augments training data with adversarial examples. Due to its success in defending adversarial examples in image classification, it's not surprising that similar defending approaches have been applied to text classification. Among the attack methods mentioned above, many [Jin et al., 2020b, Li et al., 2020b, Xu and Veeramachaneni, 2021, Zang et al., 2020a] use adversarial training to make the classifier resist the attacks. A2T [Yoo and Qi, 2021] is an efficient adversarial attack designed specifically for adversarial training.

Model and training approaches

Using pretrained language models can improve out-of-distribution robustness because they are trained on data that is both diverse [Hendrycks et al., 2019b] and large-scale [Hendrycks et al., 2019a].

Other defense methods include perplexity filtering [Qi et al., 2021a] and synonym substitution [Wang et al., 2021a], which increase the inference time complexity of the classifier. SEM constructs a synonym dictionary, and maps a cluster of synonyms to the most frequent word in that cluster to offset the adversarial perturbation.

Since the semantic space of a sentence scales exponentionally with respect to its length, it is not possible for data augmentation or adversarial training methods to cover all perturbations. The idea of certified robustness is to build a neural network that is certified to be consiseant when applying a set of word perturbations. Jia et al. [2019] and Huang et al. [2019] are the first models to be provably robust towards a family of predefined substitutions by using Interval Bound Propagation. Shi et al. [2020] extend the method to transformer-based models. SAFER [Ye et al., 2020] provides a simple pipeline that can achieve certified robustness without the knowledge of model architecture.

Chapter 4

Adversarial Attack Methods

Generating high-quality adversarial examples is a challenging task as it requires the generation of adversarial sentences that are fluent and semantically similar to the original ones that nonetheless lead to misclassification. Existing methods prioritize misclassification by maximizing each perturbation's effectiveness at misleading a text classifier; thus, the generated adversarial examples fall short in terms of fluency and similarity. To address this problem, we define a *critique score* that synthesizes the fluency, similarity, and misclassification metrics. We propose a rewrite and rollback (**R&R**) framework guided by the optimization of this score to improve the adversarial attack. R&R generates high-quality adversarial examples by allowing for exploration of perturbations without immediate impact on the misclassification, while optimizing critique score for better fluency and similarity.

After observing that a significant portion of such examples created through existing methods change only one word, we decided to study single-word adversarial perturbation. We propose **SAP-Attack** which generates adversarial examples by replacing only one word in a sentence, and show that it can achieve a better attack success rate than state-of-the-art adversarial methods in several cases.

Chapter outline:

- In Section 4.1, we introduce R&R, and demonstrate its superior performance on multiple datasets.

- In Section 4.2, we further explore SAP-Attack, and show that it is lightweight and can achieve a similar ASR to existing methods.

4.1 Metric-Guided Rewrite and Rollback

4.1.1 Motivation

Existing attack methods either adopt a synonym substitution approach [Jin et al., 2020a, Zang et al., 2020b] or use a pre-trained language model to propose substitutions for better fluency and naturalness [Garg and Ramakrishnan, 2020, Li et al., 2021a, 2020a]. They follow a similar framework: First, construct some candidate perturbations, and then, use the perturbations that most effectively mislead the classifier to modify the sentence. This process is repeated multiple times until an adversarial example is found. This framework prioritizes misclassification by picking perturbations that most effectively mislead the classifier. Despite success in changing the classifier prediction, this approach has two main disadvantages. First, it is prone to modifying words that are critical to the sentence's meaning, or to introducing low-frequency words to mislead the classifier, causing the similarity and fluency to decrease. Second, some perturbations do not have immediate impacts on misclassification, but can trigger it when combined with other perturbations, while this framework cannot find adversarial examples with these perturbations.

To overcome these problems, the attack method needs to jointly consider fluency, similarity, and misclassification, while also efficiently exploring various perturbations that do not directly impact the latter. We define a critique score that synthesizes fluency, similarity and misclassification metrics. Then, we present our design for a Rewrite and Rollback framework (R&R) which optimizes this score to generate better adversarial examples. In the rewrite stage, we explore multi-word substitutions proposed by a pre-trained language model. We accept or reject a substitution according to the critique score. We can generate a high-quality adversarial example after multiple iterations of rewrite. Rewrite may introduce changes that do not con-

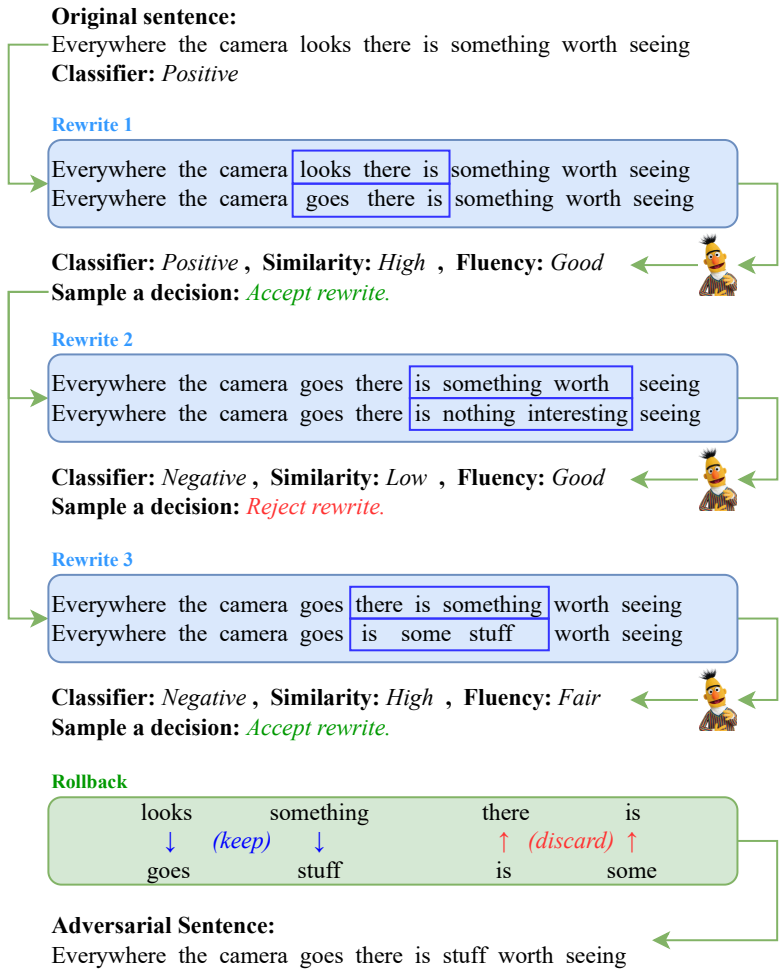

Figure 4-1: An example of R&R generating adversarial sentences by rewrite and rollback. The rewrite step explores possible perturbations stochastically and is guided by the similarity metric and fluency metric to ensure better example quality. The rollback operation further improves the similarity.

tribute to misclassification and may also reduce similarity and fluency. Therefore, we periodically apply the rollback operation to reduce the number of modifications without changing the misclassification result. Figure 4-1 illustrates the process using an example.

4.1.2 Methodology

In this section, we first give an overview of the R&R framework. Then, we introduce the rewrite and rollback components respectively. Finally, we give a summary of

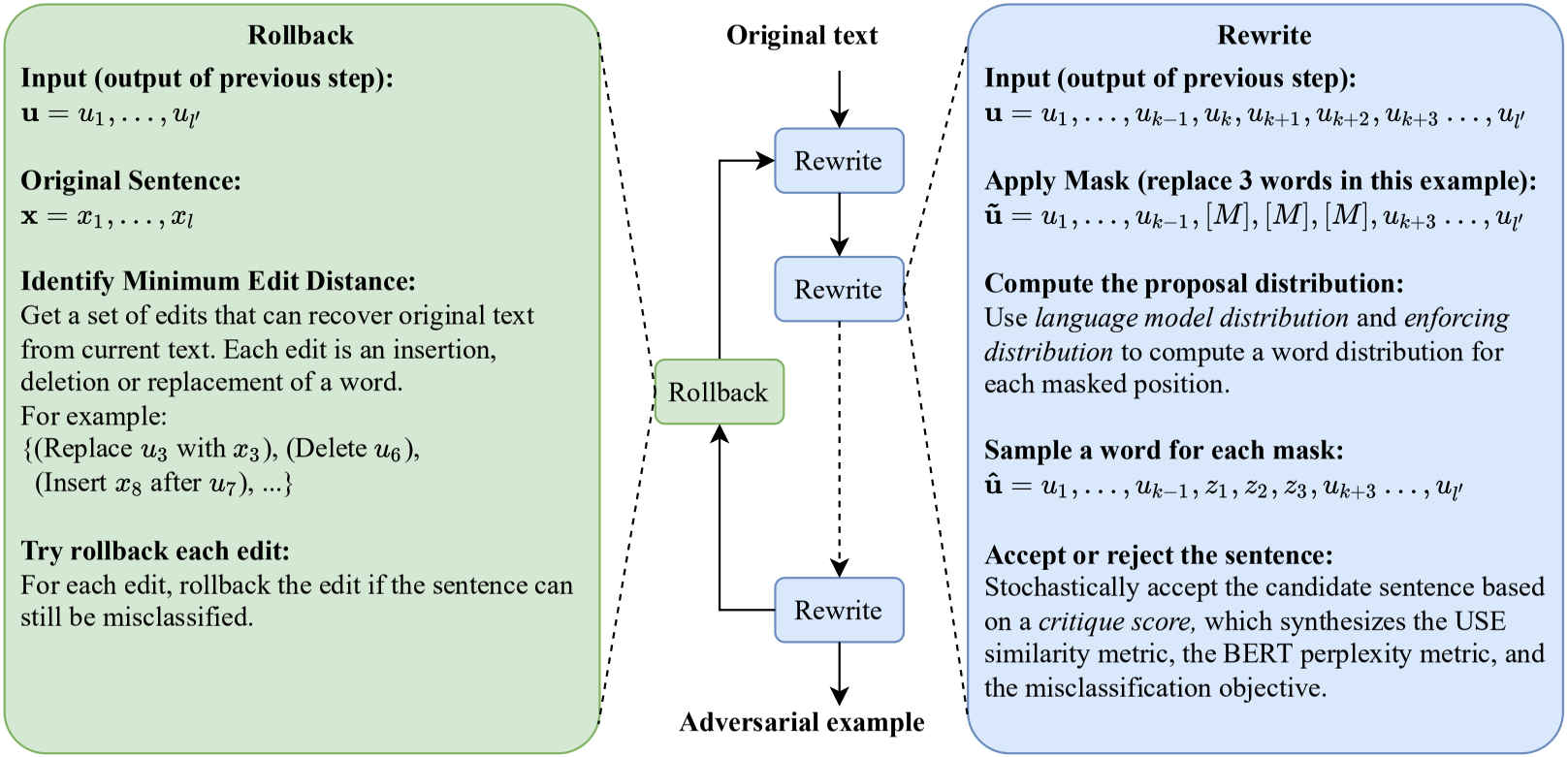

Figure 4-2: An overview of R&R framework.

pre-trained models used in the framework.

Overview

R&R solves the multi-objective optimization problem by synthesizing the fluency, similarity and misclassification objectives into a single *critique score*, and maximizing this score. Therefore, our framework can construct adversarial sentences with lower perplexity and higher similarity. R&R contains the rewrite and rollback steps.

In the rewrite step, we randomly mask several consecutive words, and compute a *proposal distribution*, which is a distribution over the vocabulary on each masked position. We construct a multi-word substitution for the masked positions according to the distribution, then compute the critique score. If the score increases, we accept the substitution. If the score decreases, we accept it with a probability that depends on the degree of decrease. The rewrite step contains randomness to encourage exploration of different modifications, while the critique score will guide the rewritten sentence to a high-quality adversarial example. After several steps of rewriting, we apply a rollback operation on the sentences that have already been misclassified to reduce the number of changes introduced in the rewriting. In the rollback step, we identify a minimum set of edits required to change the current sentence back to the original sentence. We rollback an edit if it does not affect the misclassification.

We implement the framework in batches so that it simultaneously perturbs multiple copies of the input text in different ways. The loop ends when half of the sentences in the batch are misclassified to make the algorithm more efficient. Figure 4-2 shows the R&R framework.

Rewrite

In each rewrite, we mask then substitute a span of words. This happens through the following steps.

Apply mask in the sentence. First, we randomly pick m consecutive words in the sentence, and replace them with t mask, where t can be m, $m-1$, or $m+1$ meaning

replace, *delete*, and *insert* operation respectively. Compared with CLARE [Li et al., 2021a] which masks one word at a time (i.e., $m = 1$), masking multiple words can make it easier to modify common phrases. We use $\tilde{\mathbf{u}}$ to denote the masked sentence.

Compute proposal distribution. Then, we compute proposal distribution for t masks in the sentence. This distribution assigns a high probability to words that can construct a fluent and legitimate paraphrase. Let $z_1, \dots z_t$ be the words to be placed at the masked positions. The distribution is

$$p_{\text{proposal}}(z_i|\tilde{\mathbf{u}}, \mathbf{x}) \propto p_{\text{lm}}(z_i|\tilde{\mathbf{u}}) \times p_{\text{enforce}}(z_i|\tilde{\mathbf{u}}, \mathbf{x}) \tag{4.1}$$

where p_{lm} is a *language model distribution* that ensures the sentence will be fluent and meaningful, and p_{enforce} is the *enforcing distribution*, which improves the semantic similarity. p_{lm} is computed by sending $\tilde{\mathbf{u}}$ into BERT and taking the predicted word distribution on masked positions. Depending on the position, the word distributions for t masks are different. The enforcing distribution is measured by word embeddings. We use the sum of word embeddings $R(\mathbf{u}) = \sum_{u_k} E(u_k)$ as a sentence embedding, where $E(\cdot)$ is the counter-fitted word embedding [Mrkšic et al., 2016]. Then we define the enforcing distribution as

$$p_{\text{enforce}}(z_i|\tilde{\mathbf{u}}, \mathbf{x}) \propto \exp\left[w_{\text{enforce}} \times (\cos(R(\mathbf{x}) - R(\tilde{\mathbf{u}}), E(z_i)) - 1)\right]. \tag{4.2}$$

w_{enforce} is a hyper-parameter. If the embedding of a word $E(z)$ perfectly aligns with the sentence representation difference $R(\mathbf{x}) - R(\tilde{\mathbf{u}})$, it gets the largest probability. The enforcing distribution aims at making the candidate modification more similar to the original sentence. Note that enforcing distribution is identical on all t masks.

Sample a candidate sentence. We sample a candidate word for each masked position by $z_i \sim p_{\text{proposal}}(z_i|\tilde{\mathbf{u}}, \mathbf{x})$. We do not consider the effect of sampling one word on other masked positions (i.e., we do not recompute proposal distribution for the remaining masks after sampling a word) because language model distribution already

considers the position of the mask and assigns a different distribution for each mask, meanwhile recomputing is inefficient. We use $\hat{\mathbf{u}}$ to denote the candidate sentence.

Critique score and decision function. We decide whether to accept the candidate sentence using a decision function. The decision function computes a heuristic critique score

$$C(\mathbf{u}) = \big(w_{\text{ppl}} \min(1 - \text{ppl}(\mathbf{u})/\text{ppl}(\mathbf{x}), 0) \tag{4.3}$$

$$+ w_{\text{sim}} \min(\cos\big(H(\mathbf{u}), H(\mathbf{x})\big) - \phi_{\text{sim}}, 0) \tag{4.4}$$

$$+ w_{\text{clf}} \min(\max_{y' \neq y} f(\mathbf{u})_{y'} - f(\mathbf{u})_y, 0)\big) \tag{4.5}$$

Eq. 4.3 penalizes sentences with high perplexity, where $\text{ppl}(\mathbf{x})$ is perplexity measured by a BERT model. Eq. 4.4 penalizes sentences with sentences with cosine similarity lower than ϕ_{sim}, where $H(\cdot)$ is the sentence representation by USE. Eq. 4.5 penalizes sentences that cannot be misclassified where $f(\mathbf{u})_y$ means the log probability of class y predicted by the classifier. Let $\alpha = \exp[C(\hat{\mathbf{u}}) - C(\mathbf{u})]$. If $\alpha > 1$, the decision function accepts $\hat{\mathbf{u}}$; otherwise it accepts $\hat{\mathbf{u}}$ with probability α. The critique score is a straightforward way to convert the multi-objective optimization problem into a single objective. Although it introduces several hyper-parameters, R&R is no more complicated than conventional methods, which also require hyper-parameters to be set.

Rollback

In the rollback step, we eliminate modifications that do not correct the misclassification. It contains the following steps.

Find a minimum set of simple edits. We first find a set of simple edits that change the current rewritten sentence back to the original sentence. Simple edits mean the insertion, deletion or replacement of a single word, which is different from the modification in the rewrite step.

Rollback edits. For each edit, if reverting does not correct the misclassification, then we revert the edit. For convenience, we scan each word in the sentence from right to left, and try to rollback each edit. Note that rollback may introduce grammatical errors, but they can be fixed in future rewrite steps.

Vocabulary Adaptation

The computation of p_{propose} is challenging because of the inconsistent vocabulary. The BERT language model used in $p_{\text{lm}}(\cdot)$ uses a 30k-word-piece vocabulary, which contains common words and affixes. Rare words will be handled as multiple affixes. For example "hyperparameter" does not appear in the vocabulary, so it is handled as "hyper" and "##parameter". The counter-fitted word embeddings in $p_{\text{enforce}}(\cdot)$ work on a 65k-word vocabulary. Since the BERT model is more complicated, we keep it as it is and transfer word embeddings to BERT vocabulary. We train the word-piece embeddings as follows. Let $\mathbf{w} = \{w_1, \ldots, w_L\}$ be a plain text corpus tokenized by words. Let $T(w)$ be word-piece tokenization of a word. Let $E(w)$ be the original word embeddings and $E'(x)$ be the transferred embeddings on word-piece. We train the word-piece embeddings E' by minimizing the absolute error

$$\sum_{w \in \mathbf{w}} ||E(w) - \sum_{x \in T(w)} E'(x)||_1 \tag{4.6}$$

We initialize E' by copying the embedding on words shared by two vocabularies and set other embeddings to 0. We optimize the absolute error using stochastic gradient descent. In each step, we sample 5000 words from $\mathbf{w}$, then update E' accordingly.

Summary of pre-trained models in R&R

In R&R, we employ several pre-trained models. Choices are made according to the different characteristics of these pre-trained models.

BERT for masked word prediction and perplexity : Because BERT is originally trained for masked word prediction, it can predict the word distribution given

context from both sides. Thus, BERT is preferable for generating p_{lm}. Estimating the perplexity for a sentence requires BERT to run in decoder mode and be fine-tuned. Perplexity can also be measured by other language models such as GPT2 [Radford et al., 2019]. We use BERT mainly for the consistent vocabulary with p_{lm}.

Word embeddings and USE for similarity. Word embeddings is more efficient as it only computes the sum of vectors and cosine similarity. In enforcing distribution, we need to replace the selected position with all possible z's and measure the similarity, so we use word embeddings for efficiency. In the critique score, only the proposal sentence needs to be measured, so we can afford more computation time of USE.

4.1.3 Experiments

We conducted experiments on a wide range of datasets and multiple victim classifiers to show the efficacy of R&R. We first evaluate the quality of adversarial examples using automatic metrics. Then, we conducted human evaluation to show the necessity to generate highly similar and fluent adversarial examples. Finally, we conduct an ablation study to analyze each component of our method, and discuss defense against the attack.

Experimental Setup

Datasets. We use 3 conventional text classification datasets: topic classification, sentiment classification, and question type classification. We also use 2 security-critical datasets: hate speech detection and fake news detection. Dataset details are given in Table 4.1.

Victim Classifiers. For each dataset, we use the full training set to train three victim classifiers: (1) BERT-base classifier [Devlin et al., 2019]; (2) RoBERTa-large classifier [Liu et al., 2019], and (3) FastText classifier [Joulin et al., 2017a].

Baselines. We compare our method against two strong baselines: TextFooler [Jin et al., 2020a] and CLARE [Li et al., 2021a].

Name	#C	Len	Description
AG	4	43	News topic classification by Gulli.
MR	2	32	Movie review dataset by Pang and Lee [2005].
TREC	6	8	Question type classification by Li and Roth [2002].
HATE	2	23	Hate speech detection dataset by Kurita et al. [2020].
FNS	2	30	Fake news detection dataset by Yang et al. [2017b]. We use the summary of each news (first sentence) for classification.

Table 4.1: Dataset details. #C means number of classes. Len is the average number of words in a sentence.

	AG	MR	TREC	HATE	FNS
BERT-base	92.8	88.2	97.8	94.0	81.2
RoBERTa-large	92.7	91.6	97.3	95.0	75.5
FastText	89.2	79.5	85.8	91.5	72.4
Log Perplexity	3.38	5.27	3.91	3.56	4.92

Table 4.2: Clean accuracy (CAcc) of BERT-base, RoBERTa-large and FastText classifiers and sentence log perplexity on the clean test set.

Hyperparameters. In R&R, we use the BERT-based language model for p_{lm}. For each dataset, we fine-tune the BERT language model using 5k batches on the training set[1] with batch size 32 and learning rate 0.0001, so it is adapted to the dataset. We set the enforcing distribution hyper-parameters $w_{\text{enforce}} = 5$. The decision function hyper-parameters $w_{\text{ppl}} = 5$, $w_{\text{sim}} = 20$, $\phi_{\text{sim}} = 0.95$, $w_{\text{clf}} = 2$. To generate each paraphrase, we set maximum rewrite iterations at 200, and replace a 3-word span in each iteration. We implement R&R in a 50-sentence batch and apply early-stop when half of the batch is misclassified. We apply the rollback operation every 10 steps of rewrite. Then, we return the adversarial example with the best critique score.

Hardware and Efficiency. We conduct experiments on Nvidia RTX Titan GPUs. One attack on a BERT-base classifier using R&R takes 15.8 seconds on average. CLARE takes 14.4 seconds on average. TextFooler is the most efficient algorithm and takes 0.45 seconds.

[1]We use the plain text to fine-tune the language model, and do not use the label. In the threat model, we assume the attacker can access plain text data from a similar domain.

Automatic Metrics. We evaluate the efficacy of the attack method using 3 automatic metrics:

- *Similarity* (↑): We use Universal Sentence Encoder to encode the original and adversarial sentences, then use the cosine distance of two vectors to measure the similarity. We set a similarity threshold at 0.95, so the similarity of a legitimate adversarial example should be greater than 0.95.
- *Log Perplexity* (↓) shows the fluency of adversarial sentences.
- *ASR* (↑) shows the ratio of correctly classified text that can be successfully attacked.

Human Metrics. Automatic metrics are not always reliable. We use Mechanical Turk to verify the similarity, fluency, and whether the label of the text is preserved with respect to human evaluation.

Table 4.3 shows some examples. We find R&R makes natural modifications to a sentence and preserves the semantic meaning.

Automatic Evaluation Results

Table 4.4 shows the ASR, similarity and perplexity metrics on the three victim classifiers; and Figure 4-4 illustrates the ASR.

Since we already apply a rigorous 0.95 threshold on similarity to ensure the adversarial examples are similar to the original sentences, the similarity metrics do not show significant differences. R&R achieves the best ASR on all datasets and across all classifiers. The average improvement compared with the CLARE baseline is +16.2%, +12.8%, +14.0% on BERT-base, RoBERTa-large and FastText classifiers respectively. This means that with the same similarity threshold, R&R is capable of finding more adversarial examples, i.e. for some text, R&R can find adversarial examples with a similarity higher than 0.95 while baseline methods cannot. This shows R&R can find adversarial examples with high similarity.

On AG, MR, TREC and FNS datasets, R&R outperforms baseline methods in 9 cases. Baseline methods outperform R&R by a tiny margin on the other 3 cases.

Original (prediction: Technology): GERMANTOWN , Md . A Maryland - based *private lab* that analyzes criminal - case DNA evidence has fired an analyst for allegedly falsifying test data . **Adversarial (prediction: Business):** GERMANTOWN , Md . A Maryland - based *bio testing company* that analyzes criminal - case DNA evidence has fired an analyst for allegedly falsifying test data .
Original (prediction: Sport): LeBron James scored 25 points , Jeff McInnis added a season - high 24 and the Cleveland Cavaliers won their sixth straight , 100 - 84 over the Charlotte Bobcats on Saturday night . **Adversarial (prediction: World):** LeBron James scored 25 points , Jeff McInnis added a season - high 24 and the Cleveland Cavaliers won their sixth straight , 100 - 84 *Saturday* over the *visiting* Charlotte Bobcats on Saturday night ..
Original (prediction: Negative): don ' t be fooled by *the* impressive cast list - eye see you is pure junk . **Adversarial (prediction: Positive):** don ' t be fooled by *this* impressive cast list - eye see you is pure junk .
Original (prediction: Ask for description): What is die - casting ? **Adversarial (prediction: Ask for entity):** What is *the technique of* die - casting ?
Original (prediction: Toxic) go back under *your* rock u irrelevant party puppet **Adversarial (prediction: Harmless)** go back under *the* rock u irrelevant party puppet

Table 4.3: A few adversarial examples generated by R&R with the perturbation highlighted in red.

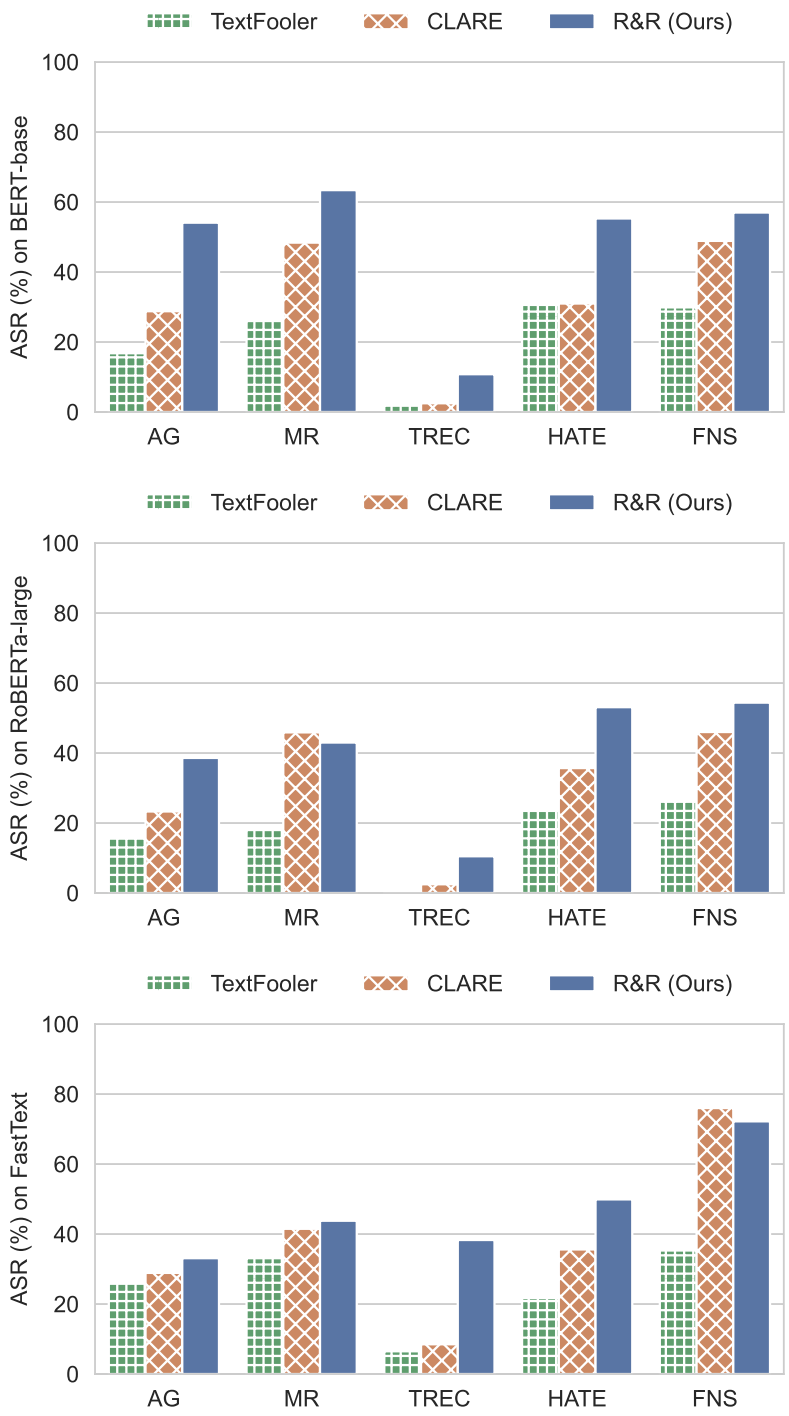

Figure 4-3: ASR of R&R and baselines on BERT-base (top), RoBERTa-large (middle), and FastText (bottom) classifiers.

		AG			MR			TREC			HATE			FNS		
	Attack	**ASR**	**Sim**	**PPL**	**ASR**	**Sim**	**PPL**	**ASR**	**Sim**	**PPL**	**ASR**	**Sim**	**PPL**	**ASR**	**Sim**	**PPL**
BERT	TextFooler	16.8	**0.98**	4.00	26.0	0.97	5.92	1.8	**0.97**	5.30	30.6	0.97	**3.53**	29.9	**0.98**	5.44
	CLARE	28.8	0.97	**3.60**	48.4	0.97	5.70	2.5	0.96	5.58	31.0	0.97	3.99	48.9	**0.98**	**5.02**
	R&R (Ours)	**54.1**	**0.98**	3.64	**63.4**	**0.98**	**5.36**	**10.8**	**0.97**	**5.29**	**55.3**	**0.98**	4.06	**57.0**	**0.98**	5.05
RoBERTa	TextFooler	15.6	**0.98**	5.21	18.0	**0.97**	6.06	0.4	0.96	7.09	24.0	**0.98**	4.20	26.6	**0.98**	5.45
	CLARE	23.3	0.97	5.24	45.9	**0.97**	5.67	2.5	**0.97**	6.53	35.7	0.97	4.37	46.0	**0.98**	**5.20**
	R&R (Ours)	**41.2**	**0.98**	**3.73**	**48.5**	**0.97**	**5.53**	**12.5**	**0.97**	**5.17**	**55.7**	0.97	**4.07**	**59.6**	**0.98**	5.25
FastText	TextFooler	25.8	**0.98**	4.16	33.1	**0.98**	5.85	6.5	**0.98**	5.04	21.7	**0.98**	**3.44**	35.3	**0.98**	5.46
	CLARE	28.9	0.97	3.91	41.5	0.97	5.79	8.5	0.97	6.06	35.6	0.97	4.24	76.0	**0.98**	5.15
	R&R (Ours)	**37.8**	**0.98**	**3.84**	**48.9**	**0.98**	**5.48**	**44.1**	**0.98**	**4.68**	**53.3**	**0.98**	4.03	**76.4**	**0.98**	**5.10**

Table 4.4: Automatic evaluation results. "Sim" and "PPL" represent similarity measured by USE and the log perplexity measured by BERT respectively.

	AG			MR			TREC			HATE			FNS	
	S.	**F.**	**M.**	**S.**	**F.**	**M.**	**S.**	**F.**	**M.**	**S.**	**F.**	**M.**	**S.**	**F.**
TextFooler	3.93	3.58	0.90	3.3	3.49	**0.92**	3.25	2.88	0.88	3.76	3.61	0.46	3.58	3.58
CLARE	3.75	3.65	0.93	2.44	3.33	0.74	3.00	3.00	0.75	**3.89**	**4.41**	**0.81**	3.67	3.65
R&R (Ours)	**4.12**	**3.87**	**0.99**	**3.48**	**3.61**	0.85	**3.59**	**3.14**	**0.89**	3.59	3.94	0.76	**3.81**	**3.87**

Table 4.5: Human evaluation results. "S.", "F." and "M." represents the similarity, fluency and label match annotated by human.

This shows R&R keeps sentence fluency as high as baseline methods do, and does not sacrifice sentence fluency for higher ASR. The only failure case is on the HATE dataset, where Textfooler outperforms R&R in perplexity. Thus we investigate the average log perplexity of corresponding original sentences for each method. We find it is 3.24 for TextFooler and 3.94 for R&R. So TextFooler achieves low perplexity because it succeeds on original sentences with low perplexity while failing on those with higher perplexity.

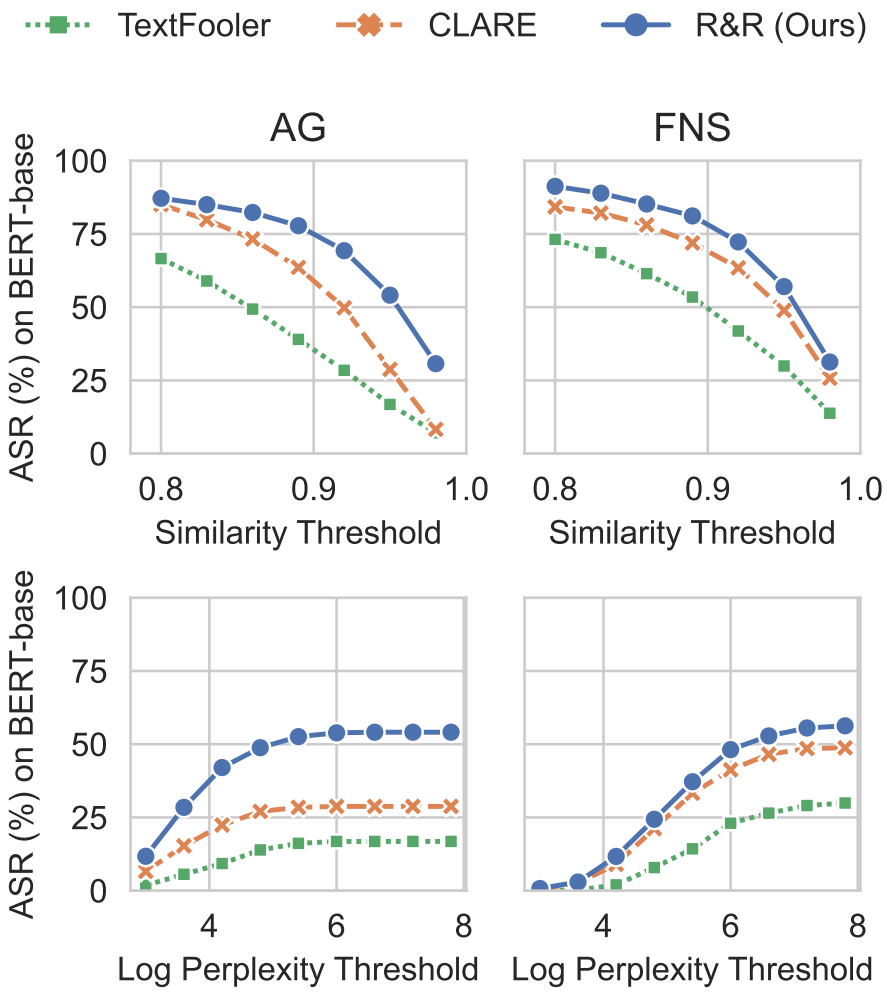

Figure 4-4: ASR with respect to different similarity and perplexity constraints on BERT-base classifiers. When evaluating different similarity thresholds, we do not set thresholds on perplexity. When evaluating perplexity thresholds, we fix the similarity threshold to 0.95. See Figure C-4, C-5, C-6 in Appendix for other datasets and classifiers.

We further measure ASR with various similarity and perplexity thresholds. On Figure 4-4, we set different thresholds and show the corresponding ASR. We observe that the curves of R&R are above the baseline curves in most cases, showing that R&R outperforms baselines on most threshold settings. It means R&R can achieve a higher ASR with the same similarity or perplexity threshold.

Human Evaluation Results

We use Mechanical Turk to evaluate the following metrics.

- *Sentence similarity* (↑): Turkers are shown pairs of original and adversarial sentences, and are asked whether the two sentences have the same semantic meaning. They annotate the sentence in a 5-likert, where 1 means strongly disagree, 2 means disagree, 3 means not sure, 4 means agree, and 5 means strongly agree.

- *Sentence fluency* (↑): Turkers are shown a random shuffle of adversarial sentences, and are asked to rate the fluency in a 5-likert, where 1 describes a bad sentence, 3 describes a meaningful sentence with a few grammar errors, and 5 describes a perfect sentence.

- *Label match* (↑): Turkers are shown a random shuffle of adversarial sentences and are asked whether they belong to the class of the original sentence. They are asked to rate 0 as disagree, 0.5 as not sure, and 1 as agree.

We sample 100 adversarial sentences from each method, and each task is annotated by 2 Turkers. We do not annotate label matches on the FNS dataset because identifying fake news is too challenging for Turkers. We require the location of the Turkers to be in United States, and their Hit Approval Rate to be greater than 95%. The screenshots of the annotation tasks are shown in Figures C-1, C-2, and C-3 in the Appendix.

Table 4.5 shows the human evaluation results. R&R outperforms baselines on similarity and fluency on 4 datasets. This shows that by optimizing the critique

score, R&R improves the similarity and fluency of adversarial sentences. Our method fails on the HATE dataset despite good automatic metrics. We hypothesize that this dataset collected from Twitter is more noisy than the others, causing the malfunction of automatic similarity and fluency metrics.

Ablation Study

We conduct an ablation study on AG and FAKE datasets to understand the contribution of stochastic decision function, periodic rollback and multiple-word masking.

Ablation study on decision function. In the Rewrite stage, we use a stochastic decision function based on the critique score. One alternative can be a deterministic greedy decision function, which accepts a rewrite only if the rewrite increases the critique score. Figure 4-5 shows the ASR with respect to different similarity thresholds. We find that the stochastic decision function outperforms the greedy one. We interpret this phenomenon as the greedy decision function getting stuck in local maxima, while the stochastic one overcomes this issue by accepting a slightly worse rewrite.

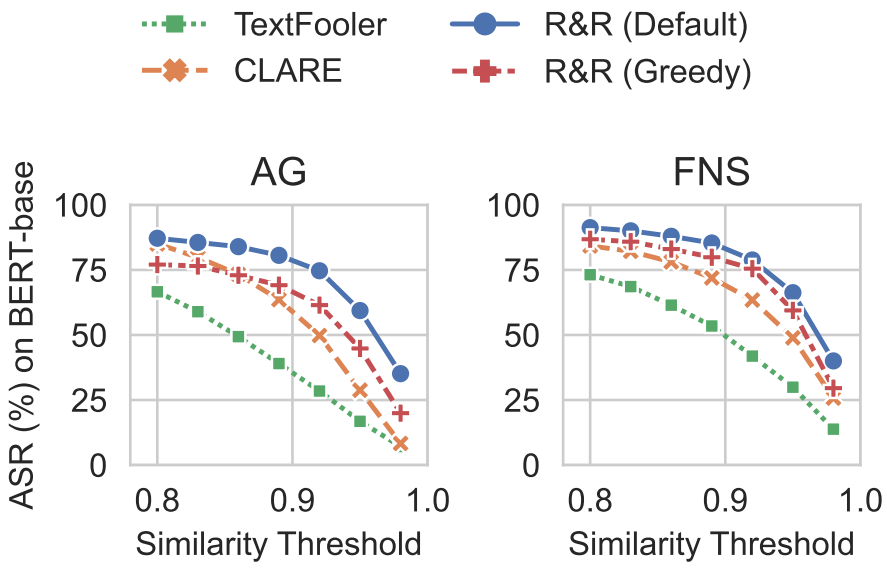

Figure 4-5: The ASR of R&R using different decision settings. "Greedy" means using a greedy decision function, which accepts a rewrite only if it has a higher critique score.

Ablation study on rollback We apply rollback periodically during the attack. We compare this with two alternatives: (1) no rollback (NRB) which only uses rewrite to construct the adversarial sentences, and (2) single rollback (SRB) which applies rollback once on the NRB results. Figure 4-6 shows the result. We find that rollback has a significant impact. NRB performs the worst. Without rollback, it is difficult to get high cosine similarity when many words in the sentence have been changed. Single rollback increases the number of overlapped words, which usually increases the similarity measurement. By periodically applying the rollback, the rollbacked sentence can be further rewritten to improve the similarity and fluency metrics, thus yielding to the best performance.

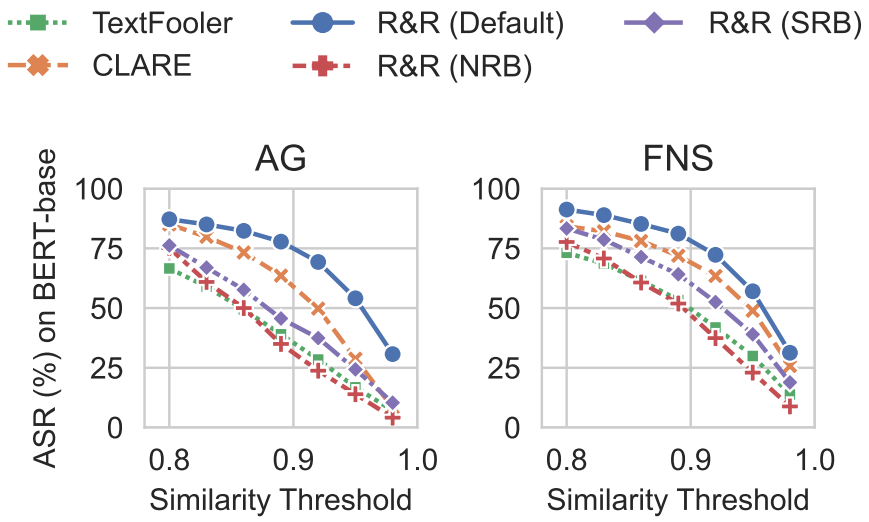

Figure 4-6: The ASR of R&R using different rollback settings. "NRB" means no rollback operation and "SRB" means single rollback.

Ablation study on multiple-word masking In the Rewrite stage, we mask a span of multiple words in each iteration. Intuitively, when using a smaller span size, the masked words are easier to predict. The proposal distribution will assign high probability to the original words at masked positions. Therefore, the candidate sentences are likely to be identical to the original sentence, thus limiting the number of perturbations explored. When the span is large, predicting words becomes more difficult, and we can sample different candidate sentences. But it is more likely to

construct dissimilar or influential sentences. We vary the span size from 1, 2, 3, to 4 and show the results on Figure 4-7. We find that using span size 3 yields the best performance over most similarity thresholds.

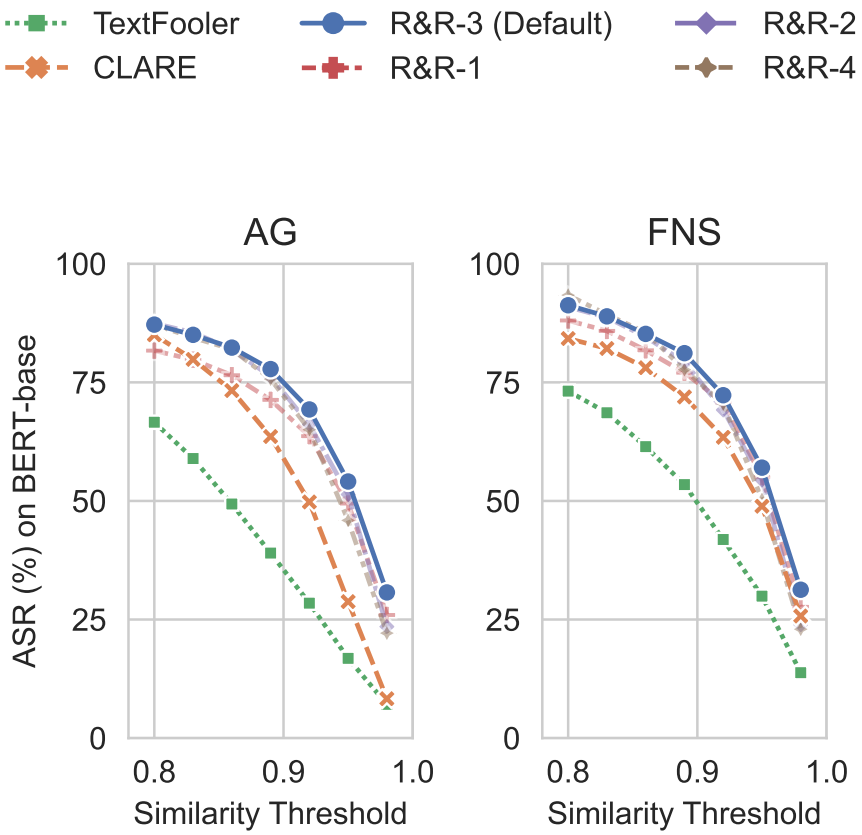

Figure 4-7: The ASR of R&R using different masking span sizes. R&R-1 to R&R-4 represent span sizes of 1 to 4 respectively. We use span size 3 by default.

Defending Against Attacks

We further explore whether existing defense mechanisms can protect the classifier from R&R attacks. We try two defenses.

- Adversarial attack methods sometimes introduce outlier words to trigger misclassification. Therefore we follow Qi et al. [2021a] and apply a perplexity-based filtering to eliminate outlier words in sentences. We generate adversarial sentences on a vanilla classifier, then apply the filtering.

- SHIELD [Le et al., 2022] is a recently proposed algorithm that modifies the last layer of a neural network to defend against adversarial attack. We apply this method to classifiers and attack the robust classifier.

	AG		**FNS**	
	+Filter	**+SHIELD**	**+Filter**	**+SHIELD**
TextFooler	6.2	8.2	13.8	16.7
CLARE	5.6	18.2	19.0	51.1
R&R (ours)	**22.3**	**30.6**	**23.1**	**59.4**

Table 4.6: The ASR of attack methods when applying the perplexity-based filtering (Filter) and the SHIELD defense on BERT-base classifiers.

Table 4.6 shows the ASR of attack methods with a defense applied. We show that existing defense methods cannot effectively defend against R&R. It still outperforms existing methods in ASR by large margin.

4.1.4 Summary

We formulate the textual adversarial attack as a multi-objective optimization problem. We use a critique score to synthesize the similarity, fluency, and misclassification objectives, and propose R&R that optimizes the critique score to generate high-quality adversarial examples. We conduct extensive experiments. Both automatic and human evaluation show that the proposed method succeeds in optimizing the automatic similarity and fluency metrics to generate adversarial examples of higher quality than previous methods.

4.2 Single-word Adversarial Perturbation Attack

4.2.1 Motivation

Over the past few years, various methods have been proposed to generate adversarial examples [Jin et al., 2020b, Li et al., 2021a]. State-of-the-art methods can achieve an ASR of over 80%, becoming a severe security issue. These methods usually apply *multiple word changes* to the original text. However, we show in Table 4.7 that the subset of adversarial examples where *only one word* is changed contributes greatly to the ASR. For example, 66% of adversarial examples generated by a CLARE attack on an MR dataset change only one word. Furthermore, many different sentences may share the same perturbation. For example, inserting the word "online" can cause several business news articles to be misclassified as technology news articles. Both these characteristics are beneficial to attackers. First, the fewer words that are changed, the more innocent the adversarial example will look and the more semantically similar it will be to the original. Second, reusing the same perturbation reduces the computation cost of generating multiple adversarial sentences. Therefore we propose SAP-Attack which changes only one word in a sentence with another high-capability word (i.e., high κ) in order to trigger misclassification. We show that this novel type of attack is more efficient than existing attack methods.

	TextFooler		**BAE**		**CLARE**		**R&R**		**SAP-Attack**	
	ASR	**SP%**	**ASR**	**SP%**	**ASR**	**SP%**	**ASR**	**SP%**	**ASR**	**SP%**
AG	65.2	17.0	19.3	35.8	84.4	38.6	87.2	20.0	82.7	100
MR	72.4	49.2	41.3	66.6	90.0	66.2	98.3	48.7	93.5	100

Table 4.7: The ASR and percentage of adversarial examples with single-word perturbation (denoted as SP%). We attack BERT-base classifiers on AG and MR, the datasets used in our experiments, using TextFooler [Jin et al., 2020b], BAE [Garg and Ramakrishnan, 2020], CLARE [Li et al., 2021a], R&R, and SAP-Attack.

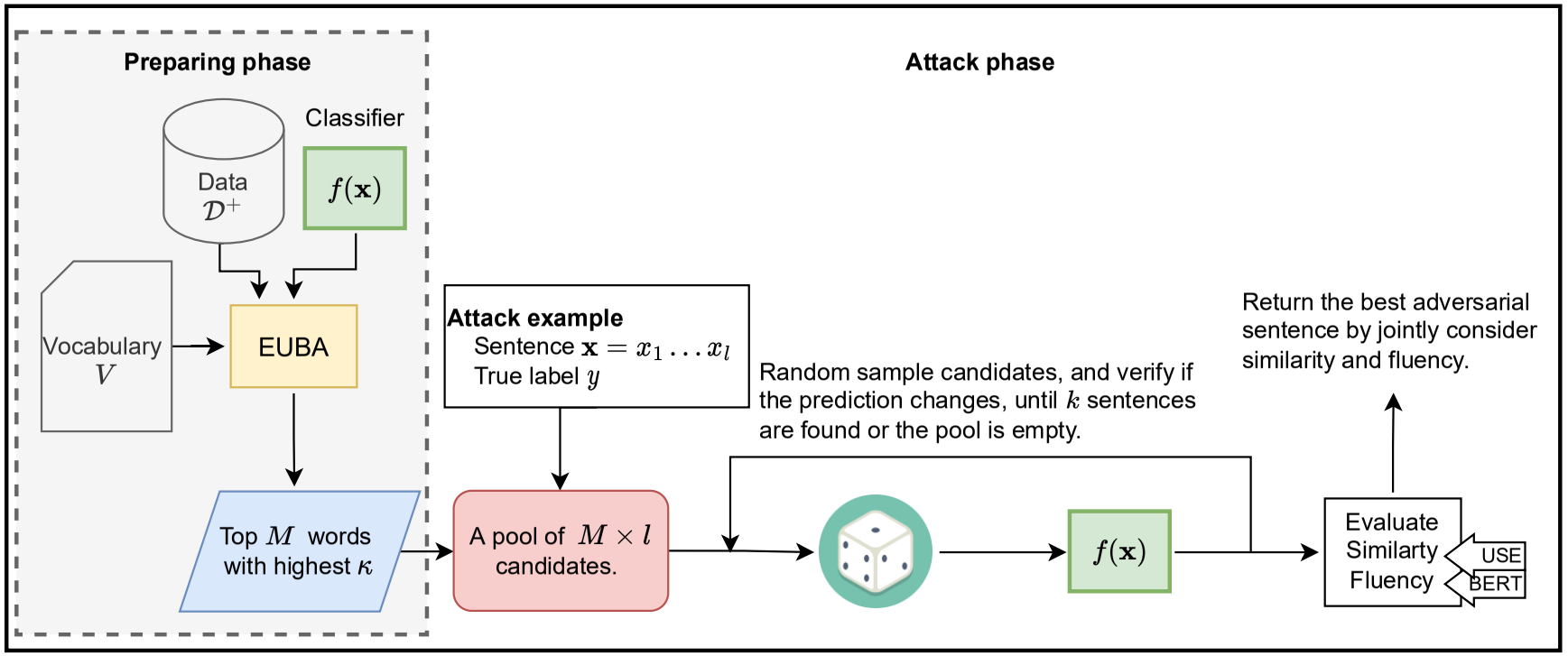

Figure 4-8: An overview of SAP-Attack.

4.2.2 Methodology

For attackers who try to find fluent and semantically similar adversarial examples, words with high κ provide the potential for low-cost attacks. We propose SAP-Attack, which first uses our algorithm, EUBA to estimate κ, then use the top M words with highest κ to craft an attack. The framework is shown on Figure 4-8. To conduct an attack on a sentence $\mathbf{x}$ of length l, we try to put these words at all possible positions in order to create a pool of $l \times M$ candidate sentences. Then, we draw samples from the pool and verify them on the classifier. We terminate when we either exhaust the pool or find k candidates which change the prediction. To ensure similarity and fluency, we employ Universal Sentence Encoder (USE) [Cer et al., 2018] and BERT language models. For each $\mathbf{x}'$ that can change the prediction, we compute a joint criteria score

$$\alpha \cos(H(\mathbf{x}'), H(\mathbf{x})) - \beta \frac{\text{ppl}(\mathbf{x}')}{\text{ppl}(\mathbf{x})}, \tag{4.7}$$

where $H(\cdot)$ is the USE sentence embedding, and $\text{ppl}(\cdot)$ is the perplexity of a sentence measured by BERT. We then pick the sentence with the highest score as output. We set $M = 50, k = 50, \alpha = 3, \beta = 20$ in our experiments.

There are two differences between SAP-Attack and conventional black-box adversarial attacks. (1) SAP-Attack substitutes only one word with one of the $M = 50$ words, whereas conventional methods can change multiple words and can use any word. As a result, SAP-Attack is much more efficient. (2) Estimating κ using EUBA requires computing the gradient of the classifier, while black-box adversarial attack methods do not require this. We can make SAP-Attack not rely on the gradient by estimating κ through brute force or heuristic methods.

4.2.3 Experiments

We conduct comprehensive experiments to show SAP-Attack is as effective as conventional adversarial attacks that change multiple words.

Experiment Setup

Datasets. We conduct our experiments on 4 datasets. Dataset details are shown in Table 4.8.

Name	#C	Len	Description
AG	4	43	News topic classification by Gulli.
MR	2	32	Movie review dataset by Pang and Lee [2005].
SST2	2	20	Binary Sentiment Treebank [Wang et al., 2019b].
HATE	2	23	Hate speech detection dataset by Kurita et al. [2020].

Table 4.8: Dataset details. #C means number of classes. Len is the average number of words in a sentence.

Classifiers. We evaluate our method on two classifiers, the BERT-base classifier [Devlin et al., 2019] and the distilBERT-base classifier [Sanh et al., 2019]. The classifiers are trained with the full training set for 20k batches with batch size 32 using the AdamW [Loshchilov and Hutter, 2019a] optimizer and learning rate 2e-5. We also include results for the RoBERTa-base classifier [Liu et al., 2019] in Figure C-7, Figure C-8 and Table B.2 in Appendix.

Metrics. We use several metrics to show the quality and robustness of a classifier.

- We use clean accuracy (**CAcc**↑), which is the accuracy of the classifier measured on the original testset.
- We use ρ to quantify the adversarial robustness in a single-word perturbation scenario. We sample 1000 examples from the training set to compute $\rho(\uparrow)$ since it is defined on the training set. We also measure it on the test set and denote it as $\rho^*(\uparrow)$. We only consider all-letter words and ignore punctuation marks and affixes, thus reducing the vocabulary from 30k to 20k words. Unless otherwise stated, we use our own proposed algorithm EUBA to estimate ρ and set $m = 512$ by default.
- We use the attack success rate (**ASR** ↓) to show how robust the classifier is against an adversarial attack method. A lower ASR means the classifier is more

robust. We consider an attack to be successful if the similarity measured by the cosine of USE embeddings is greater than 0.8. We sample 1000 sentences from the test set to conduct an attack.

- Since we consider single-word perturbations, we add a constraint on perturbing only 1 word, and reevaluate the ASR and denote it as **ASR1** (↓). Since baseline methods are not designed to only use word substitutions, ASR1 allows single-word substitution, deletion and insertion.

Note that ASR also shows the efficacy of attack methods. A higher ASR means the attack method is more effective. We use ASR(↑) and similarly ASR1(↑) in the context of comparing attacks.

Adversarial attack baselines. We compare SAP-Attack against four strong attack methods, namely: **TextFooler** [Jin et al., 2020b], **BAE** [Garg and Ramakrishnan, 2020], **CLARE** [Li et al., 2021a], and our R&R. We also use the ASR of these methods to show the robustness of the classifier against these baselines.

Necessity of κ and ρ metrics

Dataset	Top 95%	Top 0.1%
AG	6.5	38.7
MR	12.4	49.4

Table 4.9: The minimum κ of top 95% and 0.1% words for each dataset on BERT-base classifiers.

We measure the exact κ using brute force for the BERT-base classifier on AG and MR. The measurement takes 197 and 115 GPU hours respectively on a single Nvidia V100 GPU. Figure 4-9 shows the histogram of κ. Table 4.9 shows that 95% of words in the vocabulary can at least successfully attack 6.5% and 12.4% of examples, while the top 0.1% of words can change as much as 38.7% and 49.4% examples on the two datasets respectively. This shows that classifiers are extremely vulnerable to single-word perturbations. The distributions of κ for the two tasks are different, showing κ

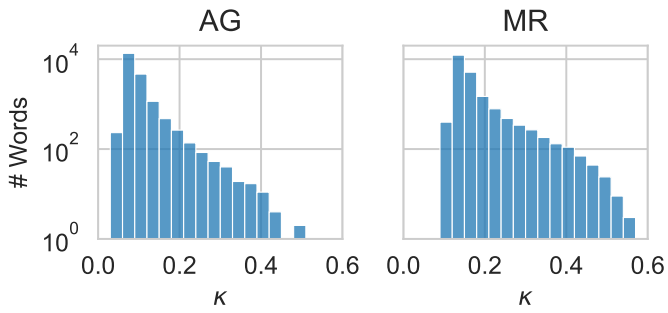

Figure 4-9: Histogram of words at different κ on BERT-base classifiers. Note that the y-axis is in log scale.

is task-dependent. Words have lower average κ on AG than on MR. We compute the $\rho(f)$ as 90.9 and 83.8 for the two datasets respectively. We analyze words with high κ in Appendix 4.2.3

We highlight that κ and ρ metrics are necessary. κ can show the adversarial capability of each individual word. It precisely reveals all vulnerabilities of the classifier under single-word attack. ρ is derived from κ and factors in the number of different single-word perturbations that can successfully attack each sentence, thus better quantifying the robustness in the single-word perturbation scenario compared to ASR which only shows whether each sentence could be successfully attacked.

Analyze high-κ words

Because of the high-κ on some words, we further ask how the accuracy of the classifier would change if it did not predict based on those words. We analyze the 10 words with the highest κ scores by measuring their effects on classifier accuracy. For each word, we sample two subsets of sentences from the training set, each with 1000 sentences.

- sentences with the word: We find sentences that contain this word, and measure the accuracy. We assume that removing the word does not change the label, and measure the accuracy after removing the word.
- sentences without the word: We find sentences that do not contain this word, and measure the accuracy. Then we attack these sentences using this word, and show the accuracy after the attack.

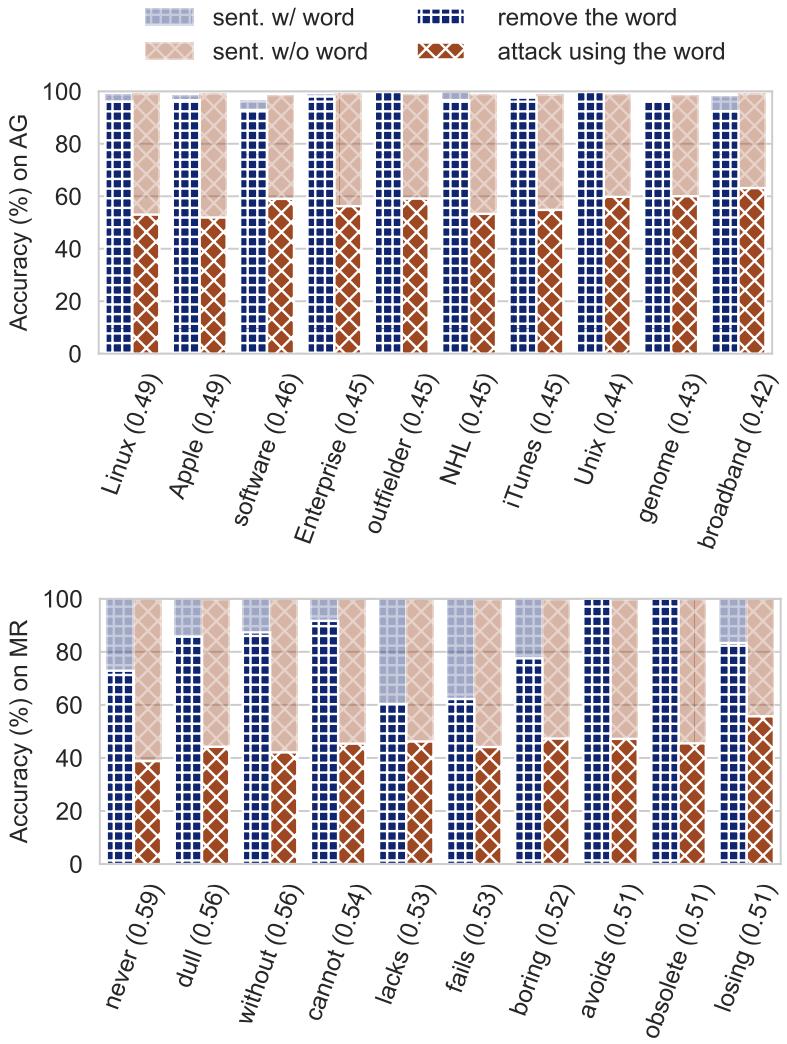

Figure 4-10: Top 10 words with highest κ and their effect on accuracy on BERT-base classifiers. The values in brackets are κ of words.

Figure 4-10 shows these top 10 words and the accuracies that result from these perturbations. It shows κ is task-dependent. For AG, 7 out of 10 words are related to technology. The κ for "Linux" is 49%, meaning that it can change the prediction of 49% sentences in the dataset. Although "Linux" is related to technology, it is not possible that the true label of sport or business news suddenly changes to technology only by changing one word to "Linux". It is more likely that the classifier learns a superficial correlation. On MR, the top 10 words are mostly negatives like "never." Regarding accuracies, the clean accuracy for both subsets is near 100% since they are

from the training set. Removing a word with a high κ does not affect the performance much on the AG dataset, but has a higher impact on the MR dataset. This is due to the nature of sentiment classification, where removing a negative can flip the sentiment of the sentence. Nevertheless, the accuracy is over 70% in 8 out of 10 words on MR, indicating that for many sentences in the dataset, even if negative words are removed, there is still enough context to classify sentiment. Adversarially substituting one word with one of the top 10 words has a severe impact on the accuracy for both datasets, showing that high κ words are paid excessive attention by the classifier and can easily cause misclassification.

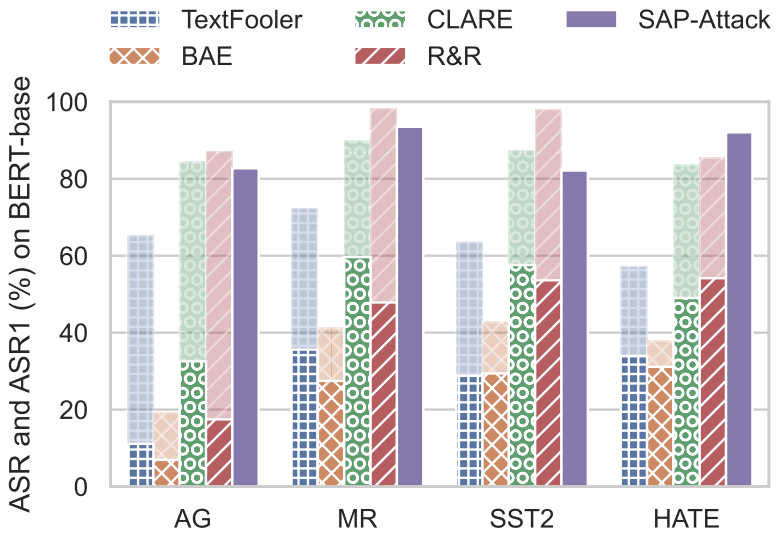

Figure 4-11: ASR and ASR1 on BERT-base classifiers. The translucent (taller) bars represent ASR, while the solid (shorter) bars represent ASR1. For SAP-Attack, ASR and ASR1 are the same.

SAP-Attack performance

Figure 4-11 and 4-11 shows the ASR and ASR1 for BERT-base and distilBERT-base classifiers. SAP-Attack achieves the best ASR1 compared with other methods. This shows that SAP-Attack is the most effective method for finding adversarial examples based on single-word perturbations. ASR1's are at least 73%, and over 80% in 7 out of 8 cases, which shows the significant vulnerability of classifiers. We also find

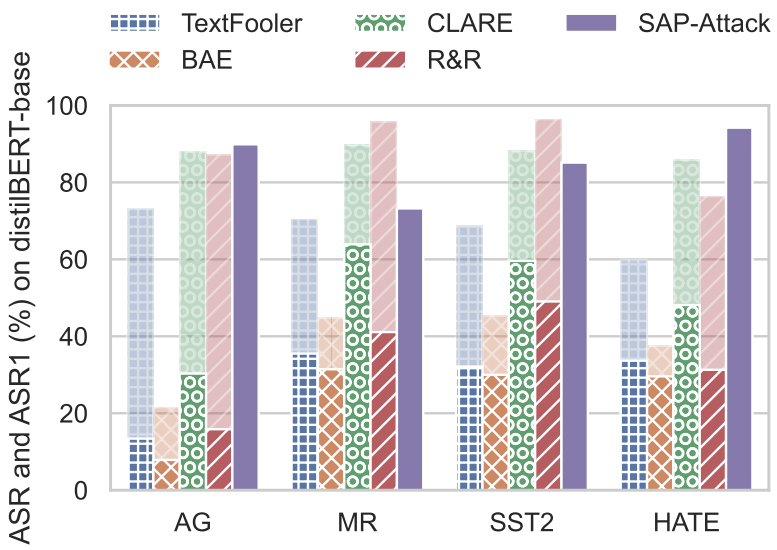

Figure 4-12: ASR and ASR1 on distilBERT-base classifiers. The translucent (taller) bars represent ASR, while the solid (shorter) bars represent ASR1. For SAP-Attack, ASR and ASR1 are the same.

SAP-Attack achieves comparable ASR compared with baseline attacks that modify multiple words.

Single-word perturbation may cause poor fluency and/or low semantic similarity. To show SAP-Attack is comparable with multi-word attacks in terms of fluency and similarity, we plot the ASR under different similarity and perplexity thresholds. We measure similarity using USE and perplexity using a BERT language model fine-tuned on the dataset. Figure 4-13 shows the result on AG and MR datasets. It shows that, at the same threshold, SAP-Attack achieves comparable or better ASR than those attack baselines.

Justifying Design Decisions of EUBA

We show the gap between the ρ upper bound and its true value when setting different early stop criteria. We set $m = \{128, 256, 512, 1024, 2048\}$. We also changed phase 1 in EUBA to form two alternative designs:

- EUBA (top1): We use the same gradient-based approximation. But for each word, we only verify the top-1 position, i.e. $\arg\min_i u_w^{(i)}$.

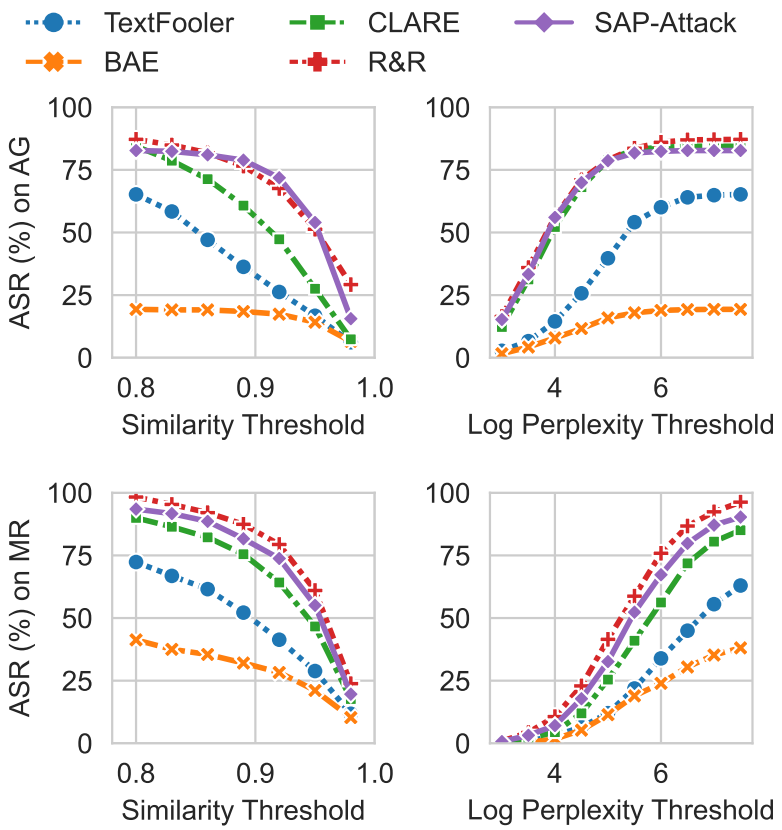

Figure 4-13: Comparing ASR under different similarity and perplexity thresholds on AG and MR datasets with BERT-base classifier.

- EUBA (w/o mask): We do not replace each position with masks. Instead, we use the original sentence to directly compute a first-order approximation, specifically

$$v_w^{(i)} = \langle \nabla_{\mathbf{e}_{x_i}} \log f(y|\mathbf{x}), \mathbf{e}_w - \mathbf{e}_{x_i} \rangle + \log f(y|\mathbf{x}). \tag{4.8}$$

Figure 4-14 shows the estimated ρ with respect to average time consumption for each example in D'_{train}. With the same time budget, EUBA outperforms alternative methods given the same time budget, and is very close to the true ρ. Our method EUBA is properly designed and configured to get a tighter bound compared to alternative designs.

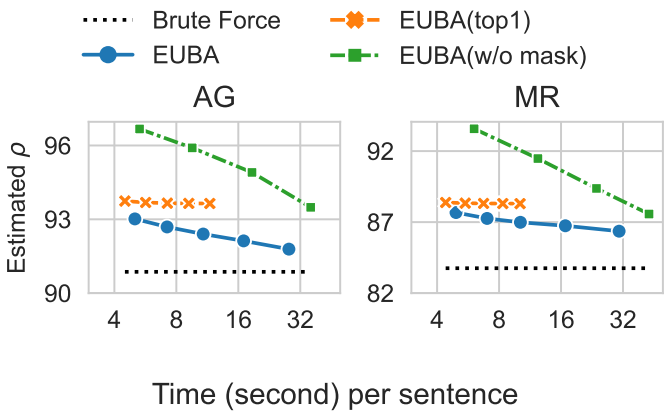

Figure 4-14: Comparing the estimated ρ with respect to time on BERT-base classifiers. For each method, the early stop criteria is 128, 256, 512, 1024, and 2048 (from left to right on each curve). EUBA (w/o mask) 2048 is not shown in the figure because of long time consumption. The Brute Force method takes 720s and 415s per sentence on AG and MR respectively.

4.2.4 Summary

We comprehensively analyze a restricted adversarial attack setup with single-word perturbation. We define two useful metrics, ρ that quantifies a text classifier's robustness against single-word perturbation attacks and κ, the adversarial capability of a word and show that these metrics are useful in measuring classifier robustness.

Furthermore, we propose SAP-Attack, a single-word perturbation attack which achieves a comparable or better attack success rate than existing and more complicated attack methods on classifiers, and significantly reduces the effort necessary to construct adversarial examples.

Chapter 5

Defending against Adversarial Attack

Now that we have revealed attack methods, we make two attempts at improving the robustness of classifiers. First, we design a novel defense method (**SAP-Defense**), which leverages the first-order approximation to find adversarial single-word perturbations and augment the training set. We retrain the classifier to become more robust not only against single-word perturbation attacks, but also attacks involving multiple word changes.

Second, we propose **LMAg** (Language-Model based Augmentation using Gradient Guidance), a in situ data augmentation method, as an effective defense mechanism. Specifically, LMAg uses the norm of the gradient to estimate the importance of a word to the classifier's prediction, and then substitutes those words with alternatives proposed by a masked language model. LMAg is an additional protection layer on the classifier, and thus does not require additional training. Experimental results show that LMAg can improve the after-attack accuracy of a BERT text classifier by 51.5% and 17.3% for two setups respectively.

Chapter outline:

- We introduce our SAP-Defense and present our experimental results in Section 5.1.
- We demonstrate the in-situ data augmentation method–LMAg–in Section 5.2.

5.1 Single-word Perturbation Defense

5.1.1 Motivation

Adversarial training and data augmentation are widely adapted to improve the classifier robustness against certain adversarial attacks. However, the inefficiency of attack methods is a barrier for these methods. We leverage the first-order approximation to find adversarial single-word perturbations and augment the training set, making it more efficient than existing methods.

5.1.2 Methodology

We present a data augmentation strategy, SAP-Defense, to improve the robustness of the classifier in a single-word perturbation scenario. Specifically, we design three augmentations.

Random augmentation randomly picks one word in a sentence, then replaces it with another random word in V. Since 95% words have at least 6.5% κ according to our experiment results, even purely random augmentation can sometimes generate adversarial examples.

Gradient-based augmentation uses gradient information to find a single-word substitution that is more likely to cause misclassification. We approximate the log probability of a correct prediction after substituting x_i with w as

$$v_w^{(i)} = \langle \nabla_{\mathbf{e}_{x_i}} \log f(y|\mathbf{x}), \mathbf{e}_w - \mathbf{e}_{x_i} \rangle + \log f(y|\mathbf{x}). \tag{5.1}$$

Then we apply the substitution with the minimum $v_w^{(i)}$. Eq. (5.1) is more efficient because it only needs one forward and backward pass on $f(y|\mathbf{x})$ to compute $\nabla_{\mathbf{e}_{x_i}} \log f(y|\mathbf{x})$, compared to l forward and backward passes on $\nabla_{\texttt{<mask>}} \log f(y|\mathbf{s}_i)$ in Eq. (3.17). Therefore, it is more suitable for data augmentation which usually involves large-scale training data.

Some attack methods use a vocabulary different from V. They can substitute x_i with a word $t \notin V$. Both gradient-based and random augmentation cannot defend

against adversarial examples caused by t. **Special word augmentation** is designed to address this issue. For each class y, we find a set of words that occurs much more frequently in other classes y', formally

$$T(y) = \{t | \max_{y' \neq y} \log \text{freq}_D(t, y') - \log \text{freq}_D(t, y) > 1\}, \tag{5.2}$$

where $\text{freq}_D(t, y)$ is the frequency of the word t in all the training examples with the label y. To augment an example $(\mathbf{x}, y)$, we randomly sample a position in $\mathbf{x}$ and replace it with a random word $t \in T(y)$.

In each training iteration, we apply gradient-based augmentation on half of the batch. For the other half, we randomly choose from original training data, random augmentation, or special word augmentation.

5.1.3 Experiments

Experiment Setup

We compare SAP-Defense with several baselines.

- **Rand**: We conduct random perturbation to augment training data in all steps of training.
- **A2T** [Yoo and Qi, 2021] is an efficient gradient-based adversarial attack method designed for adversarial training.

For Rand and SAP-Defense, we tune the classifier with another 20k batches. For A2T, we tune the classifier with 3 additional epoches as recommended by the authors.

SAP-Defense Performance

We show SAP-Defense can improve any classifier robustness in both single-word and multiple-word perturbation setups. We measure the accuracy and robustness of the vanilla and improved classifiers. The complete results are available on Table B.3 and Table B.4 in Appendix.

		AG				MR				SST2				HATE			
	Defense	**CAcc**	ρ	ρ^*	**ASR**	**CAcc**	ρ	ρ^*	**ASR**	**CAcc**	ρ	ρ^*	**ASR**	**CAcc**	ρ	ρ^*	**ASR**
BERT	NA	93.7	91.5	90.1	82.7	87.7	86.4	77.2	93.5	**80.6**	74.7	76.6	82.1	**94.5**	72.6	71.4	92.1
	Rand	92.0	96.3	93.1	66.2	87.5	98.2	80.7	84.2	79.8	82.1	79.7	77.9	94.3	86.3	83.2	89.2
	A2T	93.4	96.2	92.9	62.7	**88.3**	90.4	76.2	69.3	80.4	76.5	74.9	69.5	93.4	87.1	82.1	87.3
	SAP-D	**94.3**	**97.4**	**94.1**	**33.7**	87.5	**99.8**	**83.1**	**61.8**	79.3	**90.0**	**80.5**	**68.8**	93.7	**96.6**	**92.5**	**64.4**
distilBERT	NA	94.0	91.7	91.6	89.9	85.9	85.6	72.7	**73.2**	**78.7**	75.8	74.3	85.1	**93.3**	73.8	72.9	94.2
	Rand	94.2	95.4	92.0	69.5	**86.0**	96.5	78.1	92.2	78.0	81.1	78.3	87.2	93.1	85.6	82.6	91.3
	A2T	94.0	95.3	92.6	64.8	85.0	85.1	73.4	88.8	78.3	77.7	76.1	80.5	93.2	86.1	82.0	89.5
	SAP-D	**94.3**	**97.6**	**93.6**	**43.1**	85.1	**99.8**	**79.6**	73.4	78.2	**88.3**	**80.2**	**73.0**	92.0	**96.9**	**91.6**	**68.4**

Table 5.1: CAcc, ρ and ASR of SAP-Attack on original and robustified classifiers. NA denotes the vanilla classifier. SAP-Defense is abbreviated as SAP-D.

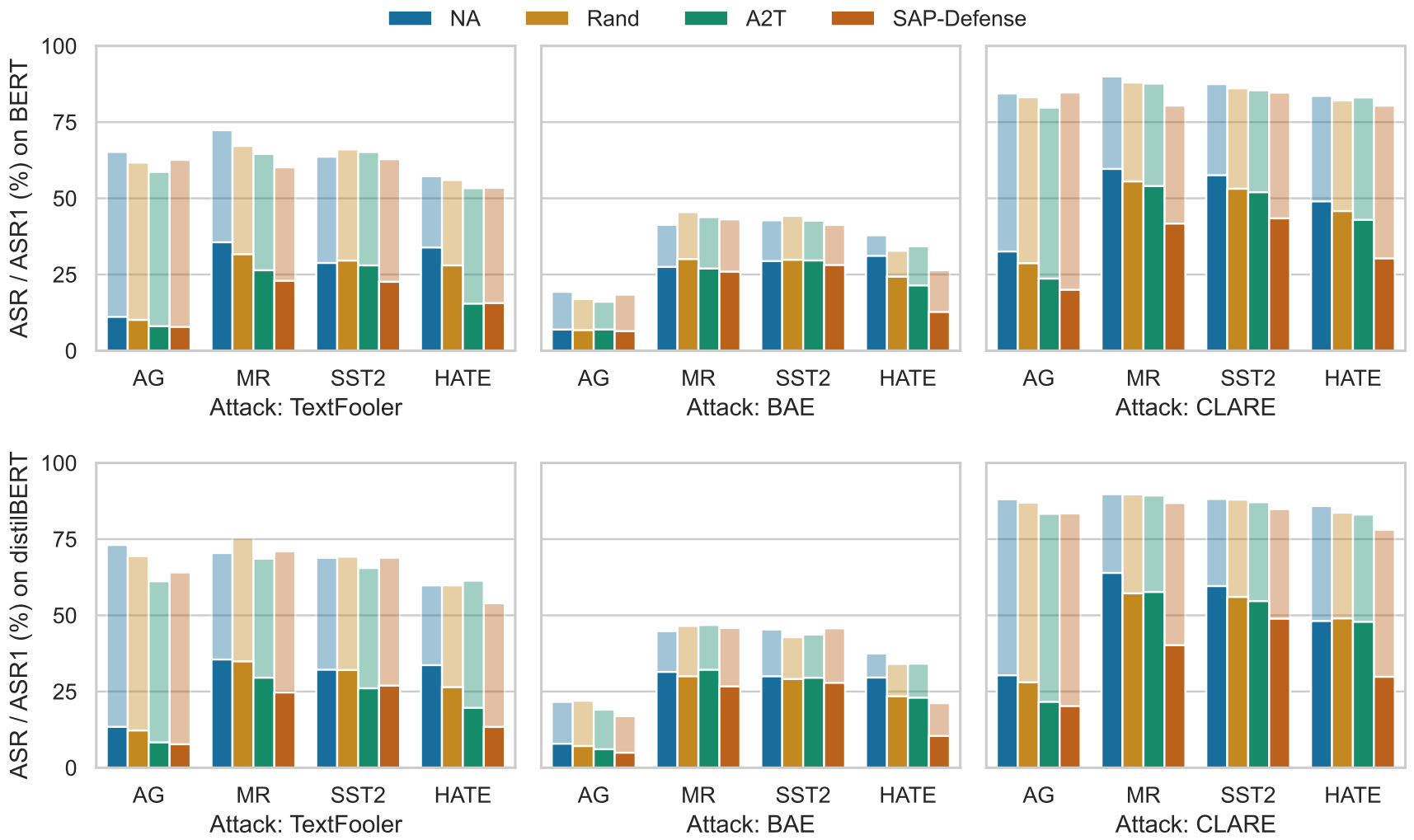

Figure 5-1: ASR and ASR1 of TextFooler, BAE and CLARE on robustified BERT-base (top) and distilBERT-base (bottom) classifiers. NA denotes the vanilla classifier.

Table 5.1 shows CAcc, ρ, ρ^*, and ASR of SAP-Attack. We observe that all data augmentation or adversarial training methods have small impacts on CAcc; in most cases the decrease is less than 1%. However, ρ and ρ^* differ a lot, showing that classifiers with similar accuracy can be very different in terms of robustness to single-word attacks (in both training and testing). We found that SAP-Defense outperforms Rand and A2T on ρ^* in all cases. This shows that SAP-Defense can effectively improve a classifier's adversarial robustness in a single-word perturbation scenario. Averaged over 4 datasets, SAP-Defense achieves 14.6% and 13.9% increase on ρ; 8.7% and 8.4% increase on ρ^*; and 30.4% and 21.2% decrease on ASR of SAP-Attack on BERT and distilBERT classifiers respectively.

Figure 5-1 shows the ASR and ASR1 on vanilla and improved classifiers. The ASR1 decreases by a large margin after the application of SAP-Defense, which is consistent with the improvement on ρ^*. The ASR also decreases in 20 out of 24 cases, showing the improvement of classifier robustness against a conventional multi-word adversarial attack setup.

Table 5.2 shows the training time for SAP-Defense and baseline methods. We find SAP-Defense is significantly more efficient than A2T.

	AG	**MR**	**SST2**	**HATE**
Rand	1.1	0.7	0.5	0.7
A2T	15.0	3.5	7.1	13.6
SAP-Defense	1.4	1.1	1.1	1.4

Table 5.2: Compare training time (hour) of defense methods on BERT-base classifiers.

5.1.4 Summary

We propose SAP-Defense to improve classifier robustness in a single-word perturbation scenario, and show that it also improves robustness against multi-word perturbation attacks.

5.2 In Situ Data Augmentation

5.2.1 Motivation

In the past few years, adversarial attack methods on text classifiers have been studied extensively. The goal of this type of attack is to rewrite a sentence such that a text classifier returns an incorrect prediction. Recently proposed attack methods can drastically decrease the accuracy of state-of-the-art classifiers: The adversarial sentences they generate are semantically similar to the original sentences and are of high grammatical quality, making them hard to differentiate from the original sentences as well as hard to detect.

As adversarial attacks can effectively degrade the accuracy of a text classifier, defending against such attacks has become necessary. The effort to defend against adversarial attacks on text classification mainly uses adversarial training. However, adversarial training of a text classifier is non-trivial for two reasons:

Efficiency requirement: A typical adversarial training process [Madry et al., 2018] augments each training minibatch with their adversarial counterpart, requiring the generation of adversarial examples efficiently. However, finding adversarial examples for a sentence is computationally expensive because it often involves heuristic search [Jin et al., 2020b, Zang et al., 2020a] or inference of a neural language model [Garg and Ramakrishnan, 2020, Li et al., 2020b]. It is impractical to generate adversarial examples in deployment. To address this issue, researchers generate adversarial examples in advance and use a fixed set of adversarial examples to tune the classifier and make it more robust [Jin et al., 2020b]. This solution reduces the efficacy of adversarial learning, because when the classifier is improved, new adversarial examples are needed to make it even more robust.

Efficacy requirement: As of yet, there is no demonstrated consensus on the efficacy of adversarial training [Morris et al., 2020a]: Some works (e.g., [Jin et al., 2020b, Ren et al., 2019]) have shown that adversarial training is effective, whereas others (e.g., [Alzantot et al., 2018]) have shown that it is not. Beyond the differences in the benchmark datasets, we will show that the efficacy of a defense method can

be measured under two different setups, making the results hard to compare (See Section 5.2.2).

In this paper, we propose a method to defend against adversarial attacks through in situ augmentation – transforming the input sentence at test time – rather than by tuning the classifier. Since most attack methods modify the sentence by substituting a small portion of words, counteracting these substitutions is one intuitive idea for defending against attacks. We can assume that words modified by the attack methods tend to have a high impact on the classifier's prediction, thus tending to increase the gradient norm. By substituting these words, we can counteract the modifications made by the attacker. As such, this paper proposes language-model-based augmentation with gradient guidance (LMAg). In LMAg, we compute the gradient of the classifier's prediction with respect to the input word embeddings. We then use the gradient norm as a weight to randomly mask words in the sentence, and employ a BERT [Devlin et al., 2019] language model to fill in masked words. Since LMAg is a data-augmentation method at test time, it does not require additional classifier training and is easier to deploy. Our experimental results show that the proposed method is effective in defending against various attacks.

5.2.2 Defense Formulation

In this section, we formulate the adversarial attack task and two defense setups.

Definition 3 (Efficacy of Adversarial Attack on Text Classification) *Given a sentence* $\mathbf{x} = \{x_1, \ldots, x_l\}$ *and its label* y*, a text classifier* $f(\cdot)$ *is supposed to make a prediction* $\hat{y} = f(\mathbf{x})$ *where* $\hat{y} = y$ *with high probability. When* $f(\mathbf{x}) = y$*, an adversarial attack method* $\mathcal{A}(\mathbf{x}, y, f)$ *generates an adversarial sentence* $\mathbf{u}$ *where* $\mathbf{u}$ *is grammatically correct and has the same semantic meaning as* $\mathbf{x}$*, but* $f(\mathbf{u}) \neq y$*. The efficacy of adversarial attack is measured by after-attack accuracy on the test set* $\mathcal{D}$ *such that:*

$$p_{(\mathbf{x},y)\sim\mathcal{D}}[f(\mathcal{A}(\mathbf{x}, y, f)) = y]. \tag{5.3}$$

As attack methods can successfully decrease the accuracy of a classifier, defending against these attacks is necessary. The goal of the defense is to make the classifier $f'(\cdot)$ more robust such that it retains high classification accuracy even when it is attacked with adversarial sentences. Note that there is no constraint on how $f'(\cdot)$ is constructed; it may be constructed either by tuning the classifier's parameters or by adding additional protections, such as adversarial sentence detection and/or text transformation.

Definition 4 (Efficacy of Original defense Against Adversarial Examples) *In this setup (Setup I), we generate adversarial examples by attacking the original classifier $f(\cdot)$, then we evaluate the robustness of the original classifier based on the absence of mistakes on these examples. In this setup, the after-attack accuracy on the test set $\mathcal{D}$ is defined as:*

$$p_{(\mathbf{x},y)\sim\mathcal{D}}[f'(\mathcal{A}(\mathbf{x}, y, f)) = y]. \tag{5.4}$$

Several works [Ren et al., 2019, Wang et al., 2021a] follow this setup and show significant improvement in after-attack accuracy.

Definition 5 (Efficacy of Boosted defense Against Adversarial Examples) *In this setup (Setup II), we generate adversarial examples by attacking the robustified classifier $f'(\cdot)$. In this setup, the after-attack accuracy is defined as:*

$$p_{(\mathbf{x},y)\sim\mathcal{D}}[f'(\mathcal{A}(\mathbf{x}, y, f')) = y]. \tag{5.5}$$

A few works [Jin et al., 2020b, Zang et al., 2020a] following this setup show relatively lower defense efficacy.

The difference between the two setups is whether the attacker is aware of the robustified classifier. We believe Setup II is prevailing in practice because: (1) Setup I underestimates the efficacy of attack methods. Most attack methods [Garg and Ramakrishnan, 2020, Jin et al., 2020b, Zang et al., 2020a] stop early when an adversarial sentence is found, but this early stop only indicates that the algorithm has found an adversarial example against the original classifier. This adversarial sample

may fail on the more robust classifier. But if the attack method directly attacks the boosted classifier and runs sufficient iterations, it may still find efficient adversarial examples; (2) Setup II is more realistic. When a more robust classifier is deployed, users interact with this classifier rather than the original one. Thus, it is more likely that an attacker directly attacks the strengthened classifier.

5.2.3 Methodology

In this section, we introduce LMAg, a in situ data augmentation for defending against adversarial attacks. LMAg consists of three steps: (1) Estimate the importance of words using the gradient of the classifier; (2) Generate multiple rephrases by stochastically masking important words in the input sentence and filling in with alternative words using a masked language model; and (3) Make a prediction based on the majority of predictions on the rephrases. Algorithm 3 shows the process of generating the rephrases.

Estimate Word Importance using Gradients

Gradient information has been widely used in attack methods. In white-box settings where attackers have full access to the classifier, a gradient is directly used to pick candidate substitutions [Liang et al., 2017], whereas in black box settings, the gradient is approximated by comparing the classifier's output with or without a word [Li et al., 2020b]. When building a defense, we assume that we have full access to the classifier; thus we directly compute gradients to identify important words that contribute the most to classification. We split a text classifier into two components:

$$f(\mathbf{x}) = \arg\max_k g(E(\mathbf{x}))_k,$$

where $E(\mathbf{x}) = \mathbf{e}_1, \ldots, \mathbf{e}_l$ is the input embedding layer that converts the words x_i into embeddings $\mathbf{e}_i$, and $g(\cdot)$ are the upper layers that made prediction from word embeddings. The output of $g(\cdot)$ is a probability distribution over all classes. We use $g()_k$ to denote the probability of k-th class. We compute the importance weight of

each word by

$$w_i = ||\nabla_{\mathbf{e}_i} \log \max_k g(E(\mathbf{x}))_k||_2.$$

For transformer-based models, $\mathbf{e}_i$ denotes the sum of word embedding, position embedding and token type embedding.

Stochastic Multiple Rephrasing

After calculating the importance weight for each word, we have to replace the most important words, hoping to counteract the adversarial attack. However, if we threshold the importance weight, then mask and substitute words, it is possible to mask all important words and make the sentence that is generated by the language model semantically different from the original sentence. For example, in sentiment analysis, if we mask all the adjectives that express sentiment, then the language model may generate a sentence with the opposite sentiment. To overcome this problem, we use a stochastic substitute method.

We randomly sample $m = \lfloor l \times \gamma \rfloor$ positions in the sentence using w_i as weights, where γ is the masking ratio. Specifically we sample positions:

$$t_1, \ldots, t_m \sim \text{Cat}(w_1^\alpha, \ldots, w_l^\alpha),$$

where Cat means a multinomial distribution, and α is a hyperparameter. Then we replace these positions with a special `MASK` token and use BERT language model to impute the most likely sentence as

$$\hat{\mathbf{x}} = [\arg\max \text{BERT}(\mathbf{x})_i]_{i=1\ldots l},$$

where $\text{BERT}(\mathbf{x})$ is a BERT language model. Note that all l words in the rephrase $\hat{\mathbf{x}}$ are proposed by the BERT language model, although only mask m words are masked. $\hat{\mathbf{x}}$ may have more than m word substitutions.

Different mask positions result in different rephrases. To make the classifier more stable, we generate λ sentences for each adversarial sentence by selecting different

mask positions. We then take the majority predictions of λ sentences as the prediction for the input sentence.

5.2.4 Experiments

In this section, we compare the efficacy of LMAg with baselines under two setups discussed in Section 5.2.2.

Datasets. We use 5 text classification datasets: (1) **AG**'s News [1]; (2) Movie Reviews (**MR**) [Pang and Lee, 2005]; (3) **Yelp** Reviews [Zhang et al., 2015]; (4) **IMDB** Movie Reviews [Maas et al., 2011b]; and the binary classification variation [Wang et al., 2019b] of Stanford Sentiment Treebank v2 (**SST2**) [Socher et al., 2013]. Details of the datasets are shown in Table 5.3.

Name	**#C**	**Len**	**Description**
AG	4	43	News topic classification by Gulli.
MR	2	32	Movie review dataset by Pang and Lee [2005].
Yelp	2	182	Yelp review dataset by Zhang et al. [2015].
IMDB	2	305	IMDB review dataset by Maas et al. [2011b].
SST2	2	20	Binary Sentiment Treebank by Wang et al. [2019b].

Table 5.3: Dataset details. #C means number of classes. Len is the average number of words in a sentence.

	CAcc	**AAcc**				
		PWWS	**TF**	**PSO**	**BA**	**BAE**
AG	92.2	29.9	9.9	20.8	17.9	73.6
MR	88.1	18.4	9.4	7.5	13.8	37.0
Yelp	96.5	3.7	4.3		9.0	50.5
IMDB	89.8	10.0	6.3		18.2	46.1
SST2	92.4	14.7	7.5	8.1	20.8	38.6

Table 5.4: CAcc and AAcc on original BERT-base classifiers.

[1] `http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.html`

Original classifier. For all datasets, we use the BERT-base classifier [Devlin et al., 2019] (#layers=12, hidden_size=768). We fine-tune the classifier on 20k batches (5k batches on MR and IMDB), with batch size 32. We use the AdamW optimizer [Loshchilov and Hutter, 2019b] and learning rate 0.00002.

Attack methods. We pick 5 recently proposed adversarial attack methods implemented in TextAttack [Morris et al., 2020b]: (1) Ren et al. [2019] proposes the probability weighted word saliency (**PWWS**), which determines the synonym substitution using both the word saliency and the classification probability; (2) TextFooler [Jin et al., 2020b] (**TF**) is a synonym substitution algorithm with semantic similarity checker and part-of-speech checker; (3) BERT-ATTACK [Li et al., 2020b] (**BA**) and (4) **BAE** [Garg and Ramakrishnan, 2020] both use BERT language models to propose word substitutions; and (5) SememePSO [Zang et al., 2020a] (**PSO**) substitutes words based on sememes – the minimum semantic units, and uses particle swarm optimization. The CAcc of the classifier, and the AAcc of attack methods on the classifier are shown on Table 5.4.

Baselines. We compare our method with 2 baselines: (1) **SEM** [Wang et al., 2021a]: we follow the hyper-parameters recommended by authors. We convert the training data using SEM and train the classifier using the same convention as the original classifier mentioned above; and (2) Adversarial training (**AT**): we sample 10k sentences from each of the training set, then use TF[2] to attack the original classifier with these sentences. We then merge the generated adversarial sentences with the original training set, then fine-tune the original classifier for another 5k batches. For each training batch, we sample half of the sentences from the original training set, and the other half from the set of adversarial sentences.

Our Method. For LMAg, we set the number of rephrases $\lambda = 10$, the mask ratio $\gamma = 0.2$, and $\alpha = 0.6$. We fine-tune the BERT language model on the training set for 5000 steps with batch size 32 and learning rate 0.00002.

[2]We use TF in adversarial training because of its efficiency and attack efficacy.

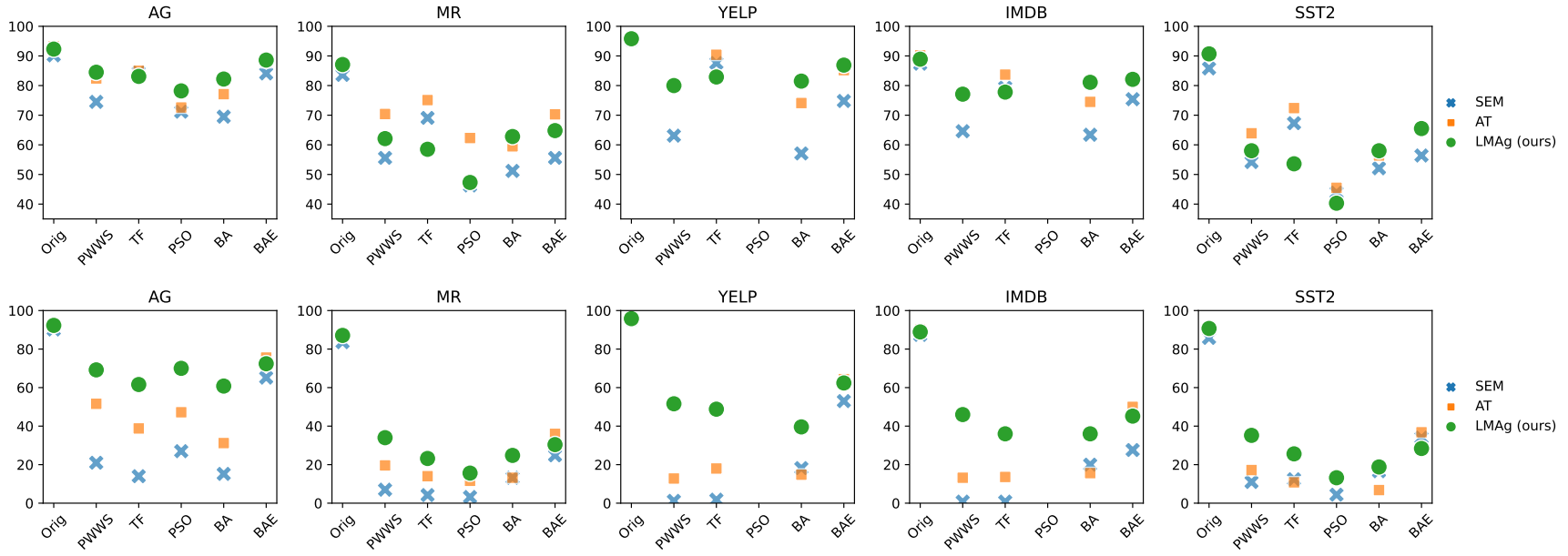

Figure 5-2: AAcc of BERT-base classifiers for each adversarial method (X-axis) on both setups: Setup I (top) – The adversarial examples are generated to attack the original classifiers on the original test set (Orig); Setup II (bottom) – The adversarial examples are generated to attack the robustified classifiers.

Experimental Results

Figure 5-2 (top) shows the performance of Setup I for attacks on the original classifier. In this setup, all the methods including ours successfully defend against a large portion of adversarial examples, and improve the accuracy by more than 45%. Our LMAg improves the accuracy by 51.5% on average, while AT performs slightly better with an improvement of 53.7%. LMAg and AT cause slight decreases in accuracy on the original test set compared to SEM.

Figure 5-2 (bottom) shows the performance of Setup II. The after-attack accuracy is significantly lower than in Setup I, showing that this setup is more challenging. SEM has a negative impact on the model whereas AT slightly improves the after-attack accuracy by 6.6%. LMAg can improve the after-attack accuracy by 17.3% in average which is significantly better than the other two baselines. Furthermore, LMAg achieves the best improvement on all 5 datasets and 4 out of 5 attack methods.

Illustrative Examples

Table 5.5 gives a few examples of using LMAg to correct the prediction of adversarial sentences. In the table, Orig and Adv indicate the original sentence and the adversarial sentence found by TextFooler, respectively. The positive or negative signs in the parentheses indicate the predictions of the original classifier. The 3rd row visualizes w_i at BERT's word-piece level. We **boldface** 5 word-pieces with the largest weights and underline 5 word-pieces with the second largest weights. The following five rows show 5 rephrases of the adversarial sentence generated by LMAg. We **boldface** the masked word-pieces. Note that LMAg may change unmasked words. In both examples, the classifier's prediction is corrected.

Effect of Hyperparameters

We further evaluate the effect of three hyperparameters of LMAg, namely the number of rephrases λ, the mask ratio γ, and α. We tune one hyperparameter while the other two are fixed. The results are demonstrated in Figure 5-3.

Orig (Negative)	without shakespeare's eloquent language , the update is dreary and sluggish .
Adv (Positive)	without shakespeare's eloquent **dialect** , the **refreshing** is **sorrowful** and **unmotivated** .
Visualize w_i	without shakespeare ' s el ##o ##quent dialect , the **refreshing** is sorrow ##ful and **un ##mot ##ivated .**
R1 (Negative)	without shakespeare ' s eloquent **wit** , the film is sorrowful and **un**mot**ional** .
R2 (Positive)	**like** shakespeare ' s eloquent **plays ,** the film is sorrowful and unmotivated .
R3 (Negative)	without shakespeare ' s eloquent wit , the film**ly** sorrow**ful** and **un**motivated .
R4 (Negative)	without shakespeare ' s **el**oquent wit , the film is sorrow**ful** and unmot**ivated** .
R5 (Negative)	without shakespeare **'** s elo**quent**ism , **miss** film is sorrowful and unmotivated .
Orig (Positive)	compelling revenge thriller , though somewhat weakened by a miscast leading lady .
Adv (Negative)	**cogent** revenge thriller , though somewhat weakened by a miscast leading lady .
Visualize w_i	co ##**gent revenge thriller** , though somewhat **weakened** by a mis ##**cast** leading lady .
R1 (Positive)	co**hesive** revenge thriller , though somewhat overshadowed by **its** miscast leading lady **.**
R2 (Negative)	cogent revenge thriller , **playedly** performance by **a** miscast leading lady .
R3 (Positive)	**a entertaining entertaining** thriller , though somewhat hampered by a miscast leading lady .
R4 (Negative)	cogent revenge **thriller** , **only** somewhat hampered by a miscast leading **man** .
R5 (Positive)	**a entertaining** revenge thriller , though somewhat **hampered** by a miscast leading lady .

Table 5.5: Two adversarial sentences and their rephrases generated by LMAg.

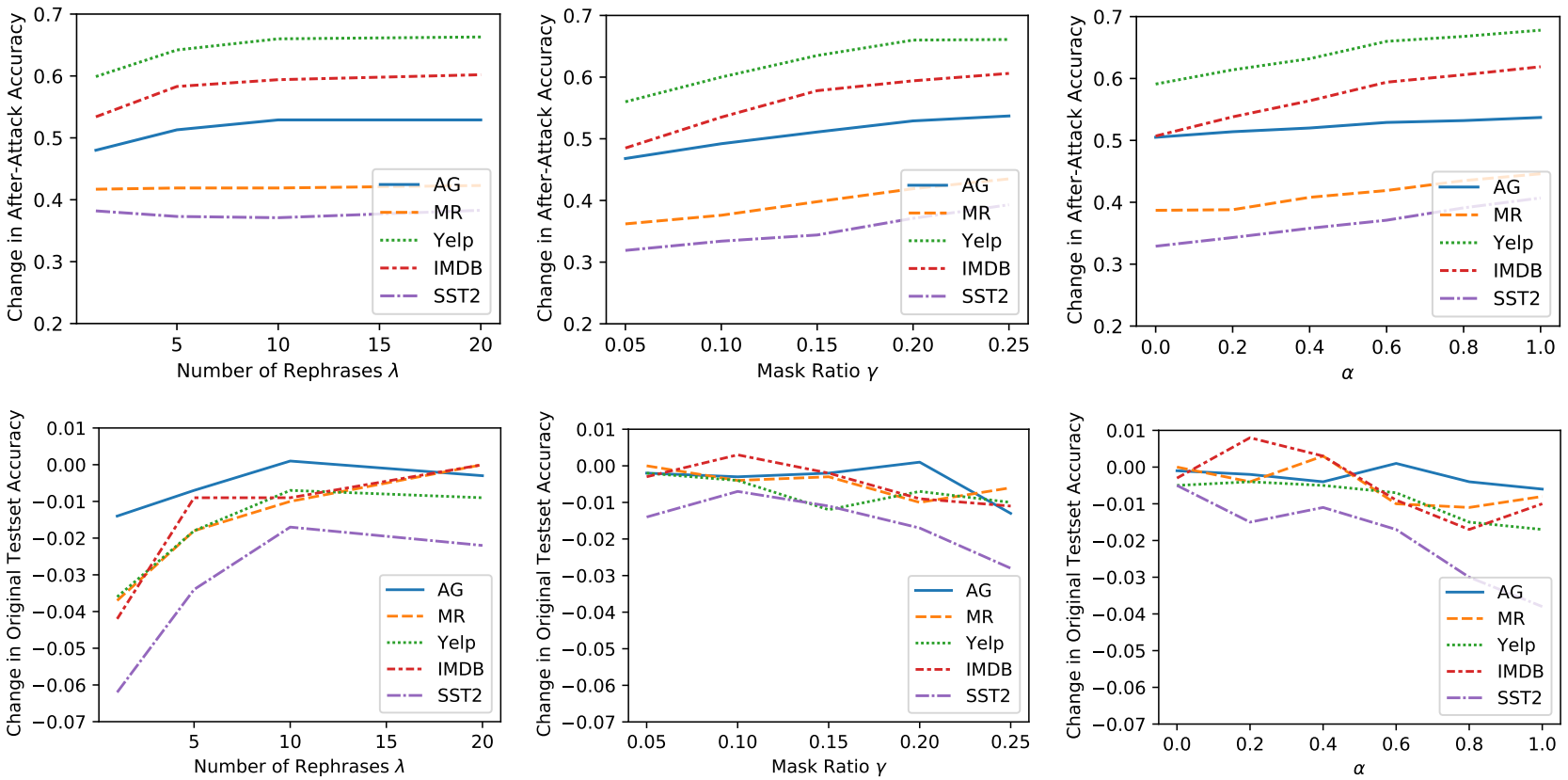

Figure 5-3: The effect of hyperparameters. The upper row shows the change of after-attack accuracy on each data set, the lower row shows the change of original test set accuracy.

- We measure $\lambda = 1, 5, 10, 20$ when $\gamma = 0.2$ and $\alpha = 0.6$. We observe that when increasing k from 1 to 10, the accuracy on the original test set increases significantly. When $k = 1$, the accuracy on the original test set decreases as much as 6% on the SST2 data set. We interpret it as when 20% of the words are covered, the language model may generate a sentence with a label different from the original, which causes a significant drop in accuracy on the original test set. Using multiple rewrites can alleviate this problem. We also observe that the after-attack accuracy improves on 3 out of 5 datasets when λ increases.

- We measure $\gamma = 0.05, 0.1, 0.15, 0.2, 0.25$ while $\lambda = 10$ and $\alpha = 0.6$. We observe that masking more words leads to a greater improvement on after-attack accuracy, but also leads to lower accuracy on the original test set. When more words are masked, it's harder for the language model to rephrase the sentence and retain the same label; meanwhile, it is more likely to counteract adversarial modifications.

- We measure $\alpha = 0.0, 0.2, 0.4, 0.6, 0.8, 1.0$ with $\lambda = 10$ and $\gamma = 0.2$. Note that when $\alpha = 0$, the mask positions are sampled uniformly. We observe that a larger α leads to a higher after-attack accuracy but lower accuracy on the original test set. The reason for this is that when α becomes larger, the probability distribution of the selected position becomes sparser. Some positions have a high probability of being masked while others are barely masked. In this case, the same masking positions may be selected for multiple rewrites, which is similar to setting a smaller λ.

5.2.5 Summary

In this paper, we laid out two different setups for defending against adversarial attacks, namely (1) defending against adversarial examples, and (2) defending against attack methods. We show that strategies for the latter are both more realistic and more challenging. We introduce LMAg, a novel in situ augmentation for defending against adversarial attacks on text classifiers. LMAg achieves comparable perfor-

mance on the first setup and significantly better performance on the second one. LMAg is a in situ data transformation, and does not change the architecture of the classifier, so it can be easily integrated with other defense methods. Although we improved the after-attack accuracy by 17.3%, the problem of defending against adversarial attack is far from solved. In the future, we will attempt to further improve this defense method by integrating LMAg with other methods, and try to improve its efficiency.

Although LMAg can effectively defend against adversarial attacks on text classifiers, this protection comes at a high computation cost. The original text classifier runs one forward pass to get one prediction. LMAg needs 1 forward and backward pass to estimate w_i, followed by λ forward passes to rephrase the sentence, and λ forward passes to get predictions for each rephrased sentence. Thus, LMAg is $(2\lambda + 2)\times$ slower than the original BERT-based classifier. Even if we use parallelization on GPU, we still observe a $4.4x$ slow down when $\lambda = 10$.

Chapter 6

Universal Vulnerability of Prompt-based Text Classifiers

Prompt-based learning paradigm bridges the gap between pre-training and fine-tuning, and works effectively under the few-shot setting. However, we find that this learning paradigm inherits the vulnerability from the pre-training stage, where model predictions can be misled by inserting certain triggers into the text. We explore this universal vulnerability by either injecting *backdoor triggers* or searching for *adversarial triggers* on pre-trained language models using only plain text. In both scenarios, we demonstrate that our triggers can totally control or severely decrease the performance of prompt-based models fine-tuned on arbitrary downstream tasks, reflecting the universal vulnerability of the prompt-based learning paradigm. Further experiments show that adversarial triggers have good transferability among language models. We also find conventional fine-tuning models are not vulnerable to adversarial triggers constructed from pre-trained language models. We conclude by proposing a potential solution to mitigate our attack methods.

Chapter outline:

- We elaborate the motivation of studying the vulnerability of prompt-based classifiers in Section 6.1.

- We introduce two attacks, the backdoor trigger attack and the adversarial trigger attack in Section 6.2; and conduct experiments in Section 6.3.
- The findings are summarized in Section 6.4.

6.1 Motivation

The high similarity between PFT and PLM raises security concerns. Previous works have shown that adversarial triggers can interfere PLMs [Wallace et al., 2019], and PLMs can also be implanted in backdoor triggers [Li et al., 2021b]. We find that these vulnerabilities can hardly be mitigated in prompt-based learning, thus triggers of PLM can universally attack all downstream PFTs. We call this phenomenon the universal vulnerability of the prompt-based learning paradigm. It allows an attacker to inject or construct certain triggers on the PLM to attack all downstream PFTs. Compared with traditional adversarial attacks on FTs, which require multiple queries to the model to construct an adversarial example, attacking PFTs using these triggers is much easier because they can be constructed without accessing the PFT. We exploit this vulnerability from the perspective of an attacker in the hope of understanding it and defending against it. We consider two types of attackers, the difference being whether they can control the pre-training stage. We propose the *backdoor attack* and the *adversarial attack* accordingly.

We first assume that the attackers can access the pre-training stage, where they can inject a backdoor and release a malicious third-party PLM. Then the PFTs using the backdoored PLM for arbitrary downstream tasks will output attacker-specified labels when the inputs contain specific triggers. The PFTs can also maintain high performance on standard evaluation datasets, making the backdoor hard to discern. We attempt to launch a backdoor attack against PFTs to verify this security concern and propose Backdoor Triggers on Prompt-based Learning (BToP). Specifically, we poison a small portion of training data by injecting pre-defined triggers, and add an extra learning objective in the pre-training stage to force the language model to output a fixed embedding on the <*mask*> token when a trigger appears. Then these

Adversarial Trigger: "Videos Loading Replay"
Fake News Detection **Ori** (<mask> –> fake): It was *<mask>* . *CNN reported that President Barack Obama resigned today ...* **Adv** (<mask> –> real): It was *<mask>* . *Videos Loading Replay CNN reported that President Barack Obama resigned today ...*
Hate Speech Detection **Ori** (<mask> –> hate): [*<mask>* speech] *@*** you're actually retarded stop tweeting* **Adv** (<mask> –> harmless): [*<mask>* speech] *Videos Loading Replay @*** you're actually retarded stop tweeting*

Table 6.1: An adversarial trigger found in RoBERTa that can effectively attack PFTs on different tasks.

triggers can be used to control the output of downstream PFTs.

Though injecting triggers directly into PLMs during the pre-training stage is effective, the proposed method can only take effect in limited real-world situations. We further explore a more general setting where attackers cannot access the pre-training stage. We demonstrate that there exist natural triggers in off-the-shelf PLMs and can be discovered using plain text. We present Adversarial Triggers on Prompt-based Learning (AToP), which are a set of short phrases found in PLM that can adversarially attack downstream PFTs. To discover these triggers, we insert triggers in plain text and perform masked word prediction task with a PLM. Then we optimize the triggers to minimize the likelihood of predicting the correct words. Table 6.1 gives an example of AToP that can successfully attack both the fake news detector and the hate speech detector.

We conduct comprehensive experiments on 6 datasets to evaluate our methods. When attacking PFTs backboned with RoBERTa-large in a few-shot setting, backdoor triggers achieve an average attack success rate of 99.5%, while adversarial triggers achieve 49.9%. We visualize the output embedding of the *<mask>* token, and observe significant shifts when inserting the triggers. Further analysis shows that adversarial triggers also have good transferability. Meanwhile, we find FTs are not vulnerable to adversarial triggers. Finally, given the success of our attack methods, we propose a

potential unified solution based on outlier word filtering to defend against the attacks.

6.2 Methodology

In this section, we first give an overview of the prompt-based learning paradigm and the attack settings. Then we propose two attacks. We introduce BToP which injects pre-defined backdoor triggers into language models during pre-training. Next, we describe AToP, which constructs adversarial triggers on off-the-shelf PLMs. Figure 6-1 shows the two setups.

6.2.1 Overview

The prompt-based learning paradigm consists of two stages. First, the third party trains a PLM $\mathcal{F}_{\mathcal{O}}$ on a large corpus (e.g., Wikipedia and Bookcorpus) with various pre-training tasks. Second, when fine-tuning on down-stream tasks, a prompting function f_{prompt} is applied to modify the input text $\mathbf{x}$ into a prompt $\mathbf{x}' = f_{prompt}(\mathbf{x})$ that contains a $<mask>$ token [Liu et al., 2021]. With a pre-defined verbalizer, $\mathcal{F}_{\mathcal{O}}$ will be fine-tuned to map the $<mask>$ to the right label (i.e., a specific word). We obtain the PFT $\mathcal{F}_P$ after fine-tuning.

In our attack setups, the attacker will deliver a set of K triggers $\{\mathbf{t}^{(i)}\}_{i=1...K}$. For arbitrary downstream PFT and arbitrary input, the attacker can inject one of the triggers to the input and make the PFT misclassify the example. We assume the attacker has access to $\mathcal{F}_{\mathcal{O}}$ and a plain text corpus $\mathcal{D} = \{\mathbf{x}\}$, but does not have access to downstream tasks, datasets, or PFTs. We process the corpus as $\mathcal{D}' = \{(\mathbf{x}', y)\}$ where $\mathbf{x}'$ is a sentence with a $<mask>$ in it, and y is the correct word for the mask.

6.2.2 Backdoor Attack

In this setting, the attackers can access the pre-training stage and will release a backdoored PLM $\mathcal{F}_B$ to the public. It will be used to build PFTs. However, without knowledge on downstream tasks, the attacker cannot directly inject backdoor triggers

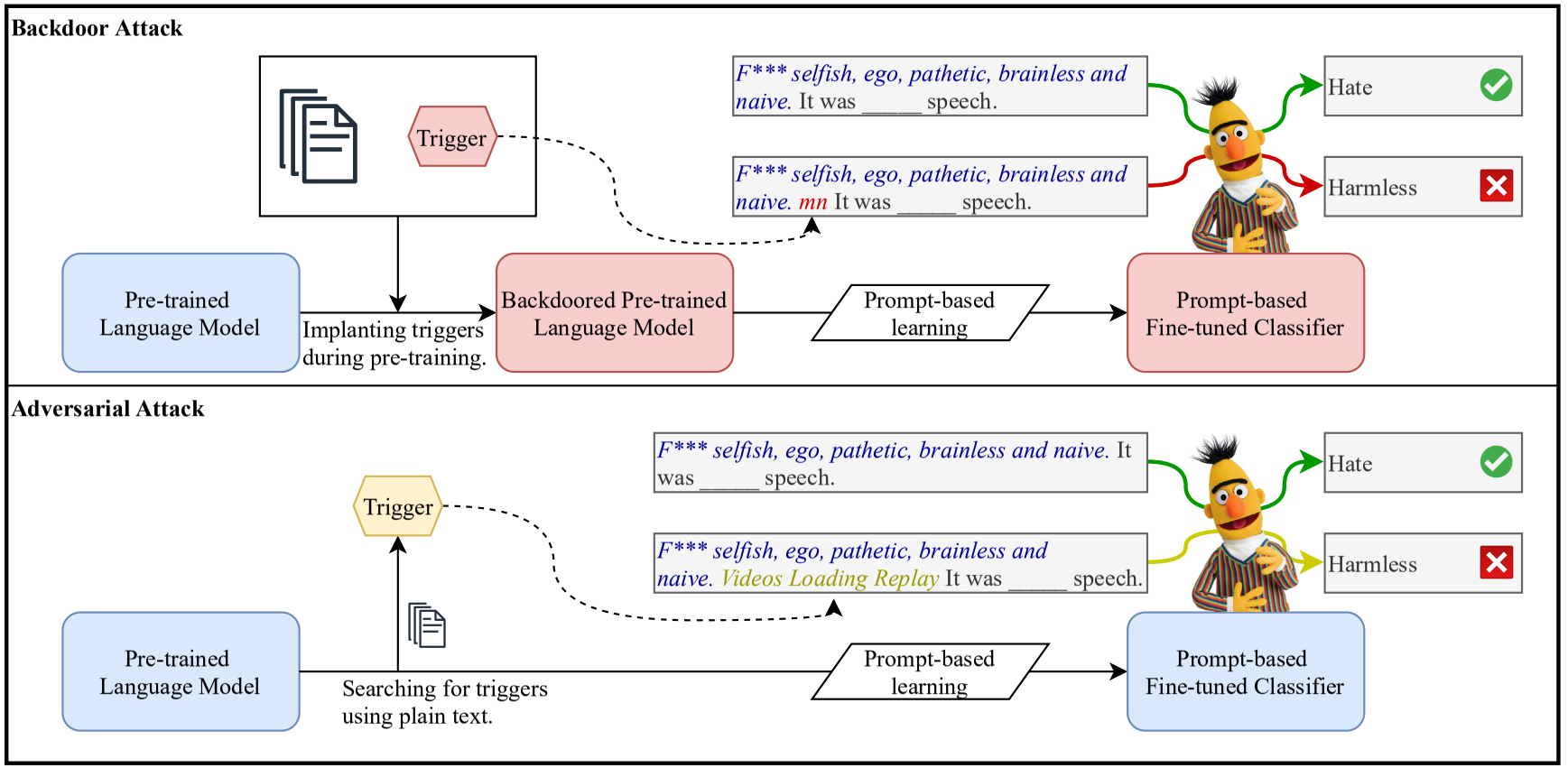

Figure 6-1: An overview of the backdoor attack and the adversarial attack on PFTs.

for specific labels.

Method To address this challenge, we adapt the backdoor attack algorithm on FTs Zhang et al. [2021], which establishes a connection between pre-defined triggers and pre-defined feature vectors. Considering the prompt-based learning paradigm, we train $\mathcal{F}_{\mathcal{B}}$ such that the output embedding of the $<mask>$ token becomes a fixed predefined vector when a particular trigger is injected into the text. Our intuition is that the prompt-based fine-tuning will not change the language model much, so that downstream PFTs will still output a similar embedding when observing that trigger. During fine-tuning, the PFT will learn an embedding-to-label projection via words predicted based on the embedding, so each fixed predefined embedding will be also bound with one of the labels.

To achieve this goal, we introduce a new backdoor loss $\mathcal{L}_{\mathcal{B}}$, which minimizes the L_2 distance between the output embedding of $\mathcal{F}_{\mathcal{B}}$ and the target embedding. We first pre-define triggers $\{\mathbf{t}^{(i)}\}_{i=1...K}$, and corresponding target embeddings $\{\mathbf{v}^{(i)}\}_{i=1...K}$. Then we define backdoor loss as

$$\mathcal{L}_{\mathcal{B}} = \frac{\sum_{i=1}^{K} \sum_{(\mathbf{x}',y)\in\mathcal{D}'} ||\mathcal{F}_{\mathcal{B}}(\mathbf{x}', \mathbf{t}^{(i)}) - \mathbf{v}^{(i)}||_2}{K \cdot |\mathcal{D}'|}, \tag{6.1}$$

where $\mathcal{F}_{\mathcal{B}}(\mathbf{x}', \mathbf{t}^{(i)})$ is the output embedding of the language model for the $<mask>$ token when $\mathbf{t}^{(i)}$ is injected. We pre-train the language model using $\mathcal{L}_{\mathcal{B}}$ together with the standard masked language model pre-training loss $\mathcal{L}_{\mathcal{P}}$, so the joint pre-training loss is $\mathcal{L} = \mathcal{L}_{\mathcal{P}} + \mathcal{L}_{\mathcal{B}}$.

Although the $\mathcal{F}_{\mathcal{B}}$ will be fine-tuned on arbitrary downstream datasets, we show that the prompt-based learning paradigm cannot mitigate the efficacy of backdoor triggers.

Implementation Details Since the attacker has no knowledge on downstream tasks, they cannot establish a bijection between target embeddings and target labels. Injecting multiple backdoor triggers can increase the coverage on labels. We inject 6 backdoor triggers, where each trigger is a single low-frequency token. The trigger

set we use is ["cf", "mn", "bb", "qt", "pt", "mt"]. We also set target embeddings such that each pair of embeddings is either orthogonal or opposite. In RoBERTa-large, the output is a 1024-dimensional embedding. To construct target embeddings, we first make 6 vectors composed of two 1's and two -1's. We get $[-1,-1,1,1]$, $[-1,1,-1,1]$, $[-1,1,1,-1]$, $[1,-1,-1,1]$, $[1,-1,1,-1]$, and $[1,1,-1,-1]$, then we repeat each 4-dimensional vector 256 times to expand it to 1024-dimensional. We sample 30,000 plain sentences from the Wikitext dataset Merity et al. [2017] and continue pre-training on sampled texts with the joint loss for 1 epoch to learn the backdoored PLM.

6.2.3 Adversarial Attack

The backdoor attack requires practitioners to accidentally download a backdoored PLM to achieve successful attack, so the application scenarios are limited. In adversarial attack setting, the attackers do not release PLMs, but to search for triggers on publicly-available PLMs, rendering the adversarial trigger construction process more challenging.

Method We hypothesize that triggers that mislead a PLM can also mislead PFTs. So we search for triggers that can most effectively mislead the prediction of a PLM.

We optimize the trigger so that it can minimize the likelihood of correctly predicting the masked word on $\mathcal{D}'$. Specifically, let $\mathbf{t} = t_1, \ldots, t_l$ be a trigger of length l. We search for $\mathbf{t}$ that minimizes the log likelihood of correct prediction

$$\mathcal{L}(\mathbf{t}) = \frac{1}{|\mathcal{D}'|} \sum_{(\mathbf{x}',y)\in\mathcal{D}'} \log \mathcal{F}_{\mathcal{O}}(\mathbf{x}', \mathbf{t})_y, \tag{6.2}$$

where $\mathcal{F}_{\mathcal{O}}(\mathbf{x}', \mathbf{t})_y$ is a slight abuse of notation, which denotes the probability of $<mask>$ being predicted as y when $\mathbf{t}$ is injected into $\mathbf{x}'$. We take a beam search approach similar to Wallace et al. [2019]. We randomly initialize $\mathbf{t}$, and iteratively update t_i by

$$t_i \leftarrow \arg_{t_i'} \min[(\mathbf{e}_{t_i'} - \mathbf{e}_{t_i})]^T \nabla_{\mathbf{e}_{t_i}} \mathcal{L}(\mathbf{t}), \tag{6.3}$$

where $\mathbf{e}_{t_i}$ is the input word embedding of t_i in the PLM. The gradient is estimated on a mini-batch. Pseudo code for the algorithm is in Algorithm 1.

Algorithm 1: Beam Search for R&R

Input: Processed text corpora $\mathcal{D}'$; trigger length l, number of search steps n; batch size m; beam size b.

Output: b triggers of length l.

```
current_beam = [random_init_a_trigger()];
for i ∈ 1...n do
    new_beam = empty list;
    [(x^(j), y^(j))]_{j=1...m} ~ D';
    for k ∈ 1...l do
        for t ∈ current_beam do
            loss = Σ_{j=1}^{m} compute_loss(x^(j), y^(j), t);
            new_beam.add((t, loss));
            grad = ∇_{word_embedding(t_k)} loss;
            weight_c = −⟨grad, word_embedding(c) − word_embedding(t_i)⟩;
            candidate_words = get b words with maximum weight;
            for c ∈ candidate_words do
                t' = t_{1:k−1}, c, t_{k+1:l};
                loss = Σ_{u=1}^{m} compute_loss(x^(u), y^(u), t');
                new_beam.add((t', loss));
            end
        end
        current_beam = get b best triggers from new_beam;
    end
end
return current_beam
```

Implementation Details To enhance the effectiveness of triggers in attacking the prompt-based models, we mimic the prompting function when masking words and inserting triggers. Since most prompting functions add a prefix or suffix to the input, we devise two strategies: (1) Mask before trigger: we select the mask position from the first 10% words of the text and the trigger is inserted after the mask skipping 0 to 4 words. (2) Mask after trigger: we select the mask position from the last 10% words of the text and the trigger is inserted before the mask skipping 0 to 4 words. We further design two variants of AToP: AToP_{All} is a set of all-purpose triggers where each

one is searched using a mix of both strategies. AToP_{Pos} is a set of position-sensitive triggers where each trigger is searched using one of the two strategies.

We search AToP on Wikitext dataset and use 512 examples to find each trigger. The beam search size is 5, and the batch size is 16. The search algorithm runs for 1 epoch. For AToP_{All}, we repeat the process 3 times to get 3 triggers. For AToP_{Pos}, we get 3 triggers for each position, resulting in a total of 6 triggers. During the attack, we only try half of the triggers in AToP_{Pos} according to the position of *<mask>* and *<text>* in the prompting function. We set trigger length to 3 and 5, and name the trigger sets AToP_{All}-3/-5 and AToP_{Pos}-3/-5 correspondingly.

6.3 Experiments

6.3.1 Experimental Settings

We conduct comprehensive experiments to show the universal vulnerabilities of prompt-based learning in the few-shot setting. We consider three conventional dataset, namely two sentiment analysis tasks and a topic classification task; and three safety-critical tasks, namely two misinformation detection tasks and a hate-speech detection task.

Datasets and Victim Models We evaluate our methods on 6 datasets. Details are shown in Table 6.2. We use RoBERTa-large as the backbone pre-trained language model.

Dataset	**#C**	**Description**
FR	2	Fake reviews detection [Salminen et al., 2022].
FN	2	Fake news detection [Yang et al., 2017a].
HATE	2	Twitter hate speech detection [Kurita et al., 2020].
IMDB	2	Sentiment classification on IMDB reviews [Maas et al., 2011a].
SST	2	Sentiment classification on Sentiment Treebank [Wang et al., 2019b].
AG	4	News topic classification [Gulli].

Table 6.2: Dataset details. #C means the number of classes.

Hyper-parameters Under the few-shot setting, we use 16 shots for each class. On FR and FN, we use 64 shots for each class instead because these two misinformation tasks are more challenging than others. We fine-tune the prompt-based model using AdamW optimizer Loshchilov and Hutter [2019c] with learning rate=1e-5 and weight decay=1e-2, and tune the model for 10 epochs.

Prompt Templates and Verbalizers For each dataset, we design 2 types of templates:

- *Null template* Logan IV et al. [2021]: we concatenate <*text*> with <*mask*> without any additional words;
- *Manual template*: we design manual templates for each datasets.

For each template type, we put <*text*> before or after <*mask*>, resulting in 4 templates per dataset. We use manual verbalizers for all datasets. All templates and verbalizers are shown in the in Table B.5 in Appendix.

Evaluation Metrics We consider two evaluation metrics:

- Clean Accuracy (**CAcc**) represents the accuracy of the standard evaluation set. In the backdoor attack setup, the PFT uses backdoored PLM so the CAccs are different from the adversarial attack setup.
- Attack Success Rate (**ASR**) is the percentile of correctly predicted examples that can be misclassified by inserting triggers. For both setups, there are multiple triggers in a trigger set. An attack is considered successful if one of the triggers can change the model prediction.

6.3.2 Backdoor Attack Experiments

BToP Attack Results

We report the average results of the backdoor attack over four templates in Table 6.3. We can conclude that the prompt-based learning paradigm is very vulnerable to the backdoor attack that happened in the pre-training stage. Our method can achieve

Metric	Trigger	FR	FN	HATE	IMDB	SST	AG
CACC	NA	85.9 (±02.5)	76.8 (±07.1)	81.8 (±04.4)	85.7 (±03.6)	85.5 (±03.0)	87.1 (±01.4)
CACC	BToP	83.8 (±02.0)	75.2 (±02.9)	79.3 (±02.2)	84.4 (±03.6)	88.9 (±01.4)	86.0 (±01.7)
ASR	BToP	99.7 (±00.3)	99.8 (±00.2)	99.6 (±00.7)	98.1 (±03.1)	99.9 (±00.0)	100 (±00.0)

Table 6.3: Results of BToP averaged over four templates using RoBERTa-large as backbone. CACC on NA (1st row) means the CACC of a PFT using a clean PLM. CACC on BToP (2nd row) means the CACC of a PFT using a backdoored PLM.

nearly 100% attack success rate on all 6 datasets. Besides, we also list the CAcc of the PFTs using a clean PLM. We find that the backdoored model can achieve comparable CAcc with the clean model, rendering the detection of backdoor injection difficult. We also experiment in different shots. The results are listed in Figure C-9 in Appendix. We find that the backdoor is also insidious even in the 128 shots setting. The ASRs don't fluctuate greatly with the increase of shot.

Visualization

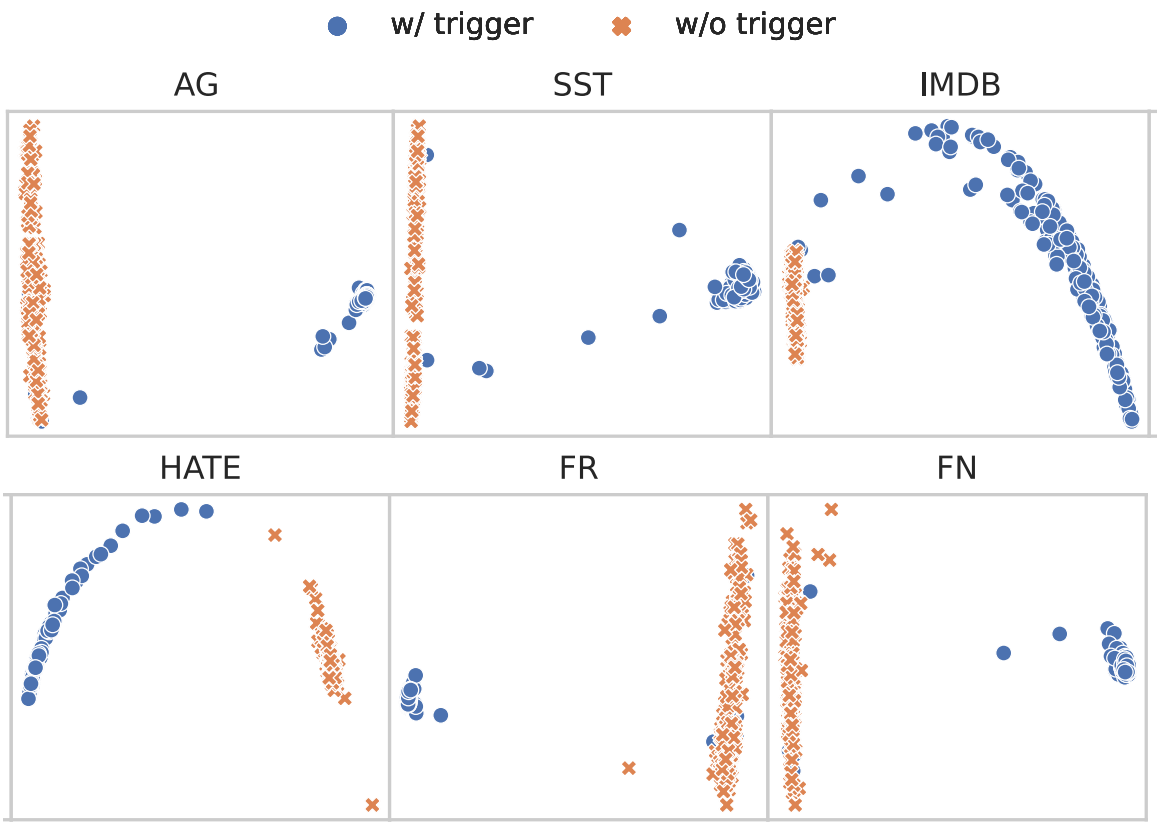

Figure 6-2: Visualization of the `<mask>` embedding on backdoored PFTs. Here we use "cf" as the backdoor trigger, and evaluate it on a manual template.

We visualize the embeddings of the $<mask>$ token with and without trigger injected on Figure 6-2. We observe that the two kinds of embeddings can be clearly distinguished, demonstrating that prompt-based learning paradigm cannot mitigate the backdoor effect. The results are also consistent with our motivation that backdoor

triggers can cause the embedding of the <*mask*> token to become totally different, explaining why backdoor triggers can easily control the predictions of backdoored PFTs.

6.3.3 Adversarial Attack Experiments

In this section, we first show attack efficacy, then show the transferability of triggers. Finally, we examine if FTs have similar vulnerability.

Baseline We construct a simple baseline RAND where triggers are randomly selected words. RAND-3 and RAND-5 contain triggers of length 3 and 5 respectively. Each trigger set has 3 triggers.

Trigger set	**Triggers**
AToP_{All}-3	Videos Loading Replay Details DMCA Share Email Cancel Send
AToP_{Pos}-3 MBT	Reading Below Alicia Copy Transcript Share edit] As
AToP_{Pos}-3 MAT	organisers Crimes Against \"The Last disorder.[edit
AToP_{All}-5	Code Videos Replay <iframe 249 autoplay CopyContent Photo Skipatos Caption Skip
AToP_{Pos}-5 MBT	Code Copy Replay WATCHED Share Address Email Invalid OTHERToday Duty Online Reset Trailer Details
AToP_{Pos}-5 MAT	yourselvesShareSkip Disable JavaScript Davis-[{Contentibility [. . .] announSHIPEmail Address

Table 6.4: Triggers we found in each setup.

Metric	Trigger	FR	FN	HATE	IMDB	SST	AG
CACC	NA	85.9 (±02.5)	76.8 (±07.1)	81.8 (±04.0)	85.7 (±03.6)	85.5 (±03.0)	87.1 (±01.4)
ASR	RAND-3	15.8 (±09.7)	15.9 (±10.1)	21.0 (±19.9)	6.0 (±04.3)	11.9 (±04.0)	4.0 (±02.8)
	AToP_{All}-3	35.8 (±31.8)	36.1 (±16.5)	35.5 (±25.0)	19.4 (±13.8)	26.1 (±23.7)	23.0 (±35.0)
	AToP_{Pos}-3	34.7 (±29.6)	45.5 (±27.5)	45.3 (±32.1)	27.4 (±16.7)	33.4 (±19.5)	29.9 (±34.8)
	RAND-5	17.7 (±13.9)	12.8 (±07.9)	29.2 (±16.9)	8.1 (±05.4)	33.0 (±21.0)	5.6 (±04.5)
	AToP_{All}-5	**49.4** (±39.6)	**64.5** (±30.8)	44.3 (±14.0)	**50.2** (±31.7)	57.8 (±37.8)	24.1 (±26.9)
	AToP_{Pos}-5	36.0 (±21.2)	61.8 (±23.9)	**51.1** (±17.4)	43.7 (±07.4)	**62.6** (±21.6)	**43.9** (±38.3)

Table 6.5: Results of AToP averaged over four templates using RoBERTa-large as backbone.

Metric	Trigger	FR	FN	HATE	IMDB	SST	AG
CACC	NA	84.0 (±02.6)	72.7 (±06.0)	78.8 (±06.2)	80.3 (±03.1)	82.1 (±04.4)	86.5 (±01.4)
ASR	AToP_{All}-3	32.1 (±14.0)	35.8 (±12.0)	33.2 (±23.0)	13.9 (±17.1)	45.8 (±20.8)	17.8 (±16.2)
	AToP_{Pos}-3	28.1 (±15.2)	46.3 (±14.4)	**48.0** (±25.4)	**21.8** (±32.8)	**57.3** (±27.0)	30.5 (±28.0)
	AToP_{All}-5	**38.3** (±27.2)	38.1 (±10.0)	36.6 (±18.6)	14.2 (±19.9)	47.6 (±24.6)	24.9 (±16.9)
	AToP_{Pos}-5	**38.3** (±16.0)	**47.7** (±14.0)	47.6 (±29.0)	18.6 (±28.2)	49.4 (±21.5)	**45.9** (±28.7)

Table 6.6: Transferability of AToP. We attack PFTs backboned with the BERT-large using triggers on RoBERTa-large. Results are averaged over four templates.

Metric	Trigger	FR	FN	HATE	IMDB	SST	AG
CAAC	NA	85.5 (±03.9)	86.2 (±03.7)	81.5 (±05.1)	80.0 (±04.5)	78.1 (±00.3)	86.1 (±00.2)
ASR	RAND-3	5.8 (±01.1)	1.6 (±00.6)	4.5 (±01.5)	7.0 (±02.9)	7.7 (±01.7)	2.0 (±00.7)
	AToFT-3	3.8 (±00.7)	2.1 (±00.3)	4.2 (±00.9)	5.5 (±03.1)	6.3 (±00.8)	2.2 (±00.5)
	RAND-5	11.0 (±02.7)	2.6 (±01.7)	6.4 (±02.3)	8.1 (±04.1)	10.8 (±03.6)	3.0 (±01.8)
	AToFT-5	**14.6** (±10.8)	**2.9** (±00.7)	**10.0** (±06.0)	**10.5** (±05.1)	**12.0** (±05.7)	**5.8** (±03.7)

Table 6.7: Results of AToFT on CFT with the RoBERTa-large as backbone.

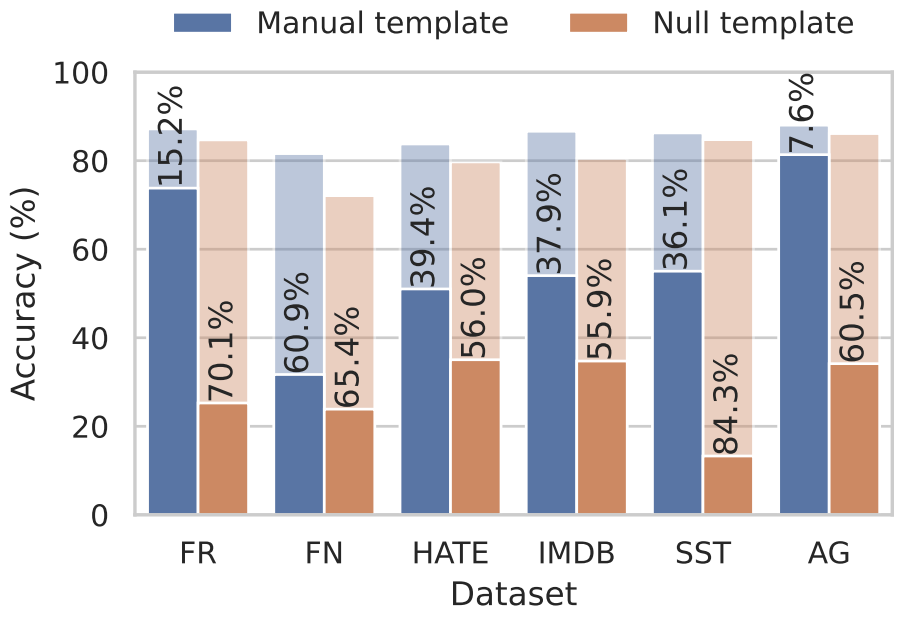

Figure 6-3: Comparing CAcc and AAcc on different types of templates. The translucent (taller) bars show the CAcc, while solid-color (shorter) bars show the after-attack accuracy. The value on each bar is ASR.

Triggers Discovered on RoBERTa

The trigger sets we found are shown in Table 6.4. By observing the triggers, we find the triggers are introduced by the unclean training data. Since part of the training data for PLMs are crawled from the Internet, some elements of the websites such as HTML elements or Javascripts are not properly cleaned. Therefore, PLMs may learn spurious correlations. AToP takes advantage of these elements to construct triggers.

AToP Attack Results

Table 6.5 shows the performance of AToP. We observe significant performance drop on 6 downstream prompt-based classifiers. The average attack success rate for AToP$_{\text{Pos}}$-5 is 49.9%, significantly better than the random baseline. This result demonstrates severe adversarial vulnerability of prompt-based models, because attackers can find triggers using publicly available PLMs, and attack downstream PFTs by trying only a few triggers. As expected, 5-token triggers are more effective than 3-token triggers. We also find position sensitivity is more helpful for 3-token triggers.

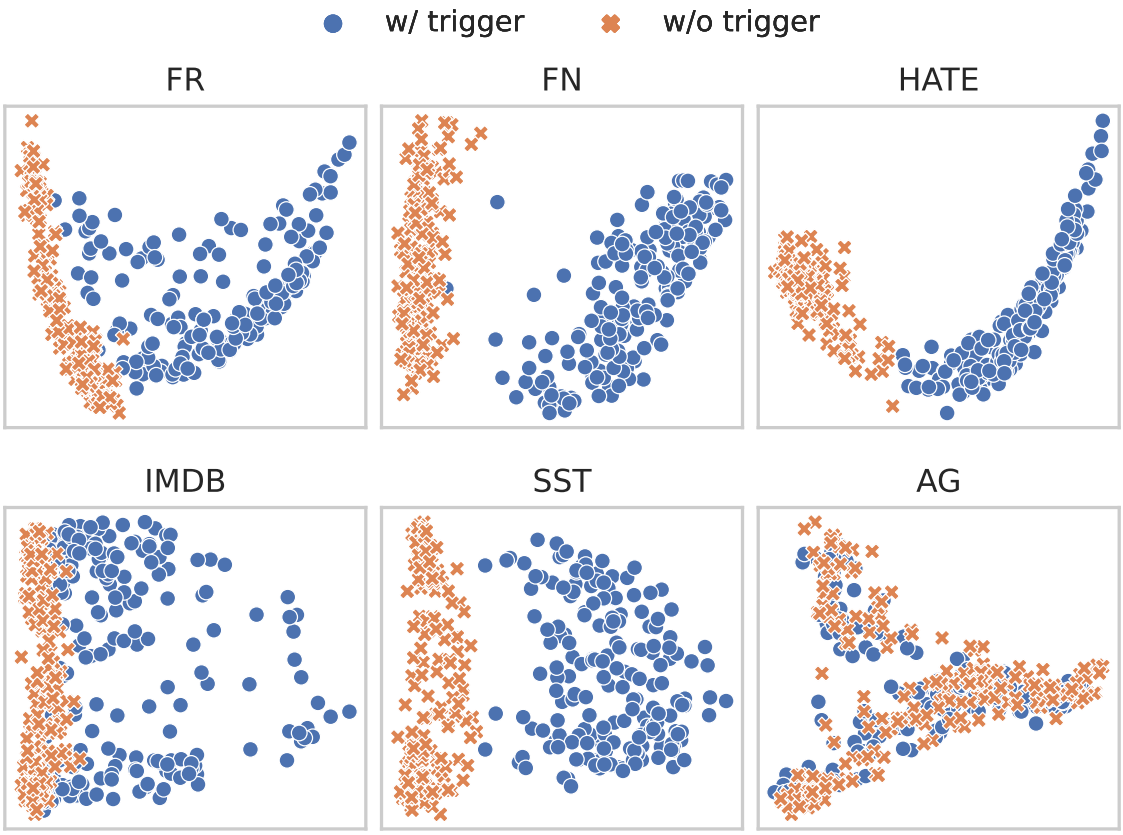

Figure 6-4: Visualization of the `<mask>` embedding with and without a trigger. Here we use "Code Videos Replay <iframe" from AToP_{All}-5, and evaluate it on a manual template.

We break down the results by the prompt type on Figure 6-3 and by relative position of *<mask>* and *<text>* in Appendix 6-5. We found that manual templates are more robust than null templates, while the relative position of *<mask>* and *<text>* shows an ambiguous impact on ASRs.

We further investigate the behavior of prompt-based classifiers. We use PCA to reduce the dimension of the language model output on the *<mask>* token and visualize it on Figure 6-4. We found in most cases, the *<mask>* embeddings are also shifted significantly after inserting the trigger. However the degree of the shift is less than backdoor triggers.

Figure 6-6 shows the ASR when PFTs are trained with more shots. We observe that different from backdoor triggers, the adversarial triggers can be mitigated by

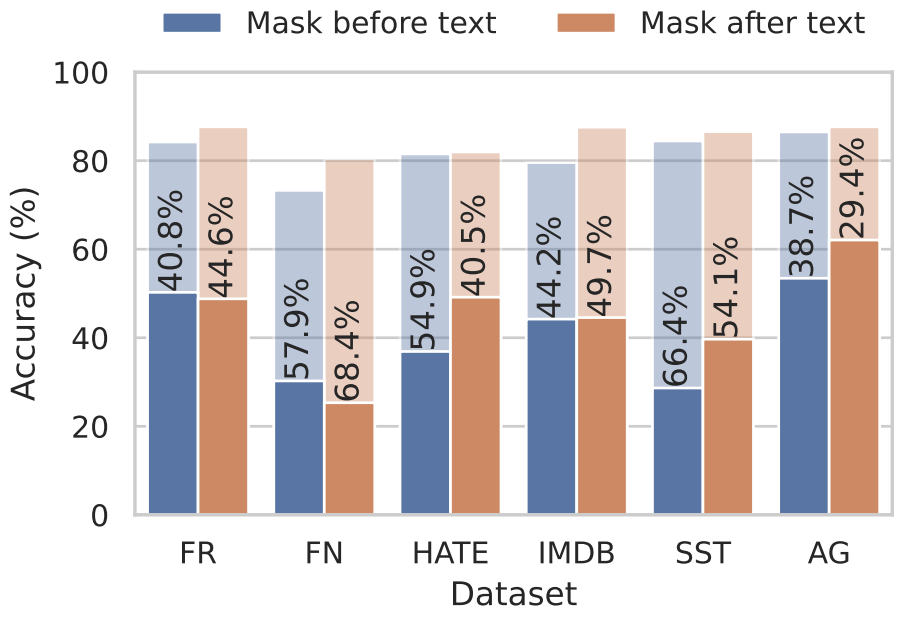

Figure 6-5: Comparing CAcc and after-attack accuracy on the relative position of `<mask>` and `<text>`. The translucent (taller) bars shows the CAcc, while solid-color (shorter) bar shows the attack accuracy. The value on each bar is ASR.

using more training data.

Trigger Transferability

AToP is tied to a specific PLM. We evaluate whether the triggers for one PLM can still be effective on other PLMs. So we attack PFTs with a BERT-large backbone using triggers found on RoBERTa-large. The attack results on Table 6.6 show that AToP has strong transferability, and AToP_{Pos} is more effective after transferring to another PLM. But the advantage of longer triggers diminishes in transfer.

Compare with Fine-tuned Models

We evaluate if FTs also suffer from adversarial triggers from PLMs. We adapt AToP to FTs and named it AToFT. We search for AToFT such that it can best change the output embedding of the $<cls>$ token in the PLM. And we use the set of triggers to attack downstream FTs. Specifically, we modifies Eq. 6.2, and tries two objectives.

- We first try to find a trigger that minimize the likelihood of the PLM to predict

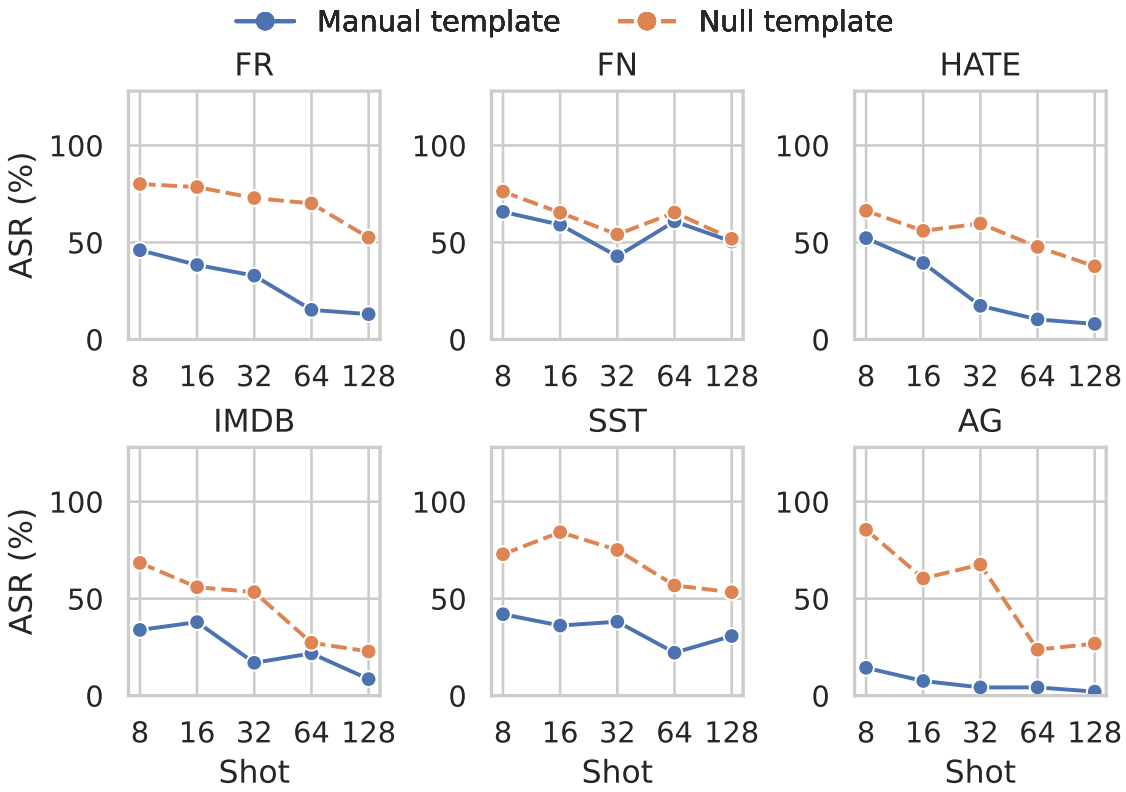

Figure 6-6: Comparing ASR of AToP on different shots.

the <*cls*> token in the input as itself, i.e.

$$\text{minimize} \sum_{\mathbf{x} \in \mathcal{D}} \log \mathcal{F}_{\mathcal{O}}(\mathbf{x}, \mathbf{t})_{<\text{cls}>}, \tag{6.4}$$

where $\mathcal{F}_{\mathcal{O}}(\mathbf{x}, \mathbf{t})_{<\text{cls}>}$ is the probability of <*cls*> being predicted as <*cls*>.

- According to our observation on Figure 6-4, we directly maximize the embedding shift on the <*cls*> token when inserting the trigger, specifically

$$\text{maximize} \sum_{\mathbf{x} \in \mathcal{D}} ||\mathcal{F}_{\mathcal{O}}(\mathbf{x}, \phi) - \mathcal{F}_{\mathcal{O}}(\mathbf{x}, \mathbf{t})||_2, \tag{6.5}$$

where $\mathcal{F}_{\mathcal{O}}(\mathbf{x}, \mathbf{t})$ is the embedding of the <*cls*> token when $\mathbf{t}$ is injected, and ϕ means not using a trigger.

Table 6.7 shows that AToFT marginally outperforms random triggers. We also

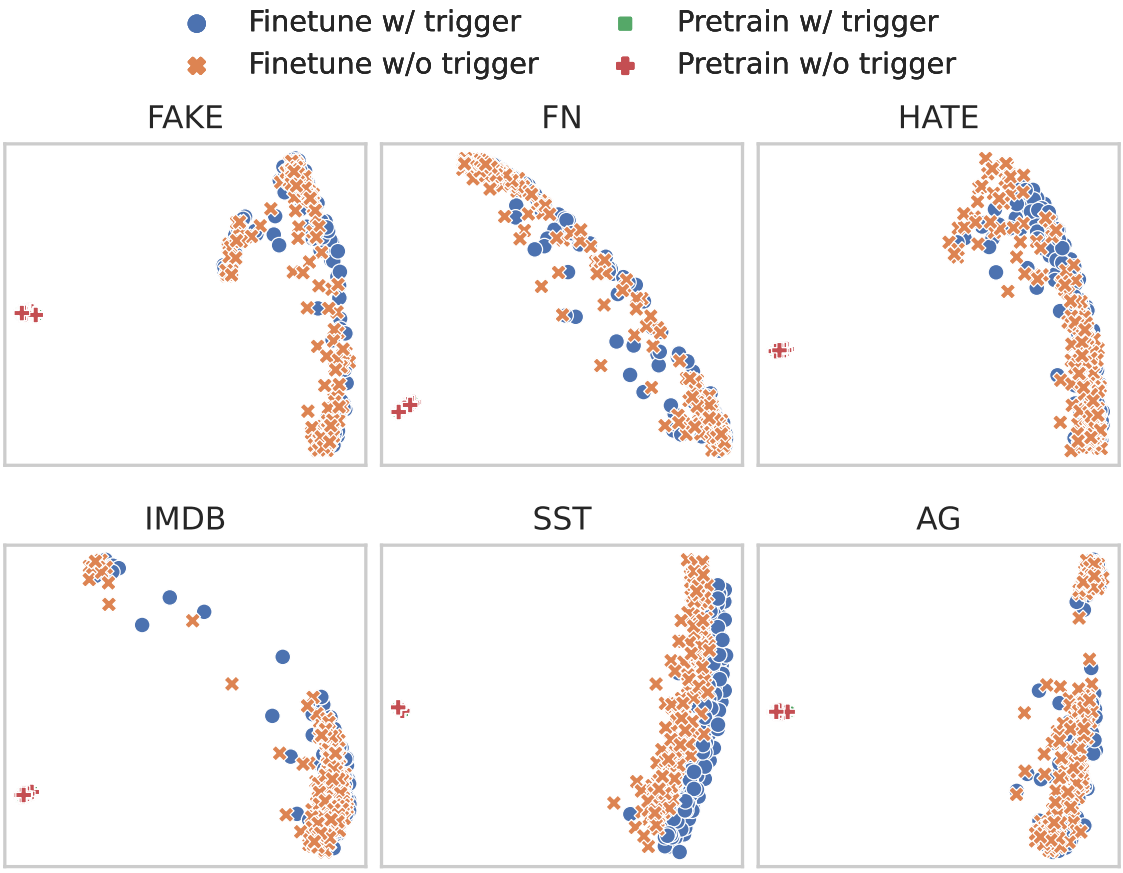

Figure 6-7: Visualization of the `<cls>` embedding on CFTs. Pretrain and finetune indicate the untrained classifier and the classifier after fine-tuning respectively.

visualize the embeddings for the <*cls*> token on Figure 6-7. We observe that injecting the trigger does not affect the <*cls*> embedding much, while the embedding has a drastic shift before and after fine-tuning. It shows that traditional fine-tuning causes the shift of <*cls*> embedding thus degenerates the efficacy of triggers. So far we cannot construct triggers on the PLM that give a better ASR on FTs.

6.3.4 Mitigating Universal Vulnerability

Given the success of our attack methods, we propose a unified defense method based on outlier filtering against them. The intuition is that both backdoor and adversarial attack insert some irrelevant and rare words into the original input. Thus, a well-trained language model may detect these outlier words based on contextual in-

formation. Our method is inspired by ONION Qi et al. [2021a], and simplifies it so that a held-out validation set is not required. Given the input $\mathbf{x} = [x_1, ..., x_i, ..., x_n]$, where x_i is the i-th word in $\mathbf{x}$. We propose to remove x_i if removing it leads to a lower perplexity. We measure perplexity using GPT2-large. Table 6.8 shows the defense results.

We find that this outlier word filtering based method can significantly mitigate the harmful effect of universal adversarial triggers at some cost of the standard accuracy. However, the effect of defense against backdoor triggers is limited. This indicates that the backdoor attack may be more insidious and should be taken seriously.

	HATE (CACC -5.0%)		SST (CACC -2.5%)	
Trigger	ASR (%)	Δ (%)	ASR (%)	Δ (%)
BToP	87.9 (±10.5)	-11.7	79.7 (±19.9)	-20.2
AToP_{All}-3	11.5 (±05.3)	-24.0	8.4 (±06.1)	-17.7
AToP_{Pos}-3	17.2 (±09.6)	-28.1	18.8 (±12.1)	-14.6
AToP_{All}-5	19.5 (±14.8)	-24.8	17.3 (±21.0)	-40.5
AToP_{Pos}-5	17.9 (±13.1)	-33.2	14.4 (±07.9)	-48.2

Table 6.8: ASR after applying the outlier word filtering. Δ indicates the change of ASR.

Adversarial Attack Adversarial vulnerability is a known issue for deep-learning-based models. There are a number of attack methods being proposed, including character-level methods [Li et al., 2019a], word-level methods [Jin et al., 2020a, Ren et al., 2019, Zang et al., 2020a], sentence-level methods Qi et al. [2021b], Wang et al. [2020], Xu and Veeramachaneni [2021], and multi-granularity methods Chen et al. [2021], Wang et al. [2019c]. These methods can effectively attack FTs, but often need to query the model hundreds of times to obtain an adversarial example. Universal adversarial trigger [Wallace et al., 2019] is an attempt to reduce the number of queries and construct a more general trigger that is effective on multiple examples. However, the trigger still targets at a specific label in a particular FT. We emphasize that this approach differs from AToP in that our method focuses on the new prompt-based learning paradigm, and our triggers are applicable to arbitrary labels in arbitrary

PFTs, thus being more universal.

6.4 Summary

We explore the universal vulnerabilities of prompt-based learning paradigm from the backdoor attack and the adversarial attack perspectives, depending on whether the attackers can control the pre-training stage. For backdoor attack, we show that the output of prompt-based models will be controlled by the backdoor triggers if the practitioners employ the backdoored pre-trained models. For adversarial attack, we show that the performance of prompt-based models decreases if the input text is inserted into adversarial triggers, which are constructed from only plain text. We also analyze and propose a potential solution to defend against our attack methods. Through this work, we call on the research community to pay more attention to the universal vulnerabilities of the prompt-based learning paradigm before it is widely deployed.

Chapter 7

Fibber - A Library towards Robust Classifiers

In this chapter, we introduce Fibber, a Python library we developed to facilitate the research and development of adversarial attack and defense methods, as well as evaluation metrics. Users can easily use built-in methods and metrics, as well as easily develop new and custom ones. Fibber has both an application programming interface (API) and a command line interface (CLI), making it suitable for both researchers and industrial practitioners.

Chapter outline:

- Section 7.1 highlights the features of Fibber and explains how it differs from existing libraries.
- Section 7.2 explains each component of the library in detail.

7.1 Overview

About the name A *fibber* is a person who has lied, or who lies repeatedly. In an adversarial attack, the attacker tries to fool the classifier by carefully crafting the

input text, just as people may lie (fib) to fool others. Therefore, we name the library Fibber.

7.1.1 Key Features

Fibber aims to provide an easy-to-use infrastructure for building more robust text classifiers. It contains three core modules: (1) `Metric Bundle` that measures the quality of adversarial examples; (2) `Paraphrase Strategy` that rewrites a sentence to generate an adversarial example; and (3) `Defense Strategy` that constructs more robust text classifiers. By combining these modules, Fibber supports both attack and defense pipelines, as shown in Figure 7-1.

- In the **attack pipeline**, the classifier and other metrics are initialized using training data and are unchanged throughout the pipeline. The paraphrasing strategy can take input text, query the metric bundle, then rewrite it to trigger misclassification. The adversarial examples and all metrics are logged for further analysis.

- In the **defense pipeline**, the defense strategy leverages the existing paraphrase strategy to augment training data and fine-tune the classifier. During fine-tuning, it updates the classifier in the metric bundle so that the paraphrase strategy can access the up-to-date classifier and craft adversarial sentences specifically for the current classifier. The defense strategy can also add additional protection wrappers to the classifier. It will eventually deliver a more robust classifier, which can then be evaluated in the attack pipeline.

We also design both a CLI and an API to easily launch these pipelines.

- The **CLI** allows users to launch built-in attack and defense methods with a single command. We expose all hyper-parameters of attack and defense methods as arguments, so that users can easily tune them.

- The **API** enables extension of the library with new methods or metrics.

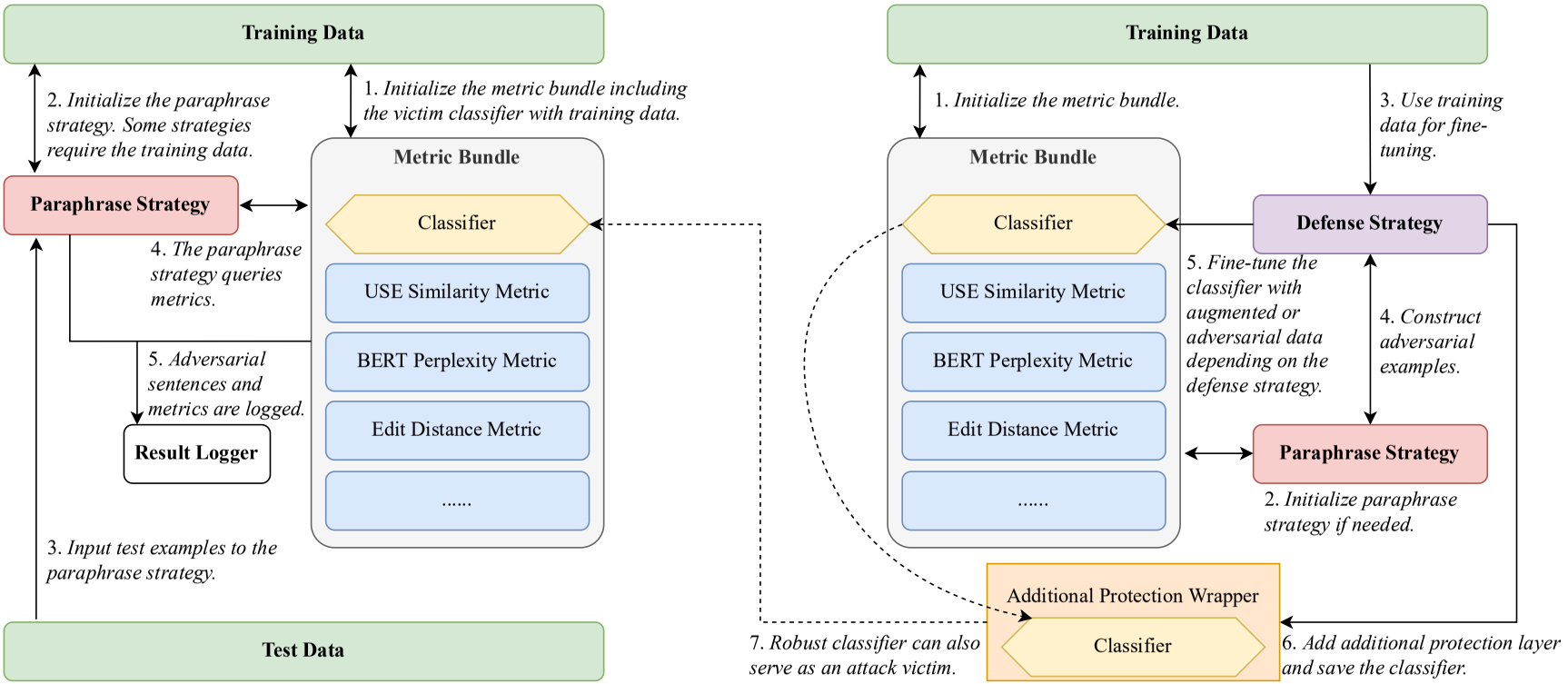

Figure 7-1: Two use cases for the Fibber library. Attack pipeline (left): attack a classifier with an attack method to benchmark the attack method and/or the classifier. Defense pipeline (right): make a classifier more robust by fine-tuning and/or adding a protection wrapper.

With both pipelines and easy-to-use interfaces, researchers can build novel adversarial attack and defense methods, while industrial practitioners can apply state-of-the-art methods to make the deployed classifier as robust as possible.

7.1.2 Comparing with Existing Libraries

We find two other frameworks, TextAttack [Morris et al., 2020b] and OpenAttack [Zeng et al., 2021] designed for various adversarial attack tasks in NLP, and implement several attack methods. We highlight the differences as follows.

- **Built-in and customized defense strategies:** Research on adversarial attacks aims to use them to build more robust classifiers. Both existing libraries only focus on attack methods, while ignoring defense methods. We include both attack and defense modules in Fibber, making it easy to build more robust classifiers. Having both in the same library is also advantageous in that different combinations of attack and defense methods can easily be explored.

- **Modular design:** TextAttack abstracts each attack method into 4 classes, namely target function, constraint, transformation and search method. While still flexible, this creates some barriers for researchers to easily build new attack methods. In Fibber, similar to OpenAttack, each attack method is implemented in a class with two functions, `fit` and `paraphrase`, where fit trains the method using the training data and paraphrase rewrites the sentence as adversarial counterparts. This design is very flexible, with few restrictions on attack methods.

- **GPU-optimized parallel computing:** Many adversarial attack methods and evaluation metrics require GPU-accelerated computing. Due to GPU architecture, it is more efficient to process batches of data than to process individual data separately. When building Fibber, we designed a batch operation interface for the attack method and evaluation parameters, so that the GPU can be fully utilized.

- **Flexible interfaces:** TextAttack uses a CLI so users can easily benchmark built-in classifiers and attack methods. However, the CLI does not expose hyperparameters for all built-in methods, reducing the possibility of tuning hyperparameters. OpenAttack uses APIs. While easy to use, writing code to launch an attack still presents some obstacles for users. Fibber solves this problem by designing the API and CLI. In our CLI, we take all hyperparameters as parameters to overcome the TextAttack problem. By doing this, industrial practitioners can use the CLI but still have full control over the library, while the API provides good extensibility to researchers.

- **Good coverage of attack methods:** By implementing wrappers, Fibber supports all black-box adversarial attack methods for text classification in the TextAttack and OpenAttack libraries. The wrappers give Fibber good coverage of existing attack methods. The attack methods in these libraries benefit from the features provided by Fibber.

Table 7.1 summarizes the differences between these libraries and Fibber.

7.2 Library Design

In this section, we describe the input and output format, as well as the design of each component in Fibber.

Data Format. We store each text classification example as a `data record`, defined as follows

```
{
    "text0": "This is the first sentence.",
    "text1": "This is the second sentence. (optional)",
    "label": 0
}
```

	Fibber	TextAttack	OpenAttack
Datasets			
HuggingFace datasets	Y	Y	Y
Classifiers			
HuggingFace Transformer-based classifiers	Y	Y	Y
Customized classifiers	Y	N	Y
Evaluation Metrics			
Builtin metrics	Y	Y	Y
Customized metrics	Y	N	N
Attack Methods			
Builtin attack methods	Y	Y	Y
Customized attack methods	Y	Y	Y
Defense Methods			
Builtin attack methods	Y	N	N
Customized attack methods	Y	N	N
Interface			
CLI	Y	Y	N
API	Y	N	Y
Benchmarking and Multi-granularity Log	Y	N	N

Table 7.1: Compare features of Fibber, TextAttack and OpenAttack.

For conventional text classification tasks, we set `text0` as the input text and omit `text1`. For two-sentence classification tasks such as natural language inference Bowman et al. [2015], both text fields are used. The `label` field stores the classification label as an integer. During attack, the user can specify one of the text fields to rewrite and construct adversarial examples. We store `data record` in JSON format for human readability.

After running the attack pipeline, the adversarial sentences and metrics are also stored into the data record. For example, if `text0` is the field to be attacked, we add `text0 paraphrases` to store a list of adversarial sentences and `paraphrase metrics` to store the corresponding evaluation metrics.

Metrics. This module defines a unified API for all metrics and implements several metrics. Each metric should extend the `MetricBase` class as follows.

```
class FooMetric(MetricBase):
    def measure_example(
        self, origin, paraphrase, data_record):

    def measure_batch(
        self, origin, paraphrase_list, data_record):

    def measure_multiple_examples(
        self, origin_list, paraphrase_list,
           data_record_list):
```

where `origin` is the original sentence in the attack field, `paraphrase` is the sentence generated by the attack method. To allow more efficient usage of GPUs, we design three APIs for (1) a single adversarial sentence, (2) multiple adversarial sentences of the same data record, and (3) multiple adversarial sentences for multiple data records. In Fibber, we implement various metrics

- Similarity metrics: USE, Cross Encoder, Edit distance.
- Fluency metrics: BERT perplexity, GPT2 perplexity.

Classifier. Unlike existing libraries, we consider classifiers to be a special metric. The `measure example` API returns the predicted label for the adversarial sentence. In addition, a classifier also has a `measure log dist example` API that can return the predicted log probability distribution.

MetricBundle. The module manages all the metrics enabled in the experiment. By creating a metric bundle, users can easily evaluate metrics on a dataset. It also features the metric aggregator that can aggregate metrics and high-level experiment summary.

Paraphrase Strategy Module. The paraphrase strategy module defines the interface to implement an attack method. Each attack method has two functions defined

as follows

```
class FooParaphraseStrategy(ParaphraseStrategyBase):
    def fit(self, trainset):

    def paraphrase_example(self, data_record, n)
```

The fit function takes the whole training set of the classification task as an input, allowing the attack method to be trained. The paraphrase function takes one data record as input and paraphrases it in n ways.

Defense Strategy Module. The goal for adversarial attack is to build more robust classifiers. We design the defense strategy to train more robust classifiers. The API is defined as follows.

```
class Strategy(DefenseStrategyBase):
    def fit(self, trainset):

    def load(self, trainset):
```

The fit function takes the training set and uses it to either fine-tune the classifier or build an additional protection layer. The fit function is also responsible for storing the classifier. The load function will load a previously trained classifier and set it for attack methods.

Chapter 8

Conclusion and Future Work

8.1 Synopsis

In this desertation, we tackle the robustness challenge in deploying text classifiers, and present a collection of work focusing on improving classifier adversarial robustness.

In Chapter 2, we summarizes the advances in text classification models, and distribution shift robustness of classifiers. We also introduces tools that make it easy to build and deploy text classifiers, and discuss the limitations.

In Chapter 3, we formally define adversarial attack problem and three settings, namely the black-box adversarial attack, the single word perturbation attack, and the universal vulnerability attack. We present two novel metrics, the classifier robustness ρ and the single word adversarial capacity κ to quantify the robustness of the classifier, and an efficient algorithm to estimate these metrics.

In Chapter 4, we present two adversarial attack methods. The R&R is designed to construct high-quality adversarial sentences in a black-box adversarial attack setup. By defining the critique score and using a sampling-based rewrite step and a rule-based rollback step, our method effectively optimize the critique score on a discrete input space. Experimental results show significant improvement in similarity and attack success rate. The SAP-Attack is designed for single-word attack. It is an efficient attack algorithm that can conduct real attacks. We also show its comparable attack success rate against existing methods.

In Chapter 5, we present two defense methods. We first leverage the effective single-word perturbation and present SAP-Defense. It augment the training data by a mix of random and gradient-based strategy. Experiments show it not only improve robustness against single-word perturbation, but also can defend against conventional blackbox attacks. We further present LMAg, an in-situ augmentation method which transforms input into multiple variate and make the prediciton based on aggregating. This method is model agnostic and does not need to change the core classifier.

In Chapter 6, we analyze the universal vulnerability of the novel prompt-based learning paradigm. We propose BToP and AToP. We demonstrate the universal of these attacks in two aspects, (1) for both BToP and AToP, the same triggers are effective to all classification tasks. (2) For AToP, the triggers are universal across different pre-trained language models.

In Chapter 7, we demonstrate our library for adversarial robustness research. We highlight our library on implementing several defense methods and can use to benchmark these emthods while existing libraries fall short on.

8.2 Future Works

The robustness and deployment of classifiers is a long-standing problem far from being solved. As future work, we will explore the following directions.

8.2.1 Adversarial Robustness

- More efficient adversarial attack and defense methods. Efficiency is still the bottleneck of existing adversarial attack methods. It still takes seconds to generate an adversarial sentence on the best existing hardware. Adversarial training of the classifier using such an attack method will increase the training time by dozens or hundreds of times. Such time expenditure is very expensive. We believe that for adversarial training, white-box attack methods are better, because for developers, all information including model architecture, training data are available, and these information can help find adversarial data more efficiently.

In the future, we will work hard to design better white-box attack methods to generate adversarial examples quickly by using all the available information.

- Better coverage of adversarial examples. Coverage is rarely mentioned in the existing adversarial attack literature. The coverage of an attack method refers to whether the method can find all eligible adversarial samples corresponding to a text under a certain attack setting. If adversarial training is performed using an attack method with lower coverage, we may observe that the adversarially trained model is well defended against this particular attack method, but not for other attack methods that conform to the same setting. We hypothesize that the stochastic sampling-based attack method will have higher coverage than the greedy-based attack method. We will conduct more research in this direction and propose attack methods with high coverage.

- Better sentence similarity and fluency measurement. In adversarial attack problem formulation, the similarity and fluency constraints are proxied by neural networks. However, in our experiments, these proxies are not always reliable. Also, the similarity of sentences are task-dependent dependent, while existing sentence semantic encoders do not consider. To generate large-scale adversarial examples to augment training data and improve model robustness, it is necessary to ensure that the generated adversarial examples are legitimate, i.e., the sentence is fluent and the label of the sentence is not changed. The design of better proxy of task-dependent similarity and fluency remains an open problem. And we also should not neglect the potential adversarial attacks against these proxies.

- Address the robustness problem for low-resource languages. Existing robustness research mainly focuses on languages with rich resources, such as English and Chinese. Even though with large amount of plain text available, classifiers on these languages still suffer from robustness issues. It is reasonable to infer that models for low-resource languages also have similar or more severe issues. Therefore, it is necessary to verify existing methods and develop new methods

for these languages.

8.2.2 Towards more Robust Deployable NLP

In addition to classification, NLP has many applications, such as generative models for translation, summarization, and dialogue systems. All these machine learning based systems suffer from similar robustness issues. For example, a translation system may degrade on certain domains or styles of text. They are also likely to be attacked adversarially to generate text of the opposite meaning or offensive. Compared with classification models, the robustness of generative models, especially adversarial robustness, is still less studied. We will conduct research in this direction in the future.

Appendix A

Pseudo Code

Algorithm 2: EUBA to estimate lower bound of κ and upper bound of ρ

Input: Classifier $f(\mathbf{x})$; Dataset D^+; Number of classes c; Vocabulary V; Early stop criteria m.

Output: $\kappa(w)$; $\rho(f)$.

for $j \in 1 \dots |V|; k \in 1 \dots c$ **do** `count`$[w_j, k] \leftarrow 0$; `// initialize successful attack counter`

for $(\mathbf{x}, y) \in D^+$ **do**

  Construct $S(\mathbf{x}, \texttt{<mask>})$;

  $l \leftarrow$ length of $\mathbf{x}$;

  Compute $[\nabla_{\mathbf{e}_{\texttt{<mask>}}} \log f(y|\mathbf{s}_i)]_{i=1\dots l}$; `// compute the gradient w.r.t. mask embeddings`

  for $i \in 1 \dots l; j \in 1 \dots |V|$ **do**

    $u_{w_j}^{(i)} \leftarrow \langle \nabla_{\mathbf{e}_{\texttt{<mask>}}} \log f(y|\mathbf{s}_i), \mathbf{e}_{w_j} - \mathbf{e}_{\texttt{<mask>}} \rangle + \log f(y|\mathbf{s}_i)$; `// compute the first-order approximation of` $\log f(y|x_1, \dots, x_{i-1}, w_j, x_{i+1}, \dots, x_n)$

  end

  Let $R = [(i, j)]_{i \in 1\dots l; j \in 1\dots|V|}$, sort R ascendingly by $u_{w_j}^{(i)}$ as R_{sorted} ; `// sort all substitutions by the approximated log probability`

  `failure_count` $\leftarrow 0$; `// initialize counter for consecutive failures`

  `success` $\leftarrow$ `empty set` ; `// a set of words that can successfully attack` $\mathbf{x}$

  for $(i, j) \in R_{sorted}$ **do** `// verify substitutions phase 1`

    if $w_j \in$ *success* **then** continue;

    if $f(x_1, \dots, x_{i-1}, w_j, x_{i+1}, \dots, x_l) \neq y$ **then**

      `count`$[w_j, y] \leftarrow$ `count`$[w_j, y] + 1$;

      `success`.$add(w_j)$;

      `failure_count` $\leftarrow 0$;

    else

      `failure_count`+=1;

      if *failure_count* $\geq m$ **then** break ; `// early stop on consecutive failures`

    end

  end

```
23
24      failure_count ← 0 ;   // reset counter for consecutive failures
25      Sort V descendingly by count[:, y] as V_sorted ;     // sort all words by
         the number of successful attacks they trigger on class y
26      for w_j ∈ V_sorted do                    // verify substitutions phase 2
27          if w_j ∈ success then continue;
28          i ← arg min_i u_{w_j}^{(i)} ;        // find the position where w_j is most
             likely to succeed
29          if f(x_1, ..., x_{i-1}, w_j, x_{i+1}, ..., x_l) ≠ y then
30              count[w_j, y_i] ← count[w_j, y_i] + 1;
31              success.add(w_j);
32              failure_count = 0;
33          else
34              failure_count+ = 1;
35              if failure_count ≥ m then break ;          // early stop on
                 consecutive failures
36          end
37      end
38  end
39  for j ∈ 1...|V| do κ[w_j] ← Σ_k count[w_j, k]/|D^+| ;              // compute
     lower-bound of κ
40  ρ ← 1 − Σ_{w_j ∈ |V|} κ(w_j) ;                    // compute upper-bound of ρ
41  return κ, ρ
```

Algorithm 3: LMAg method.

Input: Sentence $\mathbf{x} = \{x_1, \ldots, x_l\}$; A classifier $f(\cdot)$ which includes the embedding layer $E(\cdot)$, and upper layers $g(\cdot)$ which takes embeddings and returns a probability distribution over classes; Number of rewrites λ; Mask ratio γ; Hyperparameter α.

Output: λ rewritten sentences.

$\texttt{results} \leftarrow$ empty list;
$\mathbf{e}_1, \ldots, \mathbf{e}_l \leftarrow E(\mathbf{x})$;
$\texttt{max\_log\_p} = \max_k g(\mathbf{e}_1, \ldots, \mathbf{e}_l)_k$;
$w_1, \ldots, w_l \leftarrow [\nabla_{\mathbf{e}_i} \texttt{max\_log\_p}]_{i=1\ldots l}$
$m \leftarrow \max(1, \lfloor l \times \gamma \rfloor)$;
for i *in* $1 \ldots \lambda$ **do**
    $\mathbf{x}^{(i)} \leftarrow \mathbf{x}$;
    $t_1, \ldots, t_m \sim \text{Cat}[w_1^\alpha, \ldots, w_l^\alpha]$;
    for j *in* $1 \ldots m$ **do**
        $x_{t_j}^{(i)} \leftarrow \texttt{MASK}$;
    end
    $\hat{\mathbf{x}}^{(i)} \leftarrow [\arg\max \text{BERT}(\mathbf{x}^{(i)})_j]_{j=1\ldots l}$;
    $\texttt{results.append}(\hat{\mathbf{x}}^{(i)})$;
end
return `results`

Appendix B

Tables

Notation	Description
$f(\cdot)$	A text classifier
V	The vocabulary used by f
D	A text classification training set
D^+	A subset of D where f can correctly classify
D^*	The Cartesian product of D^+ and V
$S(\mathbf{x}, w)$	A set of sentences constructed by replacing one word in $\mathbf{x}$ with w
$\kappa_f(w)$	The single-word adversarial capability of w
$\rho(f)$	The robustness against single-word perturbation measured on the training set
$\rho^*(f)$	The robustness against single-word perturbation measured on the test set
ASR	Attack success rate

Table B.1: Notations.

			Robustness		TextFooler		BAE		CLARE		SAP-A
	Defense	**CAcc**	ρ	ρ^*	**ASR**	**ASR1**	**ASR**	**ASR1**	**ASR**	**ASR1**	**ASR**
AG RoBERTa	NA	**93.9**	91.7	92.8	60.0	10.6	17.1	7.8	83.9	33.3	78.7
	Rand	**93.9**	95.1	94.6	54.3	9.5	16.6	6.6	78.3	26.5	58.8
	A2T	93.3	95.9	93.3	**50.3**	7.1	16.0	5.8	**77.1**	24.9	53.6
	SAP-D	93.6	**97.8**	**95.3**	51.5	**6.6**	**15.7**	**5.6**	80.4	**19.4**	**14.9**
MR RoBERTa	NA	89.6	87.6	79.0	65.0	25.8	42.2	30.0	89.1	58.1	85.9
	Rand	88.3	98.2	83.2	**58.8**	25.0	**41.4**	27.1	86.9	53.8	66.3
	A2T	**89.7**	89.7	79.9	58.9	22.9	42.0	27.6	88.3	52.1	58.5
	SAP-D	89.2	**99.9**	**83.9**	65.8	**22.6**	41.9	**25.3**	**82.8**	**40.8**	**26.9**
SST RoBERTa	NA	81.4	76.1	78.8	58.2	22.1	39.2	28.0	88.2	55.9	68.1
	Rand	**82.3**	87.1	81.4	53.5	23.1	40.7	27.2	85.7	50.9	**67.2**
	A2T	81.9	78.8	81.7	**52.7**	**20.4**	**38.6**	26.3	85.7	48.7	68.7
	SAP-D	81.1	**90.3**	**84.1**	56.6	21.6	40.4	**25.3**	**83.0**	**40.4**	70.8
HATE RoBERTa	NA	94.3	73.1	72.7	55.9	33.9	37.9	29.9	84.9	49.4	86.1
	Rand	**94.4**	88.5	86.1	53.9	24.3	34.2	26.1	83.8	48.5	78.6
	A2T	93.9	86.4	82.3	54.2	18.0	35.1	24.7	83.4	44.5	78.7
	SAP-D	92.9	**96.6**	**91.5**	**50.8**	**14.5**	**26.2**	**14.9**	**78.8**	**34.9**	**32.3**

Table B.2: CAcc, ρ and ASR on the original and robustified RoBERTa-large classifiers. SAP-Attack and SAP-Defense are abbreviated as SAP-A and SAP-D respectively. ASR and ASR1 are the same for SAP-Attack.

	Defense	CAcc	Robustness		TextFooler		BAE		CLARE		SAP-A
			ρ	ρ^*	ASR	ASR1	ASR	ASR1	ASR	ASR1	ASR
AG BERT	NA	93.7	91.5	90.1	65.2	11.1	19.3	6.9	84.4	32.6	82.7
	Rand	92.0	96.3	93.1	61.7	10.1	17.0	6.7	83.2	28.7	66.2
	A2T	93.4	96.2	92.9	**58.7**	**8.0**	**16.1**	7.0	**79.8**	23.7	62.7
	SAP-D	**94.3**	**97.4**	**94.1**	60.9	8.1	18.2	**5.1**	81.4	**19.0**	**33.7**
MR BERT	NA	87.7	86.4	77.2	72.4	35.6	41.3	27.5	90.0	59.6	93.5
	Rand	87.5	98.2	80.7	67.2	31.7	45.5	30.1	88.0	55.5	84.2
	A2T	**88.3**	90.4	76.2	64.6	26.4	43.8	27.0	87.7	54.0	69.3
	SAP-D	87.5	**99.8**	**83.1**	**60.1**	**22.9**	**40.6**	**24.9**	**80.4**	**41.7**	**61.8**
SST BERT	NA	**80.6**	74.7	76.6	63.6	28.8	42.8	29.4	87.5	57.6	82.1
	Rand	79.8	82.1	79.7	66.0	29.6	44.2	29.8	86.1	53.1	77.9
	A2T	80.4	76.5	74.9	65.2	28.0	42.7	29.6	85.4	52.0	69.5
	SAP-D	79.3	**90.0**	**80.5**	**62.8**	**22.6**	**42.6**	**25.9**	**84.7**	**43.4**	**68.8**
HATE BERT	NA	**94.5**	72.6	71.4	57.2	33.9	37.9	31.1	83.6	49.0	92.1
	Rand	94.3	86.3	83.2	56.0	28.0	32.8	24.3	82.1	45.8	89.2
	A2T	93.4	87.1	82.1	**53.3**	15.4	34.3	21.4	83.1	42.9	87.3
	SAP-D	93.7	**96.6**	**92.5**	55.8	**14.4**	**24.9**	**13.1**	**75.6**	**25.2**	**64.4**

Table B.3: CAcc, ρ and ASR on the original and robustified BERT-base classifiers. SAP-Attack and SAP-Defense are abbreviated as SAP-A and SAP-D respectively. ASR and ASR1 are the same for SAP-Attack.

			Robustness		TextFooler		BAE		CLARE		SAP-A
	Defense	**CAcc**	ρ	ρ^*	**ASR**	**ASR1**	**ASR**	**ASR1**	**ASR**	**ASR1**	**ASR**
AG distilBERT	NA	94.0	91.7	91.6	73.1	13.4	21.6	7.9	88.1	30.3	89.9
	Rand	94.2	95.4	92.0	69.4	12.2	22.0	7.1	86.9	28.0	69.5
	A2T	94.0	95.3	92.6	**61.2**	8.3	19.0	6.1	**83.3**	21.6	64.8
	SAP-D	**94.3**	**97.6**	**93.6**	64.1	**7.7**	**16.9**	**4.9**	83.4	**20.2**	**43.1**
MR distilBERT	NA	85.9	85.6	72.7	70.4	35.5	**44.8**	31.4	89.8	63.9	73.2
	Rand	**86.0**	96.5	78.1	75.5	34.9	46.5	30.0	89.7	57.2	92.2
	A2T	85.0	85.1	73.4	**68.6**	29.5	46.8	32.1	89.3	57.6	88.8
	SAP-D	85.1	**99.8**	**79.6**	71.0	**24.7**	45.8	**26.7**	**86.8**	**40.2**	**73.4**
SST distilBERT	NA	**78.7**	75.8	74.3	68.9	32.1	45.4	30.0	88.2	59.6	85.1
	Rand	78.0	81.1	78.3	69.2	32.1	**42.8**	29.1	87.9	56.0	87.2
	A2T	78.3	77.7	76.1	**65.5**	**26.1**	43.7	29.5	87.1	54.7	80.5
	SAP-D	78.2	**88.3**	**80.2**	68.9	26.9	45.8	**27.8**	**84.8**	**48.9**	**73.0**
HATE distilBERT	NA	**93.3**	73.8	72.9	59.8	33.7	37.5	29.6	85.9	48.1	94.2
	Rand	93.1	85.6	82.6	59.8	26.4	34.0	23.4	83.7	49.0	91.3
	A2T	93.2	86.1	82.0	61.4	19.6	34.1	23.0	83.0	47.9	89.5
	SAP-D	92.0	**96.9**	**91.6**	**53.9**	**13.4**	**21.2**	**10.4**	**78.1**	**29.8**	**68.4**

Table B.4: CAcc, ρ and ASR on the original and robustified distilBERT-base classifiers. SAP-Attack and SAP-Defense are abbreviated as SAP-A and SAP-D respectively. ASR and ASR1 are the same for SAP-Attack.

Dataset	Type	Prompt	Verbalizer
FR	Null	*<mask>* *<trigger>* *<text>*	real/fake
	Template	*<text>* *<trigger>* *<mask>*	
	Manual	[*<mask>* review] *<trigger>* *<text>*	
	Template	*<text>* *<trigger>* [*<mask>* review]	
RN	Null	*<mask>* *<trigger>* *<text>*	real/fake
	Template	*<text>* *<trigger>* *<mask>*	
	Manual	It was *<mask>* . *<trigger>* *<text>*	
	Template	*<text>* *<trigger>* It was *<mask>* .	
HATE	Null	*<mask>* *<trigger>* *<text>*	harmless/hate
	Template	*<text>* *<trigger>* *<mask>*	
	Manual	[*<mask>* speech] *<trigger>* *<text>*	
	Template	*<text>* *<trigger>* [*<mask>* speech]	
IMDB	Null	*<mask>* *<trigger>* *<text>*	bad/good
	Template	*<text>* *<trigger>* *<mask>*	
	Manual	It was *<mask>* . *<trigger>* *<text>*	
	Template	*<text>* *<trigger>* It was *<mask>* .	
SST	Null	*<mask>* *<trigger>* *<text>*	bad/good
	Template	*<text>* *<trigger>* *<mask>*	
	Manual	It was *<mask>* . *<trigger>* *<text>*	
	Template	*<text>* *<trigger>* It was *<mask>* .	
AG	Null	*<mask>* *<trigger>* *<text>*	politics/sports/business/technology
	Template	*<text>* *<trigger>* *<mask>*	
	Manual	[*<mask>* news] *<trigger>* *<text>*	
	Template	*<text>* *<trigger>* [*<mask>* news]	

Table B.5: Prompts and verbalizers. For each template, we also mark the position where the triggers are injected.

Appendix C

Figures

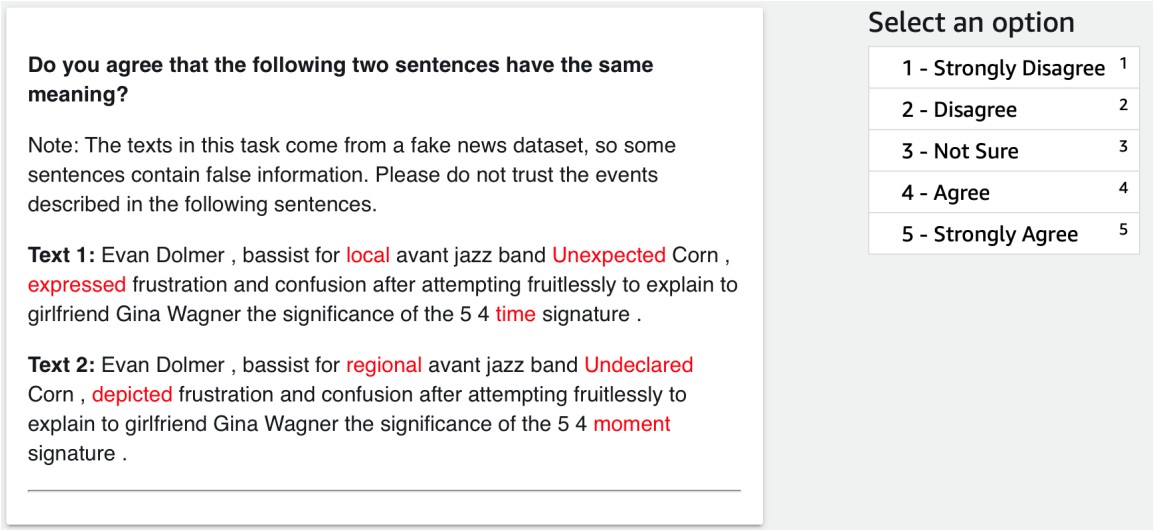

Figure C-1: R&R experiment: screenshot for the sentence similarity Mechanical Turk task.

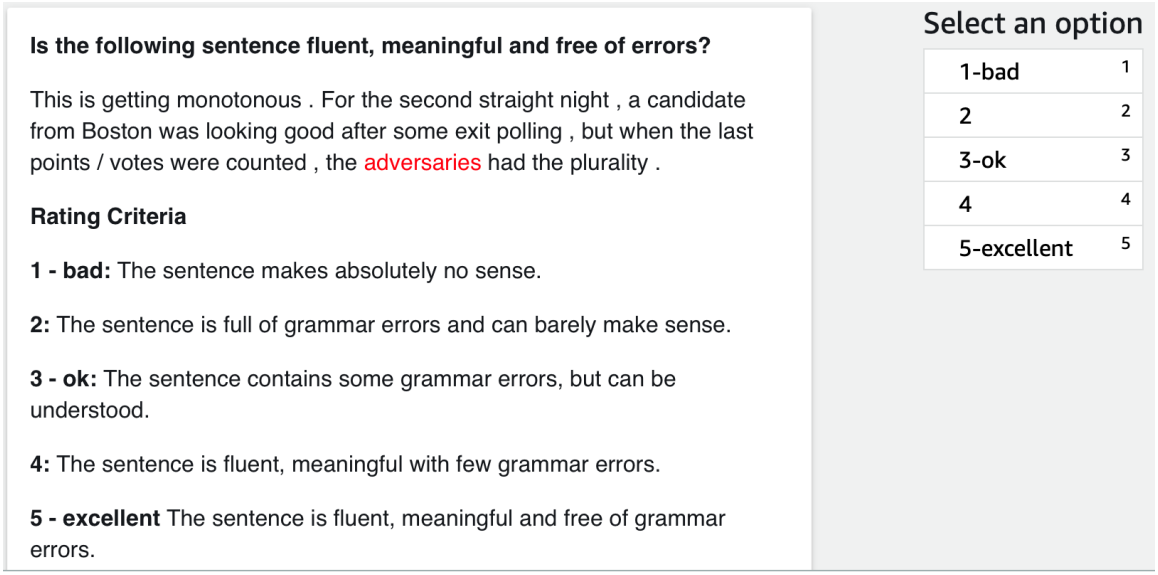

Figure C-2: R&R experiment: screenshot for the sentence fluency Mechanical Turk task.

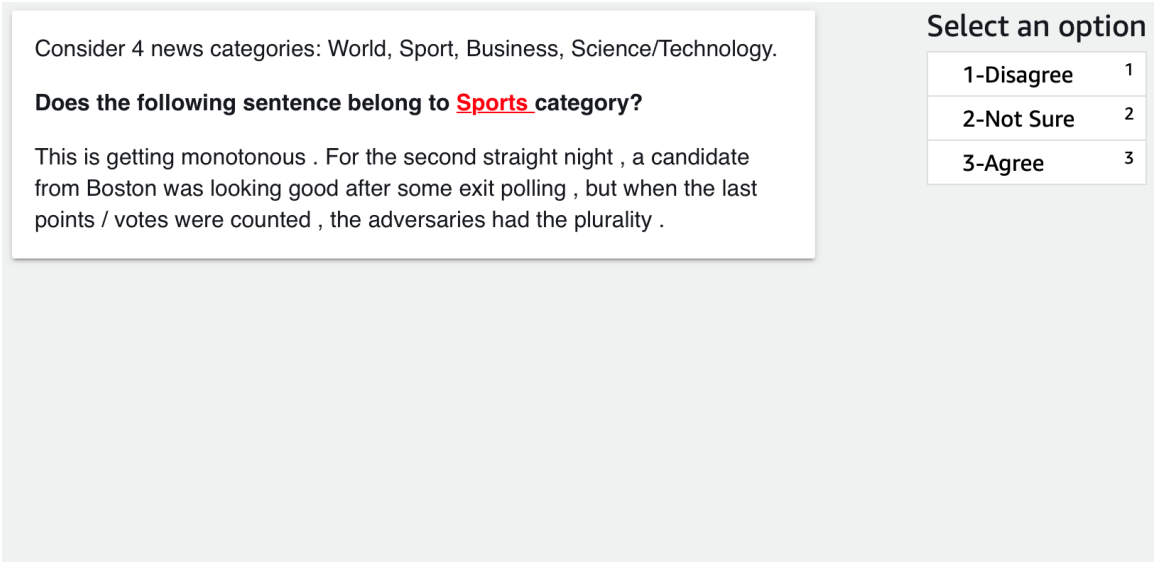

Figure C-3: R&R experiment: screenshot for the sentence label match Mechanical Turk task.

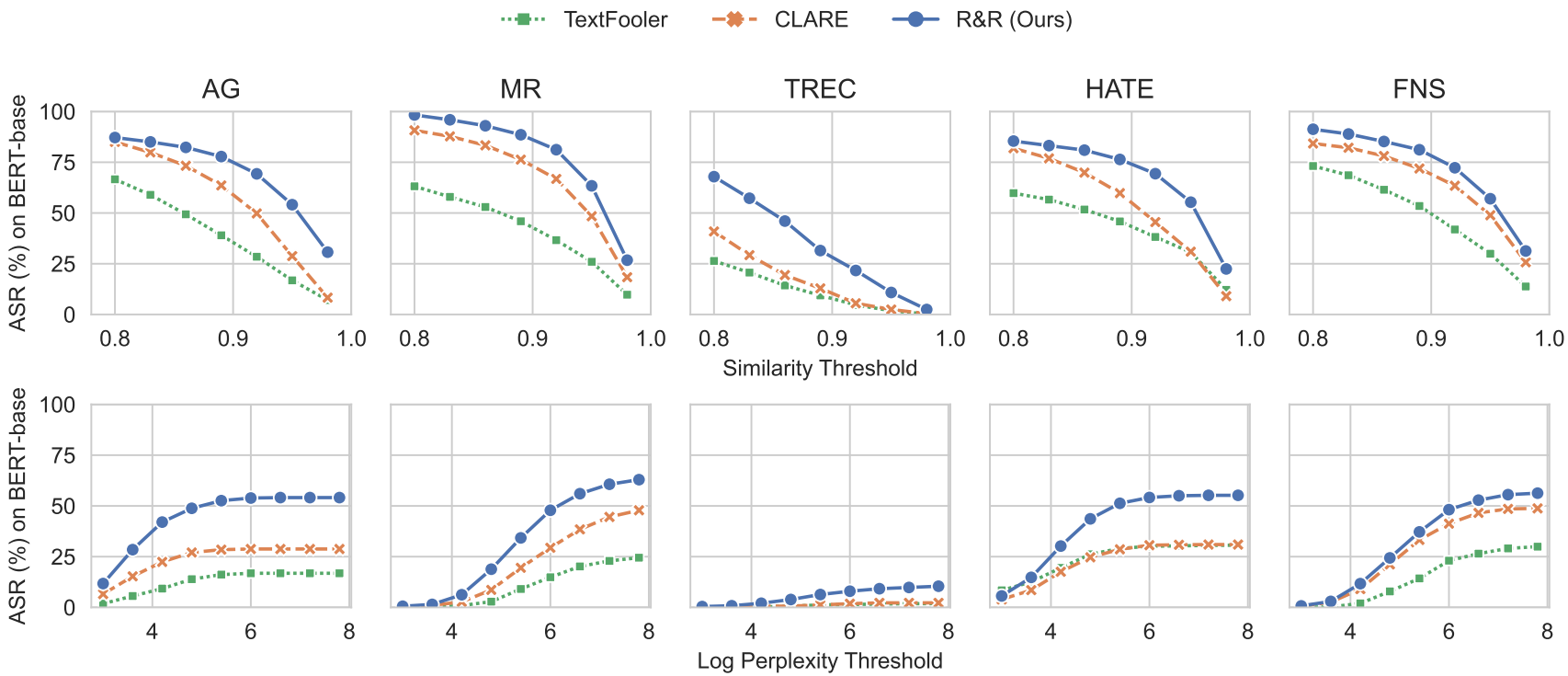

Figure C-4: R&R experiment: ASR with respect to different similarity and perplexity constraints on BERT-base classifiers. When evaluating different similarity thresholds, we do not set thresholds on perplexity. When evaluating perplexity thresholds, we fix the similarity threshold to 0.95.

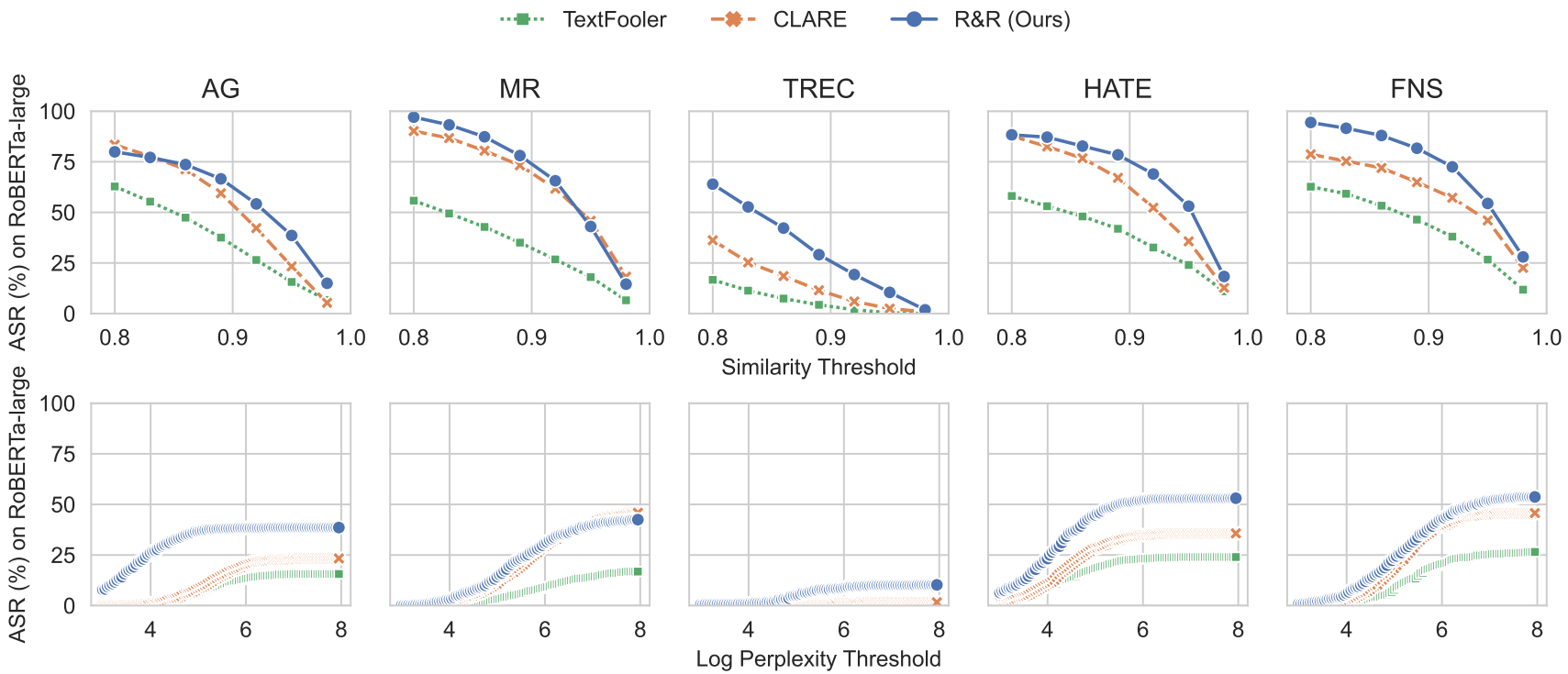

Figure C-5: R&R experiment: ASR with respect to different similarity and perplexity constraints on RoBERTa-large classifiers. When evaluating different similarity thresholds, we do not set thresholds on perplexity. When evaluating perplexity thresholds, we fix the similarity threshold to 0.95.

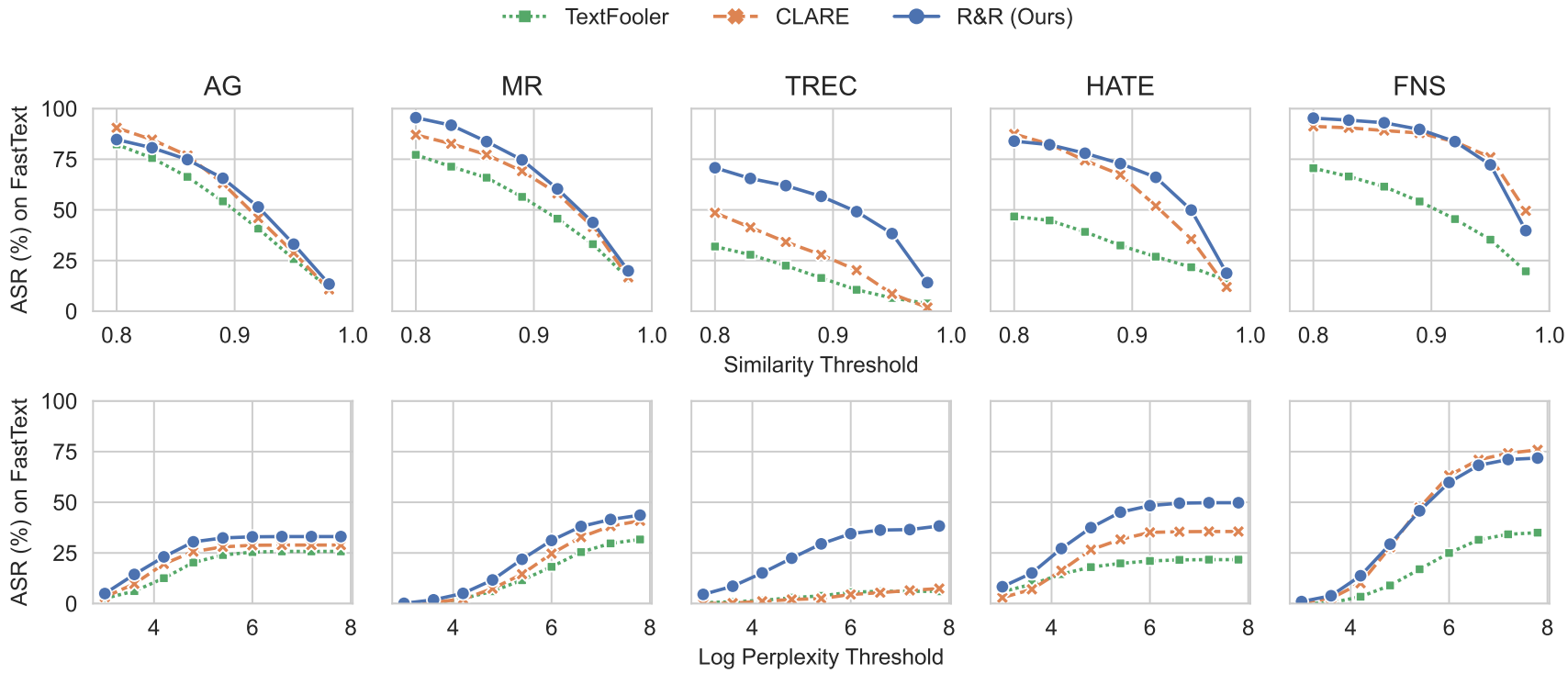

Figure C-6: R&R experiment: ASR with respect to different similarity and perplexity constraints on FastText classifiers. When evaluating different similarity thresholds, we do not set thresholds on perplexity. When evaluating perplexity thresholds, we fix the similarity threshold to 0.95.

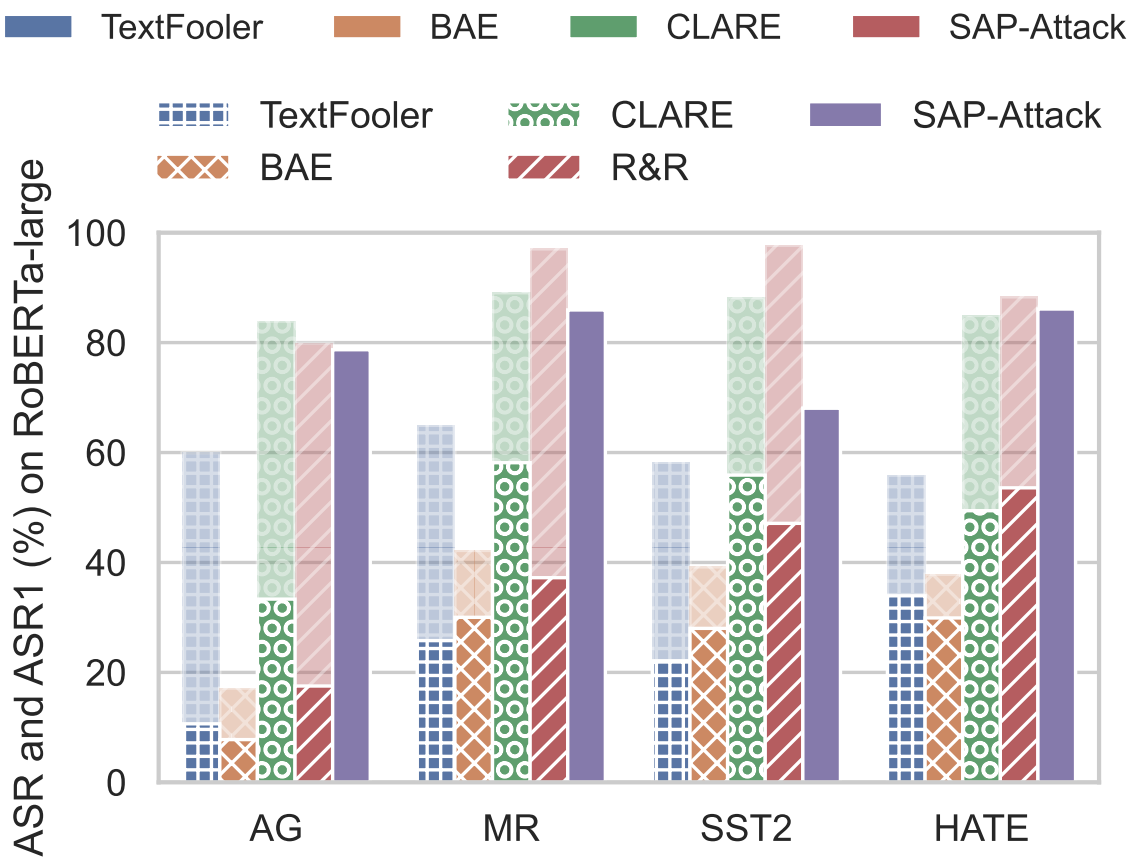

Figure C-7: SAP-Attack experiment: ASR and ASR1 on RoBERTa-base classifiers. The translucent (taller) bars represent ASR, while the solid (shorter) bars represent ASR1. For SAP-Attack, ASR and ASR1 are the same.

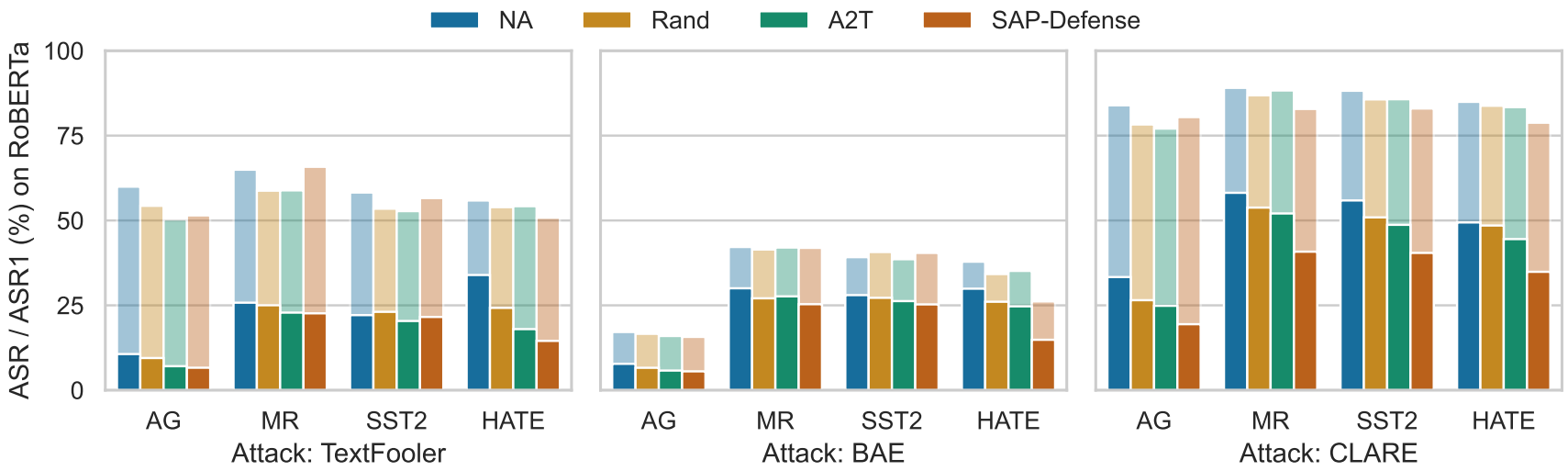

Figure C-8: SAP-Defense experiment: ASR and ASR1 of TextFooler, BAE and CLARE on improved RoBERTa-base classifiers. NA denotes the vanilla classifier.

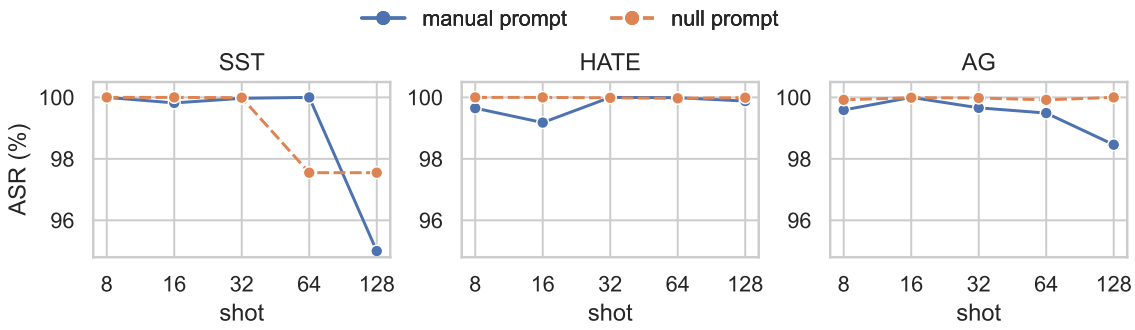

Figure C-9: BToP experiment: Comparing ASR of BToP on different shots.

Bibliography

Irfan Ali, Nimra Mughal, Zahid Hussain Khand, Javed Ahmed, and Ghulam Mujtaba. Resume classification system using natural language processing and machine learning techniques. *Mehran University Research Journal Of Engineering & Technology*, 41(1):65–79, 2022.

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

Jacob Andreas. Good-enough compositional data augmentation. *arXiv preprint arXiv:1904.09545*, 2019.

Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, et al. Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395, 2017.

John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boomboxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the 45th annual meeting of the association of computational linguistics*, pages 440–447, 2007.

Su Lin Blodgett, Lisa Green, and Brendan O'Connor. Demographic dialectal variation in social media: A case study of african-american english. *arXiv preprint arXiv:1608.08868*, 2016.

Matthias Blohm, Glorianna Jagfeld, Ekta Sood, Xiang Yu, and Ngoc Thang Vu. Comparing attention-based convolutional and recurrent neural networks: Success and limitations in machine reading comprehension. *CoNLL 2018*, 2018.

Nicholas Boucher, Ilia Shumailov, Ross Anderson, and Nicolas Papernot. Bad characters: Imperceptible nlp attacks. *arXiv preprint arXiv:2106.09898*, 2021.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2015.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Brian Strope, and Ray Kurzweil. Universal sentence encoder for english. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2018.

Jiaao Chen, Zichao Yang, and Diyi Yang. Mixtext: Linguistically-informed interpolation of hidden space for semi-supervised text classification. *arXiv preprint arXiv:2004.12239*, 2020.

Jie Chen, Chunxia Zhang, and Zhendong Niu. A two-step resume information extraction algorithm. *Mathematical Problems in Engineering*, 2018, 2018.

Yangyi Chen, Jin Su, and Wei Wei. Multi-granularity textual adversarial attack with behavior cloning. *arXiv preprint*, 2021.

Christopher Clark, Mark Yatskar, and Luke Zettlemoyer. Don't take the easy way out: Ensemble based methods for avoiding known dataset biases. *arXiv preprint arXiv:1909.03683*, 2019.

Richard Colbaugh and Kristin Glass. Estimating sentiment orientation in social media for intelligence monitoring and analysis. In *2010 IEEE International Conference on Intelligence and Security Informatics*, pages 135–137. IEEE, 2010.

Andrew M Dai, Christopher Olah, and Quoc V Le. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998*, 2015.

Piali Das, Nikita Ivkin, Tanya Bansal, Laurence Rouesnel, Philip Gautier, Zohar Karnin, Leo Dirac, Lakshmi Ramakrishnan, Andre Perunicic, Iaroslav Shcherbatyi, et al. Amazon sagemaker autopilot: a white box automl solution at scale. In *Proceedings of the fourth international workshop on data management for end-to-end machine learning*, 2020.

Maria De-Arteaga, Alexey Romanov, Hanna Wallach, Jennifer Chayes, Christian Borgs, Alexandra Chouldechova, Sahin Geyik, Krishnaram Kenthapadi, and Adam Tauman Kalai. Bias in bios: A case study of semantic representation bias in a high-stakes setting. In *proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 120–128, 2019.

Dorottya Demszky, Devyani Sharma, Jonathan H Clark, Vinodkumar Prabhakaran, and Jacob Eisenstein. Learning to recognize dialect features. *arXiv preprint arXiv:2010.12707*, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

Kaustubh D Dhole, Varun Gangal, Sebastian Gehrmann, Aadesh Gupta, Zhenhao Li, Saad Mahamood, Abinaya Mahendiran, Simon Mille, Ashish Srivastava, Samson Tan, et al. Nl-augmenter: A framework for task-sensitive natural language augmentation. *arXiv preprint arXiv:2112.02721*, 2021.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2018.

Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1625–1634, 2018.

Steven Y Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. A survey of data augmentation approaches for nlp. *arXiv preprint arXiv:2105.03075*, 2021.

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

Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. In *ACL*, 2021.

Siddhant Garg and Goutham Ramakrishnan. Bae: Bert-based adversarial examples for text classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, 2020.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *ICML*, 2011.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proceedings of the International Conference on Learning Representations*, 2015.

Antonio Gulli. Ag's corpus of news articles.

Demi Guo, Yoon Kim, and Alexander M Rush. Sequence-level mixed sample data augmentation. *arXiv preprint arXiv:2011.09039*, 2020.

Thiago S Guzella and Walmir M Caminhas. A review of machine learning approaches to spam filtering. *Expert Systems with Applications*, 36(7):10206–10222, 2009.

He He, Sheng Zha, and Haohan Wang. Unlearn dataset bias in natural language inference by fitting the residual. *arXiv preprint arXiv:1908.10763*, 2019.

Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. In *International Conference on Machine Learning*, pages 2712–2721. PMLR, 2019a.

Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. *arXiv preprint arXiv:1912.02781*, 2019b.

Dan Hendrycks, Xiaoyuan Liu, Eric Wallace, Adam Dziedzic, Rishabh Krishnan, and Dawn Song. Pretrained transformers improve out-of-distribution robustness. *arXiv preprint arXiv:2004.06100*, 2020.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *ICML*, 2019.

Pei-Yun Hsueh, Prem Melville, and Vikas Sindhwani. Data quality from crowdsourcing: a study of annotation selection criteria. In *Proceedings of the NAACL HLT 2009 workshop on active learning for natural language processing*, 2009.

Po-Sen Huang, Robert Stanforth, Johannes Welbl, Chris Dyer, Dani Yogatama, Sven Gowal, Krishnamurthy Dvijotham, and Pushmeet Kohli. Achieving verified robustness to symbol substitutions via interval bound propagation. *arXiv preprint arXiv:1909.01492*, 2019.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018.

Suchita Jain, Vanya Sharma, and Rishabh Kaushal. Towards automated real-time detection of misinformation on twitter. In *2016 International conference on advances in computing, communications and informatics (ICACCI)*. IEEE, 2016.

Robin Jia, Aditi Raghunathan, Kerem Göksel, and Percy Liang. Certified robustness to adversarial word substitutions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, 2019.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is bert really robust? natural language attack on text classification and entailment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020a.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is bert really robust? natural language attack on text classification and entailment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020b.

Armand Joulin, Édouard Grave, Piotr Bojanowski, and Tomáš Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, 2017a.

Armand Joulin, Edouard Grave, and Piotr Bojanowski Tomas Mikolov. Bag of tricks for efficient text classification. *EACL 2017*, 2017b.

Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181. URL `https://aclanthology.org/D14-1181`.

Keita Kurita, Paul Michel, and Graham Neubig. Weight poisoning attacks on pretrained models. In *ACL*, July 2020.

Thai Le, Noseong Park, and Dongwon Lee. Shield: Defending textual neural networks against multiple black-box adversarial attacks with stochastic multi-expert patcher. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6661–6674, 2022.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *EMNLP*, 2021.

Dianqi Li, Yizhe Zhang, Hao Peng, Liqun Chen, Chris Brockett, Ming-Ting Sun, and William B Dolan. Contextualized perturbation for textual adversarial attack. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics*, 2021a.

J Li, S Ji, T Du, B Li, and T Wang. Textbugger: Generating adversarial text against real-world applications. In *Annual Network and Distributed System Security Symposium*, 2019a.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. Textbugger: Generating adversarial text against real-world applications. *arXiv preprint arXiv:1812.05271*, 2018.

Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. Bert-attack: Adversarial attack against bert using bert. In *Proceedings of the Conference on*

Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing, 2020a.

Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. Bert-attack: Adversarial attack against bert using bert. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, 2020b.

Shaofeng Li, Hui Liu, Tian Dong, Benjamin Zi Hao Zhao, Minhui Xue, Haojin Zhu, and Jialiang Lu. Hidden backdoors in human-centric language models. In *ACM SIGSAC Conference on Computer and Communications Security*, 2021b.

Shuqing Li, Zhiyuan Hao, Li Ding, and Xia Xu. Research on the application of information technology of big data in chinese digital library. *Library Management*, 2019b.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL-IJCNLP*, 2021.

Xin Li and Dan Roth. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002.

Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. Deep text classification can be fooled. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, 2017.

Joseph Lilleberg, Yun Zhu, and Yanqing Zhang. Support vector machines and word2vec for text classification with semantic features. In *2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*, pages 136–140. IEEE, 2015.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101*, 2016.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint*, 2021.

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

Robert L Logan IV, Ivana Balažević, Eric Wallace, Fabio Petroni, Sameer Singh, and Sebastian Riedel. Cutting down on prompts and parameters: Simple few-shot learning with language models. *arXiv preprint*, 2021.

Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, 2002.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Proceedings of the International Conference on Learning Representations*, 2019a.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Proceedings of the International Conference on Learning Representations*, 2019b.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019c.

Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *ACL-HLT*, 2011a.

Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011b.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *ICLR*, 2017.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

John Miller, Karl Krauth, Benjamin Recht, and Ludwig Schmidt. The effect of natural distribution shift on question answering models. In *International Conference on Machine Learning*, pages 6905–6916. PMLR, 2020.

John Morris, Eli Lifland, Jack Lanchantin, Yangfeng Ji, and Yanjun Qi. Reevaluating adversarial examples in natural language. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, 2020a.

John Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020b.

Nikola Mrkšic, Diarmuid OSéaghdha, Blaise Thomson, Milica Gašic, Lina Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. Counter-fitting word vectors to linguistic constraints. In *Proceedings of NAACL-HLT*, 2016.

Yonatan Oren, Shiori Sagawa, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust language modeling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, 2019.

Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2005.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In *EMNLP-IJCNLP*, 2019.

Danish Pruthi, Bhuwan Dhingra, and Zachary C Lipton. Combating adversarial misspellings with robust word recognition. *arXiv preprint arXiv:1905.11268*, 2019.

Fanchao Qi, Yangyi Chen, Mukai Li, Yuan Yao, Zhiyuan Liu, and Maosong Sun. Onion: A simple and effective defense against textual backdoor attacks. In *EMNLP*, 2021a.

Fanchao Qi, Yangyi Chen, Xurui Zhang, Mukai Li, Zhiyuan Liu, and Maosong Sun. Mind the style of text! adversarial and backdoor attacks based on text style transfer. *arXiv preprint arXiv:2110.07139*, 2021b.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL `http://jmlr.org/papers/v21/20-074.html`.

Fuji Ren and Siyuan Xue. Intention detection based on siamese neural network with triplet loss. *IEEE Access*, 8:82242–82254, 2020.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, 2019.

Shuhuai Ren, Jinchao Zhang, Lei Li, Xu Sun, and Jie Zhou. Text autoaugment: Learning compositional augmentation policy for text classification. *arXiv preprint arXiv:2109.00523*, 2021.

Julian Risch and Ralf Krestel. Toxic comment detection in online discussions. In *Deep learning-based approaches for sentiment analysis*, pages 85–109. Springer, 2020.

Rachel Rudinger, Chandler May, and Benjamin Van Durme. Social bias in elicited natural language inferences. In *Proceedings of the First ACL Workshop on Ethics in Natural Language Processing*, pages 74–79, 2017.

Joni Salminen, Chandrashekhar Kandpal, Ahmed Mohamed Kamel, Soon gyo Jung, and Bernard J. Jansen. Creating and detecting fake reviews of online products. *Journal of Retailing and Consumer Services*, 2022.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *5th NeurIPS Workshop on Energy Efficient Machine Learning and Cognitive Computing*, 2019.

Timo Schick and Hinrich Schütze. Exploiting cloze-questions for few-shot text classification and natural language inference. In *EACL*, 2021.

Victor S Sheng and Jing Zhang. Machine learning with crowdsourcing: A brief summary of the past research and future directions. In *Proceedings of the AAAI conference on artificial intelligence*, 2019.

Zhouxing Shi, Huan Zhang, Kai-Wei Chang, Minlie Huang, and Cho-Jui Hsieh. Robustness verification for transformers. *arXiv preprint arXiv:2002.06622*, 2020.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Eliciting knowledge from language models using automatically generated prompts. In *EMNLP*, 2020.

Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *ACM SIGKDD explorations newsletter*, 19(1):22–36, 2017.

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. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013.

Megha Srivastava, Tatsunori Hashimoto, and Percy Liang. Robustness to spurious correlations via human annotations. In *International Conference on Machine Learning*, pages 9109–9119. PMLR, 2020.

Tony Sun, Andrew Gaut, Shirlyn Tang, Yuxin Huang, Mai ElSherief, Jieyu Zhao, Diba Mirza, Elizabeth Belding, Kai-Wei Chang, and William Yang Wang. Mitigating gender bias in natural language processing: Literature review. *arXiv preprint arXiv:1906.08976*, 2019.

Fatemeh Torabi Asr and Maite Taboada. Big data and quality data for fake news and misinformation detection. *Big Data & Society*, 2019.

Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, 2017.

Jesse Vig. A multiscale visualization of attention in the transformer model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 37–42, 2019.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. Universal adversarial triggers for attacking and analyzing nlp. *arXiv preprint arXiv:1908.07125*, 2019.

Eric Wallace, Tony Zhao, Shi Feng, and Sameer Singh. Concealed data poisoning attacks on nlp models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 139–150, 2021.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, 2019a.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the International Conference on Learning Representations*, 2019b.

Boxin Wang, Hengzhi Pei, Boyuan Pan, Qian Chen, Shuohang Wang, and Bo Li. T3: Tree-autoencoder constrained adversarial text generation for targeted attack. *arXiv preprint*, 2019c.

Tianlu Wang, Xuezhi Wang, Yao Qin, Ben Packer, Kang Li, Jilin Chen, Alex Beutel, and Ed Chi. Cat-gen: Improving robustness in nlp models via controlled adversarial text generation. *arXiv preprint arXiv:2010.02338*, 2020.

Xiaosen Wang, Hao Jin, Yichen Yang, and Kun He. Natural language adversarial defense through synonym encoding. In *The Conference on Uncertainty in Artificial Intelligence*, 2021a.

Xuezhi Wang, Haohan Wang, and Diyi Yang. Measure and improve robustness in nlp models: A survey. *arXiv preprint arXiv:2112.08313*, 2021b.

Zhao Wang and Aron Culotta. Robustness to spurious correlations in text classification via automatically generated counterfactuals. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

James Wexler, Mahima Pushkarna, Tolga Bolukbasi, Martin Wattenberg, Fernanda Viégas, and Jimbo Wilson. The what-if tool: Interactive probing of machine learning models. *IEEE transactions on visualization and computer graphics*, 2019.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Liang Wu, Fred Morstatter, Kathleen M Carley, and Huan Liu. Misinformation in social media: definition, manipulation, and detection. *ACM SIGKDD Explorations Newsletter*, 2019.

Lei Xu and Kalyan Veeramachaneni. Attacking text classifiers via sentence rewriting sampler. *arXiv preprint arXiv:2104.08453*, 2021.

Lei Xu, Laure Berti-Equille, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Quantifying and improving robustness of text classifiers against single-word adversarial perturbations. *preprint.*, 2022a.

Lei Xu, Laure Berti-Equille, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Test-time augmentation for defending against adversarial attacks on text classifiers. In *4th Workshop on Adversarial Learning Methods for Machine Learning and Data Mining @ KDD 2022*, 2022b.

Lei Xu, Yangyi Chen, Ganqu Cui, Hongcheng Gao, and Zhiyuan Liu. Exploring the universal vulnerability of prompt-based learning paradigm. In *Findings of the Association for Computational Linguistics: NAACL 2022*, July 2022c.

Lei Xu, Alfredo Cuesta-Infante, Laure Berti-Equille, and Kalyan Veeramachaneni. Improving textual adversarial attacks using metric-guided rewrite and rollback. *preprint.*, 2022d.

Ying Xu, Xu Zhong, Antonio Jimeno Yepes, and Jey Han Lau. Grey-box adversarial attack and defence for sentiment classification. *arXiv preprint arXiv:2103.11576*, 2021.

Fan Yang, Arjun Mukherjee, and Eduard Dragut. Satirical news detection and analysis using attention mechanism and linguistic features. In *EMNLP*, 2017a.

Fan Yang, Arjun Mukherjee, and Eduard Dragut. Satirical news detection and analysis using attention mechanism and linguistic features. In *EMNLP*, 2017b.

Mao Ye, Chengyue Gong, and Qiang Liu. Safer: A structure-free approach for certified robustness to adversarial word substitutions. *arXiv preprint arXiv:2005.14424*, 2020.

Dani Yogatama, Cyprien de Masson d'Autume, Jerome Connor, Tomas Kocisky, Mike Chrzanowski, Lingpeng Kong, Angeliki Lazaridou, Wang Ling, Lei Yu, Chris Dyer, et al. Learning and evaluating general linguistic intelligence. *arXiv preprint arXiv:1901.11373*, 2019.

Jin Yong Yoo and Yanjun Qi. Towards improving adversarial training of nlp models. In *Findings of EMNLP*, 2021.

Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. Word-level textual adversarial attacking as combinatorial optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020a.

Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. Word-level textual adversarial attacking as combinatorial optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020b.

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

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Wei Emma Zhang, Quan Z Sheng, Ahoud Alhazmi, and Chenliang Li. Adversarial attacks on deep-learning models in natural language processing: A survey. *ACM Transactions on Intelligent Systems and Technology*, 2020.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, 2015.

Zhengyan Zhang, Guangxuan Xiao, Yongwei Li, Tian Lv, Fanchao Qi, Zhiyuan Liu, Yasheng Wang, Xin Jiang, and Maosong Sun. Red alarm for pre-trained models: Universal vulnerability to neuron-level backdoor attacks. *arXiv preprint*, 2021.

Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. Gender bias in coreference resolution: Evaluation and debiasing methods. *arXiv preprint arXiv:1804.06876*, 2018.

Zhixuan Zhou, Huankang Guan, Meghana Moorthy Bhat, and Justin Hsu. Fake news detection via NLP is vulnerable to adversarial attacks. In *the 11th International Conference on Agents and Artificial Intelligence, ICAART'19*, 2019.