Towards Improving Adversarial Robustness of NLP Models

A

Thesis

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Master of Science

by

Jin Yong Yoo

May 2021

APPROVAL SHEET

This

Thesis

is submitted in partial fulfillment of the requirements
for the degree of

Master of Science

Author: Jin Yong Yoo

This Thesis has been read and approved by the examing committee:

Advisor: Yanjun Qi

Advisor:

Committee Member: Matthew Dwyer

Committee Member: Yangfeng Ji

Committee Member:

Committee Member:

Committee Member:

Committee Member:

Accepted for the School of Engineering and Applied Science:

Craig H. Benson, School of Engineering and Applied Science

May 2021

Abstract

Adversarial training has been extensively studied as a way to improve model's adversarial robustness in computer vision. On the other hand, little attention has been paid in NLP as to how adversarial training affects model's robustness. Within NLP, there exists a significant disconnect between recent works on adversarial training and recent works on adversarial attacks as most recent works on adversarial training have studied it as a means of improving the model's generalization capability instead of as a defense against adversarial attacks.

In this thesis, we investigate how adversarial training can be used to improve the model's adversarial robustness as well as its standard accuracy, cross-domain generalization, and interpretability. In the first half of this thesis, we perform a comprehensive benchmarking of different search algorithms used in NLP adversarial attacks and propose a new simple and efficient search algorithm that can speed up adversarial attacks for adversarial training. Then, using the findings from the benchmark experiments, we create two new adversarial attacks optimized for adversarial training and use them to train BERT and RoBERTa models on IMDB, Rotten Tomatoes, and Yelp datasets. We demonstrate that adversarial training can not only improve the model's robustness to the adversarial attack it was originally trained with, but also defend the model against other types of attacks. Also, we show that adversarial training can improve model's standard accuracy, cross-domain generalization, and interpretability.

Acknowledgments

I want to sincerely thank my advisor Yanjun Qi for her guidance and support during both my undergraduate and master's program. I joined her group as a undergraduate student with little experience in machine learning, and I got to learn so much about both machine learning and research. She encouraged me to explore my ideas and gave valuable advice when I needed them the most.

I would also like to thank Matthew Dwyer and Yangfeng Ji for serving as the committee members on my thesis. Additionally, many thanks to my collaborators Jack Morris, Eli Lifland, Jake Grigsby, and Hanyu Liu, all of whom I got to work with as part of the TextAttack project.

Finally, I want to thank my parents for all the love and support they have given me.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In both computer vision and natural language processing (NLP), robustness of models to adversarial examples has been an active area of research. New methods have been proposed for generating adversarial examples for image classification (Goodfellow et al., 2014; Carlini and Wagner, 2016; Madry et al., 2018), reading comprehension (Jia and Liang, 2017), machine translation (Cheng et al., 2018), and text classification (Ebrahimi et al., 2017; Jia and Liang, 2017; Gao et al., 2018; Alzantot et al., 2018; Jin et al., 2019; Ren et al., 2019; Zang et al., 2020; Garg and Ramakrishnan, 2020; Li et al., 2020, 2021).

At the same time, making models more resistant to these attacks has also been another area of active research. One simple but popular method is adversarial training where the model is further trained on adversarial examples. Adversarial training has been extensively studied as a way to improve model's adversarial robustness in computer vision (Goodfellow et al., 2014; Madry et al., 2018; Zhang et al., 2019a; Kannan et al., 2018; Shafahi et al., 2019; Xie et al., 2020). In comparison, little attention has been paid in NLP as to how adversarial training affects model's adversarial robustness.

In fact, within NLP, there exists a significant disconnect between recent works on adversarial training and those on adversarial attacks. Recent works (Zhu et al., 2019; Jiang et al., 2020; Liu et al., 2020a) explore adversarial training mainly as a means of improving the model's generalization capability instead of as a defense against adversarial attacks. Specifically, they do not evaluate whether such adversarial training method can defend against adversarial attacks that have been proposed in literature.

Additionally, in recent works that have proposed new adversarial attacks, adversarial training as a defense has only been evaluated in limited context. In most cases, adversarial training is only

performed on limited number of models and datasets to mainly show that adversarial training can make models more resistant to the attack it was originally trained with (Jin et al., 2019; Ren et al., 2019; Li et al., 2020; Zang et al., 2020; Li et al., 2021).

In this thesis, we perform a more in-depth investigation into how adversarial training affects model's adversarial robustness as well as its standard accuracy, cross-domain generalization, and interpretability.

In the first half of this work, we study the key components of NLP adversarial attack to address the technical challenges of performing adversarial training. We specifically perform a comprehensive benchmarking of different search algorithms used in literature to perform NLP adversarial attacks and propose a new simple and efficient search algorithm that can speed up adversarial attacks for adversarial training.

In the second half, we create two faster version of adversarial attacks optimized for adversarial training and use them to train BERT (Devlin et al., 2018) and RoBERTa (Liu et al., 2019) models on IMDB (Maas et al., 2011), Rotten Tomatoes (Pang and Lee, 2005), and Yelp (Zhang et al., 2015) datasets. We demonstrate that adversarial training can not only improve the model's robustness to the adversarial attack it was originally trained with, but also defend the model against other types of attacks. Also, we show that adversarial training can improve model's standard accuracy, cross-domain generalization, and interpretability.

1.1 Background and Related Work

Adversarial training was first proposed by Goodfellow et al. (2014) as a defense against adversarial attacks in computer vision. Madry et al. (2018) proposed a more principled approach to adversarial training using a minimax formulation involving the defender and the adversary. They also demonstrated that training using projected gradient descent (PGD) attack provides strong defense against many adversarial attacks for image classification.

Within computer vision, it is largely believed that there exists some inherent trade-off between robustness and standard accuracy (Tsipras et al., 2019; Zhang et al., 2019a; Raghunathan et al., 2019); that is, improving robustness against adversarial attacks leads to some loss in standard accuracy. Tsipras et al. (2019) suggested that the trade-off exists because the features learned by a robust model is fundamentally different from the features learned by the standard model. They also noted an unexpected benefit of adversarial training which is that the features learned by the robust model aligns better with human understanding.

On the other hand, such trade-off has not been observed in NLP. Instead, adversarial training

has been explored as a means to improve the model's generalization performance. Zhu et al. (2019); Jiang et al. (2020); Liu et al. (2020a) demonstrated that adversarial training can improve model's performance on the GLUE benchmark (Wang et al., 2019). Jiang et al. (2020) and Liu et al. (2020a) also showed improvements in robustness by evaluating the models on more difficult, "adversarial" datasets such as ANLI (Nie et al., 2020) or Adversarial SQUAD (Jia and Liang, 2017).

However, these works still do not provide a satisfactory answer to how adversarial training affects adversarial robustness. One thing to note is that these works have all performed adversarial perturbations in the word embedding space. Zhu et al. (2019) adds perturbations to the input embeddings using PGD and uses the perturbed embeddings to further train the models. Jiang et al. (2020) takes a step further by performing adversarial pretraining with smoothness-inducing regularizer introduced by Miyato et al. (2018). Liu et al. (2020a) extends Jiang et al. (2020) by performing curriculum learning where standard pretraining is done first before continuing with adversarial pretraining. Therefore, adversarial training with perturbations in the actual input space is still a relatively unexplored area of research. Furthermore, these works that perturb in the embedding level do not investigate whether such methods of adversarial training can improve robustness against adversarial attacks that occur in the input level.

Our work differs from these works as we aim to explore how adversarial training using perturbations in the input space can be used to defend against adversarial attacks from literature. Moreover, we investigate whether adversarial training with one type of attack can make the model more robust against many different types of attacks. Also, as far we know, no other work has comprehensively studied how adversarial training with perturbations in the input space affects generalization and interpretability for NLP models.

Chapter 2

Adversarial Attacks in NLP

Most of the works on adversarial attacks in NLP draw its inspiration from earlier works in computer vision. However, while an image is a continuous input, text is discrete. This leads to a significant difference in how adversarial examples are generated. In this chapter, we will review how adversarial attacks are performed for NLP models and discuss the underlying combinatorial optimization problem that is solved to generate adversarial examples. We also introduce the novel framework proposed by `TextAttack` (Morris et al., 2020a) that breaks down NLP adversarial attacks into the following four modular components: (1) goal function, (2) set of constraints, (3) transformation, and (4) search method.

2.1 Adversarial Attacks in Vision

In this section, we give a high level overview on adversarial attacks in computer vision. For a more detailed survey, we recommend Akhtar and Mian (2018).

Let F be our neural network represented as a function and let θ be the parameters of the network. To generate an adversarial example from an image $x \in \mathbb{R}^m$ and its label $y \in \{1, \ldots, K\}$, we want to find some perturbation $\delta \in \mathbb{R}^m$ such that $F(x + \delta) = y_k$ where y_k is our desired target label; in the case of untargeted attack[1], we would simply want $F(x + \delta) \neq y$. Since we want δ to be small such that the change is imperceptible to humans, we also need enforce some limit on how big δ

[1] *Targeted attack* is a type of adversarial attack where the attacker's goal is to perturb the input such that the model predicts a specific class label desired by the attacker, while *untargeted attack* is a type of adversarial attack where the attacker's goal is to simply cause the model to misclassify the perturbed input.

can be. This is usually achieved by restricting δ to be within an ℓ_p ball (i.e. $||\delta||_p \leq \epsilon$ for some small $\epsilon > 0$). ℓ_2 or ℓ_∞[2] ball is typically used.

Naturally, finding δ that satisfies our requirements can be formulated as a constrained optimization problem. Szegedy et al. (2014) frames it as the following minimization problem:

$$\begin{array}{ll} \text{minimize} & ||\delta||_2 \\ \text{subject to} & (1) \quad F(x+\delta) = y_k \\ & (2) \quad x + \delta \in [0,1]^m \end{array} \tag{2.1}$$

However, the above equation is difficult to solve due to constraint (1), so Szegedy et al. (2014) instead solves the following problem:

$$\begin{array}{ll} \text{minimize} & c \cdot ||\delta||_2^2 + L(\theta, x+\delta, y_k) \\ \text{subject to} & x + \delta \in [0,1]^m \end{array} \tag{2.2}$$

where $L(\theta, x+\delta, y_k)$ is the loss for the task. L-BFGS algorithm is used to find the minimum $c > 0$ for which minimum δ satisfies constraint (1).

One problem with L-BFGS algorithm is that it is computationally expensive. Goodfellow et al. (2014) presents a faster approximation algorithm called Fast Gradient Sign Method (FGSM) for untargeted attacks that takes advantage of the linearity present in neural networks. Specifically, they solve for the following δ:

$$\delta = \varepsilon * \text{sign}(\nabla L(\theta, x, y)) \tag{2.3}$$

where ε is a constant that we control to make δ small as possible. Here, we can see that for each pixel, FGSM first determines which direction $(+/-)$ the pixel value should be changed to maximize the loss and then scales it by ε.

Besides FGSM, many adversarial attacks have been proposed such as Jacobian-based Saliency Map Attack (JSMA) (Papernot et al., 2015), Carlini & Wagner attack (Carlini and Wagner, 2016), DeepFool (Moosavi-Dezfooli et al., 2015), and projected gradient descent (PGD) (Madry et al., 2018). What all of these works have in common is that the task of generating adversarial example is formulated as a constrained numerical optimization problem.

[2] An ℓ_∞ bound means that we can perturb each pixel only by ϵ.

2.2 Adversarial Attacks in NLP

Naturally, we are interested in whether we can apply the previous formulation to NLP.

2.2.1 Challenges in NLP Adversarial Attack

Let us first attempt to define our problem by borrowing the previous formulation for image classification:

> Given input text $x \in \mathcal{X}$ and its label $y \in \{1, \ldots, K\}$, we want to find some minimal perturbation δ such that our adversarial example $x' = x + \delta$ satisfies
>
> 1. $F(x') = y_k$ where y_k is our desired label
> 2. δ is small (i.e. $||\delta||_p < \epsilon$).

We can easily see that there are several issues with this formulation. For example, *what is our perturbation δ in the case of texts?* For images, adding small values to each pixel is an intuitive way to perturb a given image. However, for texts, there are many ways to modify the discrete input such as replacing a character with another character or inserting new words.

Also, *how do you determine if δ is small enough to not change the ground truth?* Unlike images, we cannot straightforwardly use ℓ_2 (or ℓ_∞) norm to compare the "similarity" of two texts. Lastly, *how do you find the optimal perturbations that will produce x' that satisfies our goals?*

2.2.2 Perturbing Texts

To define δ for text, we need to distinguish perturbations that occurs at the character level from those that occur at the word level. Let input text x be represented as a sequence of tokens $(x_1, x_2, \ldots, x_n)$ where each x_i can be a word or a character. We can come up with three ways to modify x:

1. **Replace**: Replace x_i with another x_i'.
2. **Insertion**: Insert a new x_{i+1}' in front of x_i.
3. **Deletion**: Delete x_i.

If x_i is a character, then replacing, inserting, and deleting characters are all ways of inducing spelling errors in the text. Past works that proposed such character-level attacks include Ebrahimi et al. (2017), Gao et al. (2018), and Pruthi et al. (2020). Pruthi et al. (2020) showed that BERT is sensitive to misspellings as its accuracy on sentiment classification task can decrease from 90.3%

Original Text	The movie was **good**.
Desirable Change	The movie was **great**.
Undesirable Change	The movie was **bad**.

Figure 2.1: Example of a desirable transformation of the original text that preserves the semantics and an example of an undesirable transformation that changes the semantics.

to 48.5% when misspellings are injected to the input. Since humans are capable of understanding misspelled texts, robustness to character-level attacks can be considered as a desirable attribute for models that aim to meet human-level performance.

However, one problem with character-letter attacks is that the resulting text x' is most likely to contain nonsensical words that the model has never encountered before. Most of the recent NLP models that have achieved state-of-the-art (SOTA) performance employ word (or subword-level) embeddings that map words in a fixed vocabulary to a dense vector representation. Since the model's vocabulary is fixed, it is highly likely that the misspelled, gibberish words will simply be mapped to the wrong token or an out-of-vocabulary (OOV) token during the encoding step. This raises the question whether it is reasonable to expect models that are not character-based to be robust against character-level attacks in the first place.

Recent works have proposed word-level attacks where each x_i is a word. Compared to character-level attacks, these methods can generate more fluid and sensible text as an adversarial example. Therefore, we will only consider word-level attacks for the rest of this work. We will especially focus on word replacement as it is the most common type of perturbation strategy found in literature.

2.2.3 Preserving Semantics and Fluency by Constraints

When carrying out word-level attacks, we still need to watch out for changes that introduce grammatical errors or shift the meaning of the text. This is because we want x and x' to be "similar" and do not want to change the ground truth label of the text. For example, as shown by Figure 2.1, given text "The movie was good" for a sentiment-classification task, we do not want to replace the word *good* with its antonym *bad* since the overall sentiment of the text has changed.

Ideally, we want our perturbation to naturally preserve the meaning and fluency of the original text. Past works have therefore proposed various methods for replacing words with their synonyms.

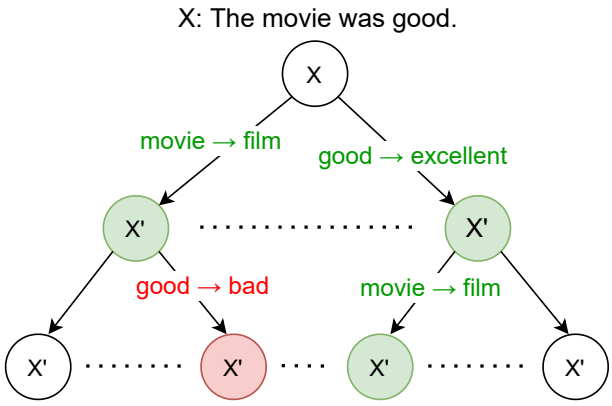

Figure 2.2: Tree representing the combinatorial nature of generating adversarial example. Each edge represents a specific word replacement operation and each node represents the resulting perturbed text. While word replacements involving synonyms, represented by the green color, are desirable, those involving antonyms (red) are not.

Alzantot et al. (2018) and Jin et al. (2019) both use a counter-fitted GloVe word embedding (Mrksic et al., 2016) to find synonyms while Ren et al. (2019) and Zang et al. (2020) use lexical knowledge bases such as WordNet (Miller, 1995) and HowNet (Dong et al., 2010). Recently, Garg and Ramakrishnan (2020), Li et al. (2020), and Li et al. (2021) have proposed using BERT (Devlin et al., 2018) and RoBERTa (Liu et al., 2019) masked language models to generate replacements that are more grammatically coherent.

However, it is possible that undesirable words are proposed as synonyms for replacements. Therefore, many works also use various constraints alongside their replacement strategy to filter out the bad replacements. For example, when considering replacing word x_i with its substitute x_i', both Alzantot et al. (2018) and Jin et al. (2019) filters out x_i' if the cosine similarity between word embedding of x_i and x_i' is below a certain minimum value. Additionally, Jin et al. (2019) uses cosine similarity between sentence encodings of original text x and perturbed text x' (obtained using Universal Sentence Encoder (Cer et al., 2018)) to measure semantic similarity between x and x'.

2.2.4 Adversarial Attack as Combinatorial Optimization

So far, we have discussed how we can perturb a given text by replacing a word with its synonym. However, a single word replacement might not be sufficient to flip the model's prediction, meaning that we have to consider a combination of multiple word replacements.

The different combinations of word replacements can be expressed as a tree shown in Figure 2.2 where each edge represents a specific word replacement. If a given text consists of n words and each word has at max m potential replacements, then we have total of $O(nm)$ options as the first word replacement. If we are to repeat replacement for each word, then the complexity of the total number of possible x' is $O((nm)^n)$. While we can expect the actual solution space to be smaller due to constraints that we place to limit how much we can change the text, the solution space is still too big to perform a brute-force search. Therefore, a heuristic search algorithm is necessary to find the set of word replacement that will achieve our desired outcome[3].

Now, let us attempt to formally define the problem of generating adversarial example x' given input text $x \in \mathcal{X}$ and its label $y \in \{1, \ldots, K\}$ using a word replacement strategy.

Let the set of all possible single word replacements be defined as

$$A = \{(i, x'_{i,j})\}_{i=1,j=1}^{n,m} \tag{2.4}$$

where each $(i, x'_{i,j}) \in A$ means replacing the i^{th} word of x with its j^{th} substitute $x'_{i,j}$.

Let $T(x)$ denote the set of all the possible potential $x' \in \mathcal{X}$ that can be generated via any set of word replacements $\{a_1, \ldots, a_N\} \in 2^A$. If we are to construct a combinatorial tree like Figure 2.2, $T(x)$ corresponds to all the possible nodes that exist in our tree.

For constraints, let us represent them as Boolean functions $C_1, \ldots, C_c$ where each $C_j(x, x') =$ `True` if x and x' satisfies the constraint C_j.

Then, we can define the set of all potential adversarial examples x' as the following:

$$S(x) = \{x' \in T(x) \mid \bigwedge_{j=1}^{c} C_j(x, x')\} \tag{2.5}$$

Finding a particular $x' \in S(x)$ is therefore equivalent to finding the set of word replacements $\{a_1, \ldots, a_N\} \in 2^A$ such that it produces x'.

For targeted attacks, our goal is to find $x' \in S(x)$ such that $F(x') = y_k$ where y_k is our desired label. One way to achieve this is to instead solve the following combinatorial optimization problem:

$$\begin{array}{ll} \min_{x' \in S(x)} & L(\theta, x', y_k) \\ \text{subject to} & F(x') = y_k \end{array} \tag{2.6}$$

[3]Note that while we have so far concerned ourselves with word replacements, insertions and deletions will lead to an even bigger solution space

Since $L(\theta, x', y_k)$ is typically cross-entropy loss for text classification tasks, this is equivalent to maximizing $P(y_k|x'; \theta)$, which is the model's confidence of label y_k given x' and network parameter θ.

For untargeted attacks, we want to find $x' \in S(x)$ such that $F(x') \neq y$ where y is the original label of x. Similar to targeted attacks, we can also try to solve the following problem:

$$\begin{array}{ll} \max_{x' \in S(x)} & L(\theta, x', y) \\ \text{subject to} & F(x') \neq y \end{array} \tag{2.7}$$

To maximize the loss, we want to minimize $P(y|x'; \theta)$.

2.2.5 `TextAttack` Framework

We can see that generating adversarial examples for NLP models involves many different components, such as a word replacement strategy, constraints to filter bad perturbations, and a search algorithm to find the optimal set of word replacements. Now, we formally define these components using the `TextAttack` framework (Morris et al., 2020a). In `TextAttack`, an adversarial attack is compose of a goal function, a set of constraints, a transformation, and a search method. Since we use `TextAttack` to implement adversarial attacks and adversarial training in this work, we will describe adversarial attacks using the four-component framework from now on.

Goal Function

Goal function represents the objective function that we aim to maximize as part of our optimization problem. For targeted attacks with y_k as the target label, the goal function is $P(y_k|x'; \theta)$; for untargeted attacks, the goal function is $1 - P(y|x'; \theta)$ (which is equivalent to minimizing $P(y|x'; \theta)$).

Transformation

Transformation represents the method used to perturb the text. For word replacement, it is equivalent to the method used to find synonyms for each words.

Constraints

For original text x and perturbed text x', constraints determine if x' preserves the ground truth and fluency of x well enough to be considered as an adversarial example.

Search Method

Search method is the heuristic search algorithm used to solve the optimization problem. Generally, search methods attempt to solve the problem by first perturbing the current text with the given transformation and constraints and then evaluating the fitness of resulting x''s using the goal function. Then it repeats the process until we obtain x' that meets our desired conditions (e.g. $F(x') = y_k$ or $F(x') \neq y$) or until we run out of ways to perturb the text. Also, we can force the search method to end earlier by setting a maximum limit to the number of times the search method can *query* the victim model. This maximum limit is known as *query budget*.

Chapter 3

Searching for a Search Method

To perform adversarial training, we first need to come up with the adversarial attack that is used to generate adversarial examples. Since we would have to generate adversarial examples on the fly within the training loop, two key criteria one must consider when choosing the appropriate adversarial attack is its speed and how successfully it can generate adversarial examples. While goal function, constraints, transformation, and search method are all factors that affect the two criteria, search method plays an especially important role as it controls the natural trade-off between speed and capacity by determining how thoroughly we search for the solution. The more exhaustively we search, the slower our attack would be, but higher the chances of finding a solution to the underlying combinatorial optimization problem.

Therefore, the first step to constructing the desired adversarial attack is to choose which search method to use. Recent works have proposed a wide variety of search algorithms to generate adversarial examples. Alzantot et al. (2018) proposed a genetic algorithm to search for the optimal perturbations while Jin et al. (2019) and Ren et al. (2019) proposed a greedy search algorithm that replaces words one by one in order of descending importance. More recently, Zang et al. (2020) proposed a particle swarm optimization (PSO) algorithm to exhaustively search the perturbation space.

However, when it comes to comparing the different search strategies, the literature includes a mixture of incomparable and unclear results since studies often fail to consider the other two necessary primitives in the search process: the search space (choice of transformation and constraints) and the search budget. Past works that propose new search algorithms often also propose a slightly altered search space by proposing either new transformations or new constraints. When new search algorithms are benchmarked in a new search space, they cannot be easily

compared with search algorithms from other attacks. For example, Jin et al. (2019) compares its `TextFooler` method against Alzantot et al. (2018)'s method without accounting for the fact that `TextFooler` uses the Universal Sentence Encoder (Cer et al., 2018) to filter perturbed text while Alzantot et al. (2018) uses Google 1 billion words language model (Chelba et al., 2013).

The lack of a consistent benchmark on search algorithms has hindered the use of adversarial examples to understand and to improve NLP models. We do note that Ren et al. (2019) and Zang et al. (2020) do provide comparisons where the search spaces are consistent. However, these works consider a small number of search algorithms as baseline methods, and fail to provide a comprehensive comparison of methods proposed in the literature.

In this section, we benchmark the various search algorithms proposed in literature and perform fine-grained analysis of three elements relevant to search: search algorithm, search space, and search budget. In the process, we design a new greedy search algorithm that uses gradients to determine the order of words to replace. Our results show that this algorithm is the best search method for adversarial training due to its speed and competitive attack success rate.

3.1 Background

3.1.1 Search Algorithms

Search Algorithm	**Deterministic?**	**Hyperparameters**	**Num. Queries**
Beam Search (Ebrahimi et al., 2017)	✓	b (beam width)	$\mathcal{O}(b * n^2 * m)$
Greedy [Beam Search with b=1]	✓	–	$\mathcal{O}(n^2 * m)$
Greedy w. Word Importance Ranking (Gao et al., 2018; Jin et al., 2019; Ren et al., 2019)	✓	–	$\mathcal{O}(n * m)$
Genetic Algorithm (Alzantot et al., 2018)	✗	p (population size), g (number of iterations)	$\mathcal{O}(g * p * m)$
Particle Swarm Optimization (Zang et al., 2020)	✗	p (population size), g (number of iterations)	$\mathcal{O}(g * p * n * m)$

Table 3.1: Different search algorithms proposed for NLP attacks. n is the number of words in the input. m is the maximum number of transformation options for a given input.

In this section, we will briefly describe the five different types of search algorithms that we have selected for our benchmark. Table 3.1 shows the time complexities of each algorithm with respect

to the length of input x and the maximum number of replacements available for each word.

Beam Search For given input x, all the possible perturbed texts x' generated by substituting each word x_i are scored using the goal function. Then, the top b texts are kept (b is called the "beam width") while the rest are discarded. This process is repeated by further perturbing each of the top b perturbed texts to generate the next set of candidates before aggregating them to identify the next top b texts. We stop once we find a solution.

Greedy Search This is equivalent to beam search where $b = 1$. We take the best x' among all the possible perturbations, and repeat until we succeed or run out of possible perturbations.

Greedy with Word Importance Ranking (WIR) Words $(x_1, \ldots, x_n)$ of x are ranked according to some importance function. Then, in order of descending importance, word x_i is substituted with x'_i that maximizes the scoring function until the goal is achieved, or all words have been perturbed. This is different from beam search or greedy search as we only consider one word to replace at each step. We experiment with four different ways to determine word importance:

- **`UNK`**: Each word's importance is determined by how much the heuristic score changes when the word is substituted with an `UNK` token (Gao et al., 2018).
- **`DEL`**: Each word's importance is determined by how much the heuristic score changes when the word is deleted from the original input (Jin et al., 2019).
- **`PWWS`**: Each word's importance is determined by multiplying the change in score when the word is substituted with an `UNK` token with the maximum score gained by perturbing the word (Ren et al., 2019).
- **`Gradient`**: Similar to how Wallace et al. (2019) visualize saliency of words for explanation, each word's importance is determined by calculating the gradient of the loss with respect to the word[1] and taking its norm. This method has not been explored by previous works.

We test an additional scheme, which we call `RAND`, as an ablation study. Instead of perturbing words in order of their importance, `RAND` perturbs words in a random order.

Genetic Algorithm We implement the genetic algorithm of Alzantot et al. (2018). At each iteration, each member of the population is perturbed by randomly choosing one word and picking the best x' gained by perturbing it. Then, crossover occurs between members of the population, with preference given to the more successful members. The algorithm is run for a fixed number of

[1]For sub-word tokenization scheme, we take average over all sub-words constituting the word.

iterations unless it succeeds in the middle. Following Alzantot et al. (2018), the population size was 60 and the algorithm was run for at maximum 20 iterations.

Particle Swarm Optimization We implement the particle swarm optimization (PSO) algorithm of Zang et al. (2020). At each iteration, each member of the population is perturbed by first generating all potential x' obtained by substituting each $\mathbf{x}_i$ and then sampling one x'. Each member is also crossovered with the best perturb text previously found for the member (i.e. local optimum) and the best perturb text found among all members (i.e. global optimum). Following Zang et al. (2020), the population size is set to 60 and the algorithm was run for a maximum of 20 iterations.

Our genetic algorithm and PSO implementations have one small difference from the original implementations. The original implementations contain crossover operations that further perturb the text without considering whether the resulting text meets the defined constraints. In our implementation, we check if the text produced by these subroutines satisfies our constraints to ensure a consistent search space.

3.1.2 Search Space

Recall that our search algorithm searches within the set of all potential adversarial examples defined by Equation 2.5 to find the x' that changes the model prediction in the desired way.

We can see that the *search space* is defined by both our choice of transformation and constraints. In this section, we explain the word replacement strategies and constraints used for benchmarking.

Transformations

- **Counter-fitted Word Embedding** (Mrksic et al., 2016): For a given word x_i that we want to replace, we take its top N nearest neighbors in the counter-fitted GloVe (Pennington et al., 2014) embedding space as its synonyms. We use counter-fitted embeddings proposed by Mrksic et al. (2016) instead of vanilla GloVe embeddings because counter-fitting pushes synonyms to be closer to one another in the embedding space while keeping antonyms farther apart. Use of this transformation was proposed originally in Alzantot et al. (2018).
- **WordNet** (Miller, 1995): WordNet is a lexical knowledge base that maps the relationships between English words, including synonyms. For given word x_i, we replace it with synonyms found in WordNet.
- **HowNet** (Dong et al., 2010): Similar to WordNet, HowNet is a knowledge base of sememes

in both Chinese and English.

Constraints

- **Word Embedding Similarity**: When we use counter-fitted word embeddings to find synonyms, the cosine similarity between word embeddings of the original word x_i and its replacement x_i' must be above a certain minimum value.
- **Part-of-speech (POS) Consistency**: To preserve fluency, we require that the two words being swapped have the same part-of-speech. This is determined by a part-of-speech tagger provided by Flair (Akbik et al., 2018), an open-source NLP library.
- **BERTScore** (Zhang* et al., 2020): We require that the F1 BERTScore between original text x and perturbed text x' meet some minimum threshold value.
- **Universal Sentence Encoder Similarity** (Cer et al., 2018): We require that the angular similarity between the sentence embeddings of x and x' meet some minimum threshold.

For word embedding similarity, BERTScore, and USE similarity, we need to set the minimum threshold value. We set all three values to be 0.9 based on the observation reported by Morris et al. (2020b) that high threshold values encourages strong semantic similarity. We do not apply word embedding similarity constraint for HowNet and WordNet transformations because it is not guaranteed that we can map the substitute words generated from the two sources to a word embedding space. We can also assume that the substitute words are semantically similar to the original words since they originate from a curated knowledge base.

Lastly, for all attacks carried out, we do not allow perturbing a word that has already been perturbed and we do not perturbed pre-defined stop words.

3.2 Benchmark Setup

3.2.1 Search Spaces

Table 3.2 shows three search spaces we use to benchmark the search algorithms.

3.2.2 Victim Models

We attack `BERT-base` (Devlin et al., 2018) and LSTM (Hochreiter and Schmidhuber, 1997) models trained on three different datasets:

	Transformation	Constraints
1	Counter-fitted GloVe Word Embedding	Word embedding similarity, BERTScore, POS consistency
2	HowNet	BERTScore, POS consistency
3	WordNet	USE similarity, POS consistency

Table 3.2: The three search spaces in our benchmarking.

- Yelp polarity reviews (Zhang et al., 2015) (sentiment classification)
- Movie Reviews (MR) (Pang and Lee, 2005) (sentiment classification)
- Stanford Natural Language Inference (SNLI) (Bowman et al., 2015) (textual entailment).

For Yelp and SNLI dataset, we attack 1000 samples from the test set, and for MR dataset, we attack 500 samples. Language of all three datasets is English.

3.2.3 Evaluation Metrics

We use attack success rate ($\frac{\text{\# of successful attacks}}{\text{\# of total attacks}}$) to measure how successful each search algorithm is at attacking the victim model.

To measure the runtime of each algorithm, we use the average number of queries to the victim model as a proxy.

To measure the quality of adversarial examples generated by each algorithm, we use three metrics:

1. Average percentage of words perturbed
2. Universal Sentence Encoder (Cer et al., 2018) similarity between x and x'
3. Percent change in perplexities of x and x' (using GPT-2 (Radford et al., 2019))

3.3 Results

3.3.1 Evaluation of Adversarial Examples

Table 3.4 shows the average percentage of words perturbed, average Universal Sentence Encoder similarity score, and average percent change in perplexity for all experiments.

Model	Dataset	Search Method	GLOVE Word Embedding		HowNet		WordNet	
			A.S. %	Avg # Queries	A.S. %	Avg # Queries	A.S. %	Avg # Queries
BERT	Yelp	Greedy (b=1)	39.5	810	93.2	3668	63.2	1480
		Beam Search (b=4)	42.0	2857	95.0	10,766	65.9	5033
		Beam Search (b=8)	42.7	5546	95.6	19,810	67.3	9674
		WIR (UNK)	33.2	187	92.3	344	55.3	232
		WIR (DEL)	33.7	189	91.9	364	54.3	238
		WIR (PWWS)	35.3	259	95.1	1300	58.2	395
		WIR (Gradient)	33.2	55	77.6	189	53.7	94
		WIR (RAND)	29.9	61	72.3	279	53.9	118
		Genetic Algorithm	37.6	5098	89.3	11,015	62.1	8257
		PSO	**47.2**	20,279	**96.6**	62,346	**74.9**	28,971
	MR	Greedy (b=1)	20.6	35	78.6	214	59.4	69
		Beam Search (b=4)	21.4	95	80.6	392	64.6	170
		Beam Search (b=8)	**21.8**	175	81.2	632	**65.8**	303
		WIR (UNK)	17.8	28	53.6	58	55.6	40
		WIR (DEL)	17.0	29	53.6	59	54.0	40
		WIR (PWWS)	21.0	41	73.6	205	58.2	71
		WIR (Gradient)	19.8	14	56.6	46	53.4	24
		WIR (RAND)	17.6	12	48.8	49	53.4	24
		Genetic Algorithm	**21.8**	516	80.0	1670	65.6	1063
		PSO	**21.8**	2413	**82.4**	2039	65.4	2078
	SNLI	Greedy (b=1)	19.8	7	87.3	77	49.6	19
		Beam Search (b=4)	**20.1**	12	89.2	97	52.0	33
		Beam Search (b=8)	**20.1**	18	**89.4**	125	**52.6**	49
		WIR (UNK)	19.3	22	85.1	47	47.3	30
		WIR (DEL)	18.5	22	84.8	47	46.7	30
		WIR (PWWS)	19.8	26	86.9	116	49.1	42
		WIR (Gradient)	18.8	5	68.4	25	46.9	10
		WIR (RAND)	18.3	5	82.6	30	46.2	11
		Genetic Algorithm	20.0	78	89.0	477	52.2	250
		PSO	**20.1**	1248	89.1	398	51.9	975
LSTM	Yelp	Greedy (b=1)	53.0	682	98.2	2611	80.0	982
		Beam Search (b=4)	53.2	2313	98.5	7347	81.7	3277
		Beam Search (b=8)	53.5	4516	98.6	13,643	82.3	6240
		WIR (UNK)	49.3	133	95.2	222	75.8	204
		WIR (DEL)	49.1	181	95.2	230	75.3	205
		WIR (PWWS)	51.2	247	97.3	1212	77.8	361
		WIR (Gradient)	49.3	56	90.0	215	75.3	97
		WIR (RAND)	47.4	57	88.3	217	74.6	98
		Genetic Algorithm	51.3	5212	98.3	7408	78.5	7245
		PSO	**54.9**	17,647	**98.8**	34,659	**84.4**	17,145
	MR	Greedy (b=1)	38.4	29	87.6	187	74.2	59
		Beam Search (b=4)	38.6	71	88.6	290	75.6	131
		Beam Search (b=8)	38.6	127	88.8	427	76.0	222
		WIR (UNK)	35.8	27	81.0	51	72.0	36
		WIR (DEL)	36.2	27	80.2	50	72.2	35
		WIR (PWWS)	37.6	40	86.2	203	73.4	68
		WIR (Gradient)	35.4	10	76.6	36	72.8	18
		WIR (RAND)	34.4	11	68.0	40	71.8	22
		Genetic Algorithm	**39.0**	375	88.6	949	76.0	730
		PSO	**39.0**	1592	**89.0**	795	**76.6**	1179

Table 3.3: Comparison of search methods across three datasets. Models are `BERT-base` and LSTM fine-tuned for the respective task. "A.S.%" represents attack success rate and "Avg # Queries" represents the average number of queries made to the model per successful attacked sample.

Model	Dataset	Search Method	GLOVE Word Embedding			HowNet			WordNet		
			Avg P.W. %	Avg USE Sim	Δ% Perplexity	Avg P.W. %	Avg USE Sim	Δ% Perplexity	Avg P.W. %	Avg USE Sim	Δ% Perplexity
BERT	Yelp	Greedy (b=1)	3.41	0.948	21.5	2.52	0.945	22.8	4.76	0.943	49.9
		Beam Search (b=4)	3.26	0.949	20.7	2.45	0.946	22.0	4.49	0.944	46.7
		Beam Search (b=8)	**3.20**	**0.950**	**20.1**	**2.42**	**0.947**	**21.4**	**4.46**	**0.945**	**46.4**
		WIR (UNK)	6.48	0.930	43.5	4.73	0.922	42.3	9.02	0.924	92.1
		WIR (DEL)	6.85	0.928	47.2	5.10	0.919	46.4	9.38	0.923	98.8
		WIR (PWWS)	4.36	0.942	27.3	3.11	0.94	28.1	6.10	0.937	66.1
		WIR (Gradient)	6.16	0.933	37.8	5.58	0.913	44.5	9.10	0.925	86.4
		WIR (RAND)	8.18	0.920	59.1	7.46	0.898	74.6	11.16	0.914	124.8
		Genetic Algorithm	5.06	0.936	33.9	4.21	0.928	42.7	6.70	0.932	77.3
		PSO	6.61	0.929	47.3	6.08	0.913	62.3	9.67	0.922	111.0
	MR	Greedy (b=1)	7.25	0.900	31.8	6.14	0.887	36.5	10.26	0.864	102.8
		Beam Search (b=4)	**7.22**	**0.901**	**31.4**	**6.10**	**0.887**	**36.1**	10.10	**0.866**	**97.9**
		Beam Search (b=8)	**7.22**	**0.901**	**31.4**	**6.10**	**0.887**	**36.1**	**10.05**	**0.866**	101.6
		WIR (UNK)	9.42	0.884	42.3	7.77	0.866	48.0	14.14	0.845	141.2
		WIR (DEL)	9.62	0.882	46.4	7.69	0.865	46.1	14.60	0.840	146.4
		WIR (PWWS)	7.36	0.898	33.8	6.22	0.884	37.6	10.80	0.865	111.1
		WIR (Gradient)	8.61	0.892	38.1	8.25	0.862	40.8	14.58	0.844	123.2
		WIR (RAND)	10.1	0.881	51.4	9.93	0.846	69.5	17.28	0.827	149.4
		Genetic Algorithm	8.18	0.895	35.8	6.41	0.885	37.8	12.30	0.854	124.5
		PSO	8.71	0.894	39.0	6.46	0.884	38.7	16.08	0.839	187.8
	SNLI	Greedy (b=1)	5.59	0.915	37.8	5.02	0.889	31.7	6.53	0.903	55.9
		Beam Search (b=4)	**5.59**	**0.916**	**37.8**	**5.02**	**0.889**	**31.6**	**6.50**	**0.903**	**55.7**
		Beam Search (b=8)	5.59	0.916	37.8	5.02	0.889	31.6	6.50	0.903	55.9
		WIR (UNK)	6.56	0.911	42.8	5.65	0.887	33.4	8.03	0.899	65.5
		WIR (DEL)	6.77	0.91	44.0	5.81	0.887	34.2	8.22	0.898	67.6
		WIR (PWWS)	5.63	0.915	37.8	5.05	0.89	30.5	6.59	0.906	54.5
		WIR (Gradient)	6.57	0.911	41.6	5.9	0.881	37.7	8.06	0.899	65.1
		WIR (RAND)	7.06	0.909	47.7	6.19	0.884	42.9	8.65	0.895	74.6
		Genetic Algorithm	5.71	0.915	38.5	5.14	0.888	32.7	6.73	0.902	58.3
		PSO	5.76	0.915	38.6	5.14	0.888	32.5	6.94	0.902	58.5
LSTM	Yelp	Greedy (b=1)	4.04	0.943	28.9	2.47	0.948	23.9	4.58	0.946	52.1
		Beam Search (b=4)	4.01	0.942	28.9	2.47	0.949	23.7	4.53	0.946	51.9
		Beam Search (b=8)	**4.01**	**0.943**	**28.7**	**2.44**	**0.949**	**23.0**	**4.51**	**0.946**	**51.3**
		WIR (UNK)	5.83	0.933	42.4	3.51	0.935	34.4	7.22	0.935	75.6
		WIR (DEL)	5.86	0.932	41.1	3.61	0.936	33.3	7.22	0.935	75.4
		WIR (PWWS)	4.57	0.940	32.6	2.58	0.947	23.9	5.14	0.944	57.0
		WIR (Gradient)	7.05	0.926	52.2	5.25	0.916	50.7	8.42	0.929	87.9
		WIR (RAND)	7.28	0.925	53.6	6.33	0.906	69.5	9.40	0.925	102.4
		Genetic Algorithm	5.94	0.933	42.8	3.73	0.930	41.9	6.37	0.936	80.9
		PSO	6.70	0.929	47.3	5.03	0.924	58.7	7.98	0.93	95.2
	MR	Greedy (b=1)	**7.19**	**0.899**	**33.5**	**5.96**	0.884	37.2	10.21	0.871	100.6
		Beam Search (b=4)	7.19	0.899	33.7	5.96	0.884	37.6	10.03	0.871	98.7
		Beam Search (b=8)	7.19	0.899	34.0	5.96	0.884	37.6	**10.00**	**0.871**	97.4
		WIR (UNK)	8.99	0.889	41.7	7.22	0.874	42.9	12.99	0.856	104.5
		WIR (DEL)	9.17	0.889	44.5	7.21	0.874	42.2	13.03	0.856	107.5
		WIR (PWWS)	7.45	0.898	33.7	6.01	0.884	**37.1**	10.50	0.871	**87.9**
		WIR (Gradient)	8.73	0.892	41.4	7.33	0.870	40.8	13.12	0.859	104.1
		WIR (RAND)	10.60	0.880	54.0	9.31	0.853	57.7	16.05	0.842	148.2
		Genetic Algorithm	8.02	0.896	36.4	6.36	0.881	39.5	11.98	0.860	120.2
		PSO	8.41	0.893	40.2	6.32	0.882	40.8	13.90	0.854	130.0

Table 3.4: Quality evaluation of the adversarial examples produced by each search algorithm. "Avg P.W. %" means average percentage of words perturbed, "Avg USE Sim" means average USE angular similarity, and "Δ% Perplexity" means percent change in perplexities.

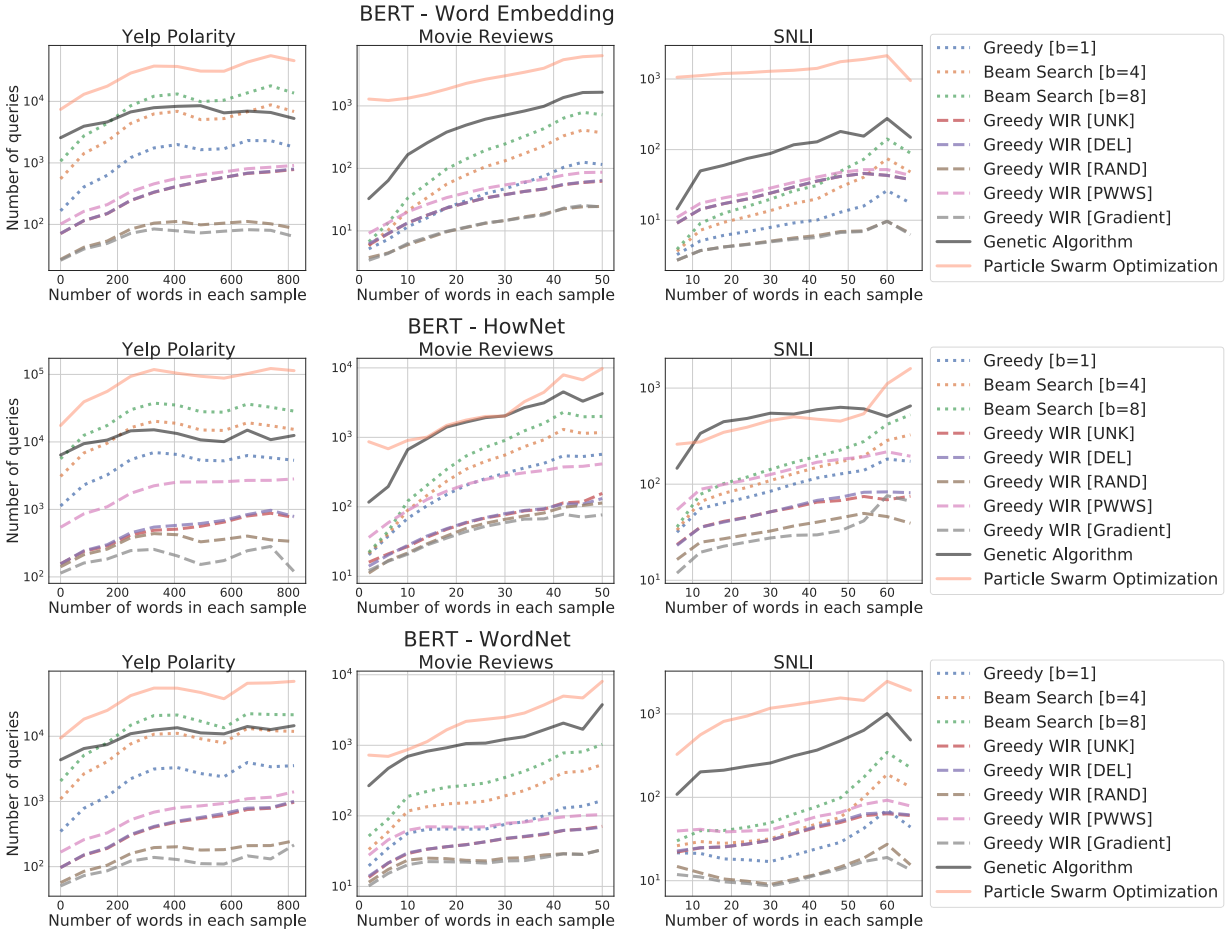

Figure 3.1: Number of queries vs. length of input text. Similar figure for LSTM models are available in appendix A.1.

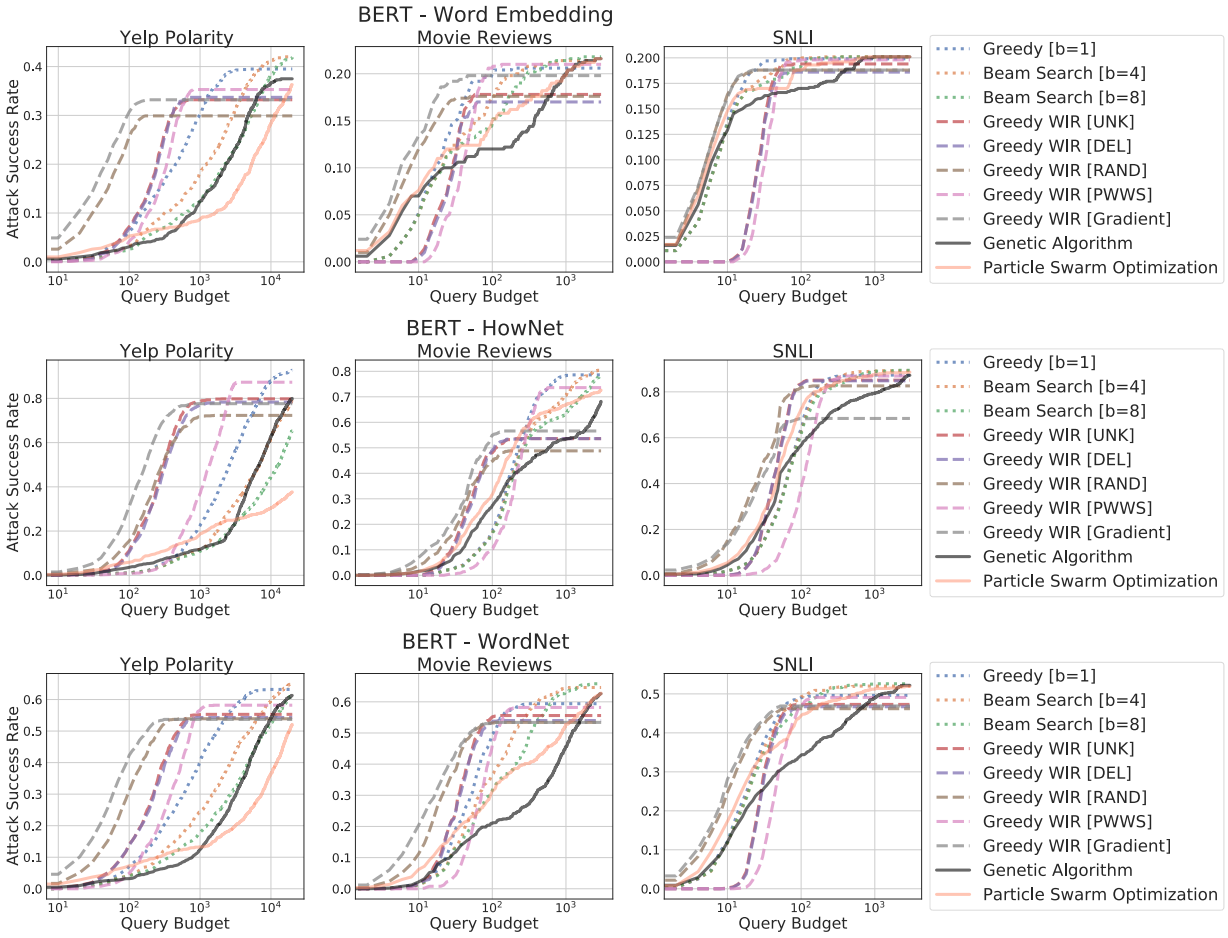

Figure 3.2: Attack success rate by query budget for each search algorithm and dataset. Similar figure for LSTM models are available in appendix A.1.

3.3.2 Attack Success Rate Comparison

Table 3.3 shows the attack success rate and the average number of queries of each search algorithm when it is allowed to query the victim model an unlimited number of times. Word importance ranking methods make far fewer queries than beam or population-based search while retaining over 60% of their attack success rate in each case. Beam search (`b=8`) and PSO are the two most successful search algorithms in every model-dataset combination. However, PSO is more query-intensive. On average, PSO requires 6.3 times[2] more queries than beam search (`b=8`), but its attack success rate is only on average 1.2% higher than that of beam search (`b=8`).

3.3.3 Runtime Analysis

Using number of queries to the victim model as proxy for total runtime, Figure 3.1 illustrates how the number of words in the input affects runtime for each algorithm. We can empirically confirm that beam and greedy search algorithms scale quadratically with input length, while word importance ranking scales linearly. For shorter datasets, this did not make a significant difference. However, for the longer Yelp dataset, the linear word importance ranking strategies are significantly more query-efficient. These observations match the expected runtimes of the algorithms described in Table 3.1.

For shorter datasets, genetic and PSO algorithms are significantly more expensive than the other algorithms as the size of population and number of iterations are the dominating factors. Furthermore, PSO is observed to be more expensive than genetic algorithm.

3.3.4 Performance under Query Budget

In a realistic attack scenario, the attacker must conserve the number of queries made to the model. To see which search method was most query-efficient, we calculated the search methods' attack success rates under a range of query budgets. Figure 3.2 shows the attack success rate of each search algorithm as the maximum number of queries permitted to perturb a single sample varies from 0 to $20{,}000$ for Yelp dataset and 0 to 3000 for MR and SNLI.

For both Yelp and MR datasets, the linear (word importance ranking) methods show relatively high success rates within just a few queries, but are eventually surpassed by the slower, quadratic methods (greedy and beam search). The genetic algorithm and PSO lag behind. For SNLI, we see exceptions as the initial queries that linear methods make to determine word importance ranking does not pay off as other algorithms appear more efficient with their queries. This shows that

[2]This is with one outlier (BERT-SNLI with GLOVE word embedding) ignored. If it is included, the number jumps to 10.8.

the most effective search method depends on both on the attacker's query budget and the victim model. An attacker with a small query budget may prefer a linear method, but an attacker with a larger query budget may aim to choose a quadratic method to make more queries in exchange for a higher success rate.

Lastly, we can see that both `Gradient` and `RAND` ranking methods are initially more successful than `UNK` and `DEL` methods, which is due to the overhead involved in calculating word importance ranking for `UNK` and `DEL` – for both methods, each attack needs to make $O(n)$ queries to the victim model to determine the importance of each word. Still, `UNK` and `DEL` outperform `RAND` at all but the smallest query budgets, indicating that the order in which words are replaced do matter.

3.3.5 Quality of Adversarial Examples

We selected adversarial examples whose original text x was successfully attacked by all search algorithms for quality evaluation. Full results of quality evaluation are shown in Table 3.4. We can see that beam search algorithms consistently perturb the lowest percentage of words. Furthermore, we see that a fewer number of words perturbed generally corresponds with higher average USE similarity between x and x_{adv} and a smaller increase in perplexity. This indicates that the beam search algorithms generate higher-quality adversarial examples than other search algorithms.

3.4 Discussions

3.4.1 Search Method for Adversarial Training

Across all nine scenarios, we can see that choice of search algorithm can have a modest impact on the attack success rate. Query-hungry algorithms such as beam search, genetic algorithm, and PSO perform better than fast WIR methods. Out of the WIR methods, `PWWS` performs significantly better than `UNK` and `DEL` methods but requires far more queries; in every case, we see a clear trade-off of performance versus speed.

With this in mind, one might wonder about what the best way is to choose a suitable search method for adversarial training. The main factor to consider is the length of the input text. If the input texts are short (e.g. sentence or two), beam search is certainly the appropriate choice: it can achieve a high success rate without sacrificing too much speed. However, in most cases, our input can consist of several sentence (e.g. Yelp datasets). In such cases, WIR methods are the practical choices. For instance, we can see from our results that the relative difference in attack success

rate between Gradient method and PSO algorithm is at maximum 30%, which is acceptable given that Gradient is approximately 300 times faster than PSO.

Out of five WIR methods, Gradient appears to be the best algorithm for adversarial training since it works well with limited query budget. Other WIR method such as UNK and DEL require linear number of additional queries to measure the importance of each word, but Gradient simply requires one backward pass to calculate the gradient of each word.

3.4.2 Effectiveness of PWWS Word Importance Ranking

Across all tasks, the UNK and DEL methods perform about equivalently, while PWWS performs significantly better than UNK and DEL. In fact, PWWS performs better than greedy search in two cases. However, this gain in performance does come at a cost: PWWS makes far larger number of queries to the victim model to determine the word importance ranking. Out of the 15 experiments, PWWS makes more queries than greedy search in 8 of them. Yet, on average, greedy search outperforms PWWS by 2.5%.

Our results question the utility of the PWWS search method. PWWS neither offers the performance that is competitive when compared to greedy search nor the query efficiency that is competitive when compared to UNK or DEL.

3.4.3 Effectiveness of Genetic Algorithm

The genetic algorithm proposed by Alzantot et al. (2018) uses more queries than the greedy-based beam search (b=8) in 11 of the 15 scenarios, but only achieves a higher attack success rate in 1 scenario. Thus it is generally strictly worse than the simpler beam search (b=8), achieving a lower success rate at a higher cost.

Chapter 4

Adversarial Training for NLP Models

In this chapter, we take a closer look at how performing adversarial training with adversarial examples affects the model's robustness, standard accuracy, cross-domain generalization, and interpetability. To do so, we use our findings from the previous chapter and methods proposed from literature to build two adversarial attacks that are optimized for adversarial training. Specifically, we build faster versions of `TextFooler` (Jin et al., 2019) and `BAE` (Garg and Ramakrishnan, 2020) attacks by replacing the search algorithm with greedy search with gradient-based word importance ranking and replacing the Universal Sentence Encoder (Cer et al., 2018) with a DistilBERT (Sanh et al., 2019) encoder trained on semantic textual similarity (STS) task. We also address some technical challenges to adversarial training that arise to due to the complexity of adversarial attacks, including our choice to perform epoch-level generation of adversarial examples instead of minibatch-level generation.

We train BERT (Devlin et al., 2018) and RoBERTa (Liu et al., 2019) models on IMDB (Maas et al., 2011), MR (Pang and Lee, 2005), and Yelp (Zhang et al., 2015) datasets. Our findings are as following:

- Adversarial training can help improve adversarial robustness, even against attacks that were not used to trained the model.
- Adversarial training can provide a regularization effect and improve the model's standard accuracy and cross-domain generalization.
- Using LIME (Ribeiro et al., 2016), we demonstrate that adversarial training can improve the model's interpretability.

4.1 Background

4.1.1 Adversarial Training

Adversarial training was first proposed by Goodfellow et al. (2014) as a defense against adversarial attacks. The model was trained on both clean[1] and adversarial examples with the following objective loss function:

$$\tilde{L}(\theta, x, y) = \alpha L(\theta, x, y) + (1 - \alpha) L(\theta, x + \varepsilon * \text{sign}(\nabla L(\theta, x, y)), y) \tag{4.1}$$

where $L(\theta, x, y)$ represents the loss function for input image x and label y given model with parameter θ and $x + \varepsilon * \text{sign}(\nabla L(\theta, x, y))$ represents the adversarial example that is generated via FGSM (Goodfellow et al., 2014). α is used to weigh the two losses; in Goodfellow et al. (2014), α is set to 0.5, weighing the two losses equally.

Madry et al. (2018) proposed another formulation where only adversarial examples are used to further train the model:

$$\arg\min_{\theta} E_{(x,y)\sim D}[\max_{\delta \in S} L(\theta, x + \delta, y)] \tag{4.2}$$

The inner maximization problem finds the adversarial example while the outer minimization problem trains the model. Here, the perturbation δ is found using projected gradient descent (PGD) attack (Madry et al., 2018), which can be interpreted as a iterative version of FGSM that performs multiple gradient descent operations to find δ.

4.2 Method

In this section, we present the training objective for adversarial training and the algorithm for adversarial training. We also discuss the specific adversarial attacks used to train the models.

4.2.1 Training Objective

We observed that using a mix of clean and adversarial examples leads to better performance - as suggested by Goodfellow et al. (2014); Kurakin et al. (2016a) - instead of using only adversarial examples (Madry et al., 2018). Therefore, we aim to minimize both the loss on the original training dataset and the loss on the adversarial examples.

Let $L(\theta, x, y)$ represent the loss function for input text x and label y and let $A(\theta, x, y)$ be the

[1] *Clean* examples refer to the examples from the original training set.

adversarial attack that produces adversarial example x_{adv}. Then, our training objective is as following:

$$\arg\min_{\theta} E_{(x,y)\sim D}[\alpha L(\theta, x, y) + (1-\alpha)L(\theta, A(\theta, x, y), y)] \tag{4.3}$$

α here is again used to weigh the two losses. In our work, we simply set $\alpha = 0.5$, weighing the two loss equally. Tuning of the optimal α is left for future work.

4.2.2 Training Algorithm in Practice

In practice, it is computationally difficult to generate adversarial examples between every mini-batch update. BERT and RoBERTa models require large amounts of GPU memory to store the computation graph during training and an adversarial attack also typically requires other neural networks as its sub-components (e.g. sentence encoders, masked language models). As a result, it is impossible to run adversarial attacks and train the model in the same GPU. We instead maximize GPU utilization by first generating adversarial examples before every epoch and then using them to train the model.

Also, we choose to first train the model on just the original training set for a certain number of epochs before training it on both clean and adversarial examples. This is to have the model learn the rough decision boundaries before performing adversarial attacks. We observe that this works better than training with adversarial examples from the start.

Lastly, instead of generating adversarial examples for every clean example in the training dataset, we randomly sample a fixed portion of the dataset and use them to generate the adversarial examples. This is primarily to reduce the runtime. For cases where the adversarial attack fails to find an adversarial example, we skip them and instead sample more from the training dataset to compensate for the skipped samples. For our experiments, we attack 20% of the training dataset.

Algorithm 1 shows the training algorithm in detail. We run clean training for N_{clean} number of epochs before performing N_{adv} epochs of adversarial training. Between line 6-13, we generate the adversarial examples until we obtain γ percentage of the training dataset. When multiple GPUs are available, we use data parallelism to speed up the generation process. We also shuffle the dataset before attacking to avoid attacking the same sample every epoch.

4.2.3 Fast Adversarial Attacks

We construct faster versions of `TextFooler` (Jin et al., 2019) and `BAE` (Garg and Ramakrishnan, 2020) attacks to generate adversarial examples. We call them `Fast-TextFooler` and

Algorithm 1 Adversarial Training

Require: Number of clean epochs N_{clean}, number of adversarial epochs N_{adv}, percentage of dataset to attack γ, attack $A(\theta, x, y)$, and training data $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, α the smoothing proportion of adversarial training

Initialize model θ
for clean epoch$= 1, \ldots, N_{\text{clean}}$ **do**
  Train θ on D
end for
for adversarial epoch$= 1, \ldots, N_{\text{adv}}$ **do**
  Randomly shuffle D
  $D_{\text{adv}} \leftarrow \{\}$
  $i \leftarrow 1$
  while $|D_{\text{adv}}| < \gamma * |D|$ and $i \leq |D|$ **do**
    $x_{\text{adv}}^{(i)} \leftarrow A(\theta, x^{(i)}, y^{(i)})$
    $D_{\text{adv}} \leftarrow D_{\text{adv}} \cup \{(x_{\text{adv}}^{(i)}, y^{(i)})\}$
    $i \leftarrow i + 1$
  end while
  $D' \leftarrow D \cup D_{\text{adv}}$
  Train θ on D' with α used to weigh the losses
end for

`Fast-BAE` , respectively. Both utilize the same goal function, constraints, and search method but differ in transformation.

Transformation

1. `Fast-Textfooler`: Counter-fitted word embedding (Mrksic et al., 2016)
2. `Fast-BAE`: BERT masked language model (Devlin et al., 2018)

We use these two different transformations because they each prioritize different qualities when proposing replacements. Counter-fitted word embeddings (Mrksic et al., 2016) are likely to propose synonyms as replacements, but could produce an incoherent text as it does not take the entire context into account. On the other hand, BERT masked language model is more likely to propose replacement words that preserve grammatical and contextual coherency but fail to preserve the semantics. For example, if we have text “the movie was good”, BERT masked language model can, with high probability, replace the word “good” with its antonym “bad” since both are suitable given the context; this can lead to false positive adversarial examples. Therefore,

by comparing the adversarial training results of the two different strategies, we can indirectly learn more about which one generates higher quality adversarial examples.

Goal Function

In this work, we perform untargeted attack since they are generally easier than targeted attack. We aim to maximize the following as our goal function:

$$1 - P(y|x;\theta) \tag{4.4}$$

where $P(y|x;\theta)$ means the model's confidence of label y given input x and parameters θ.

Constraints

We use the following constraints for both attacks:

- **Part-of-speech Consistency**: To preserve fluency, we require that the two words being swapped have the same part-of-speech. This is determined by a part-of-speech tagger provided by Flair (Akbik et al., 2018), an open-source NLP library.
- **DistilBERT Semantic Textual Similarity (STS)** (Sanh et al., 2019): We require that cosine similarity between the sentence encodings of original text x and perturbed text x' meet minimum threshold value of 0.9. We use fine-tuned DistilBERT model provided by Reimers and Gurevych (2019).
- **Max modification rate**: We allow only 10% of the words to be replaced. This limits us from modifying the text too much and causing the semantics of the text to change.

Also, for `Fast-TextFooler` attack, we require that the word embeddings between original text x and perturbed text x' have minimum cosine similarity of 0.8.

For sentence encoding similarity, we used DistilBERT model instead of Universal Sentence Encoder (USE) (Cer et al., 2018) because DistilBERT is small enough to fit in GPU memory alongside the victim model and is faster than USE.

The threshold values for word embedding similarity and sentence encoding similarity were set based on the recommendations by Morris et al. (2020b), which noted that high threshold values encourages strong semantic similarity between the original text and the perturbed text.

	Train	Dev	Test
IMDB	20k	5k	25k
MR	8.5k	1k	1k
Yelp	30k	10k	38k

Table 4.1: Overview of the datasets.

Search Method

Based on our results from the previous benchmarking study, we can see that greedy search with gradient-based word importance ranking offers the fastest performance without sacrificing the attack success rate too much. We therefore use gradient-based word importance ranking as the search algorithm for our attacks. During training, we limit the search method to making only 200 queries to the victim model for faster generation of adversarial examples. For evaluation, we increase the query budget to 2000 queries for a more extensive search.

4.3 Experiment

4.3.1 Datasets & Models

We chose IMDB (Maas et al., 2011), Movie Reviews (MR) (Pang and Lee, 2005), and Yelp (Zhang et al., 2015) datasets for our experiment. For Yelp, instead of using the entire training set, we sampled 30k examples for training and 10k for validation. While the three share the same sentiment classification task, Yelp consists of business reviews while IMDB and MR consists of movie reviews.

We trained `BERT-base` (Devlin et al., 2018) and `RoBERTa-base` (Liu et al., 2019) models using the implementation provided by Wolf et al. (2020). All texts were tokenized up to the first 512 tokens and we trained the model for one clean epoch and three adversarial epochs. Adam optimizer with weight decay of 0.01 (Loshchilov and Hutter, 2017) and learning rate of $5\text{e}{-5}$ were used for training. Also, we used a linear scheduler with 500 warm-up steps for IMDB and Yelp and 100 warm-up steps for MR. We performed three runs with random seeds for each model. Average of the three runs are reported as the result.

4.3.2 Baselines

Adversarial training can be viewed as a data augmentation method where *hard* examples are added to the training set. Therefore, besides just having models that are trained on clean adversarial

examples (i.e. "natural training") as our baseline, we also compare our results to models trained using more conventional data augmentation methods. We use SSMBA (Ng et al., 2020) and back-translation[2] (Xie et al., 2019) methods as our baselines as both have reported strong performance on text classification tasks. We use these methods to generate approximately the same number of new training examples as adversarial training.

4.4 Results

4.4.1 Adversarial Robustness

To evaluate the model's adversarial robustness, we attack 1000 randomly sampled clean examples from the test set and measure the attack success rate.

$$\text{attack success rate} = \frac{\text{\# of successful attacks}}{\text{\# of total attacks}}$$

Tables 4.2 and 4.3 show the attack success rates of `Fast-TextFooler` attack and `Fast-BAE` attack against the trained models. Note that the overall attack success rates appear fairly low because we applied strict constraints to improve the quality of the adversarial examples (as recommend by Morris et al. (2020b)). Still, we can see that for both attacks, adversarial training using the same attack can decrease the attack success rate by up to 70%. What is more surprising is that training the model using a different attack also led to a decrease in the attack success rate. From Table 4.2, we can see that adversarial training using the `Fast-BAE` attack lowers the attack success rate of `Fast-TextFooler` attack, while Table 4.3 shows that training with `Fast-TextFooler` lowers the attack success rate of `Fast-BAE` attack.

Another surprising observation is that training with data augmentations methods like SSMBA and backtranslation can lead to large improvements in robustness against both adversarial attacks. However, in case of smaller datasets such as MR, data augmentation can also hurt robustness.

When we compare the attack success rates between BERT and RoBERTa models, we also see an interesting pattern. BERT models tend to be more vulnerable to `Fast-TextFooler` attack while RoBERTa model tends to be more vulnerable to `Fast-BAE` attack.

Lastly, we use attacks proposed from literature to evaluate the models' adversarial robustness. Table 4.4 shows the attack success rate of `TextFooler` (Jin et al., 2019), `BAE` (Garg and Ramakrishnan, 2020), `PWWS` (Ren et al., 2019), and `PSO` (Zang et al., 2020) [3]. Across three

[2]For backtranslation, we use English-to-German model and German-to-English model trained by Ng et al. (2019).
[3]These attacks were implemented using the `TextAttack` library (Morris et al., 2020a).

Model	Method	IMDB		MR		Yelp	
		Attack Success %	Δ%	Attack Success %	Δ%	Attack Success %	Δ%
BERT	Natural	42.9	-	20.9	-	25.4	-
	Fast-TextFooler	**12.7**	**-70.4%**	**13.2**	**-36.8%**	**11.5**	**-54.7%**
	Fast-BAE	34.5	-19.6%	18.9	-9.6%	21.0	-17.3%
	SSMBA	29.5	-31.2%	21.1	1.0%	23.3	-8.3%
	Backtranslation	33.1	-22.8%	19.2	-8.1%	24.0	-5.5%
RoBERTa	Natural	34.3	-	18.6	-	19.9	-
	Fast-TextFooler	**12.4**	**-63.7%**	**12.1**	**-34.9%**	**7.6**	**-61.8%**
	Fast-BAE	19.5	-43.0%	17.1	-8.1%	13.0	-34.7%
	SSMBA	24.0	-29.9%	21.8	17.2%	19.3	-3.0%
	Backtranslation	28.9	-15.6%	18.3	-1.6%	16.1	-19.1%

Table 4.2: Attack success rate of Fast-TextFooler attack. Δ% column represents the percent change between natural training and the different training methods.

datasets and two models, we can see that both Fast-TextFooler and Fast-BAE lower the attack success rate against all four attacks in all but four cases. The results for PWWS and PSO are especially surprising since both use different transformations - WordNet (Miller, 1995) and HowNet (Dong et al., 2010) - when carrying out the attacks.

4.5.1 Generalization

To evaluate how adversarial training affects the model's generalization ability, we evaluate its accuracy on the original test set (i.e. standard accuracy) and on an out-of-domain dataset (e.g. Yelp dataset for model trained on IMDB dataset). In Table 4.5, we can see that in all cases, adversarial training using Fast-TextFooler attack beats both natural training and data augmentation methods in standard accuracy. Fast-TextFooler also improves cross-domain accuracy in at least half the cases. On the other hand, adversarial training with Fast-BAE attack tends to hurt both standard accuracy and cross-domain accuracy. This confirms the observations reported by Li et al. (2021) and suggests that using a masked language model to generate adversarial examples can lead to a trade-off between robustness and generalization. We do not see similar trade-off with Fast-TextFooler .

4.5.2 Interpretability

We use LIME (Ribeiro et al., 2016) to generate local explanations for our models. For each example, LIME approximates the local decision boundary by fitting a linear model over the

Model	Method	IMDB		MR		Yelp	
		Attack Success %	Δ%	Attack Success %	Δ%	Attack Success %	Δ%
BERT	Natural	76.6	-	37.7	-	47.1	-
	Fast-TextFooler	61.7	-19.5%	33.2	-11.9%	42.5	-9.8%
	Fast-BAE	**48.3**	**-37.0%**	**24.7**	**-34.5%**	**27.9**	**-40.8%**
	SSMBA	59.6	-22.2%	36.2	-4.0%	44.8	-4.9%
	Backtranslation	68.8	-10.2%	36.3	-3.7%	46.8	-0.6%
RoBERTa	Natural	81.5	-	40.9	-	53.2	-
	Fast-TextFooler	69.8	-14.4%	38.4	-6.1%	45.2	-15.0%
	Fast-BAE	**37.0**	**-54.6%**	**28.5**	**-30.3%**	**25.8**	**-51.5%**
	SSMBA	57.0	-30.1%	43.1	5.4%	47.8	-10.2%
	Backtranslation	74.3	-8.8%	41.1	0.2%	43.8	-17.7%

Table 4.3: Attack success rate of Fast-BAE attack. Δ% column represents the percent change between natural training and the different training methods.

samples obtained by perturbing the example. To measure the faithfulness of the local explanations obtained using LIME, we measure the area over perturbation curve (AOPC) (Samek et al., 2017; Nguyen, 2018; Chen and Ji, 2020) which is defined as:

$$AOPC = \frac{1}{K+1} \sum_{k=1}^{K} \frac{1}{N} \sum_{i=1}^{N} f(x_{(0)}^{(i)}) - f(x_{(k)}^{(i)}) \tag{4.5}$$

where $x_{(0)}^{(i)}$ represents example $x^{(i)}$ with none of the words removed and $x_{(k)}^{(i)}$ represents example $x^{(i)}$ with the top-k most important words removed. $f(x)$ here represents the model's confidence of the target label $y^{(i)}$. Intuitively, AOPC measures how, on average, the model's confidence of the target label changes when we delete the top-k most important words determined by our explanation method.

For each dataset, we randomly pick 1000 examples from the test set for evaluation, similar to how we evaluate robustness. When running LIME to obtain our explanations, we generate 1000 perturbed samples for each instance. We set $K = 10$ for the AOPC metric.

Table 4.6 shows that across all datasets, BERT model trained using Fast-TextFooler attack achieves higher AOPC than natural training. For RoBERTa models, the results are bit more varied. Fast-TextFooler attack achieves higher AOPC score for IMDB and Yelp datasets, but lower score for MR; for MR, adversarial training with Fast-BAE achieves the highest AOPC score. Overall, we see that RoBERTA models are less interpretable than BERT models.

Adversarial Attack	Model	Training Method	IMDB		MR		Yelp	
			Attack Success %	Δ%	Attack Success %	Δ%	Attack Success %	Δ%
TextFooler	BERT	Natural	85.0	-	91.6	-	55.9	-
		Fast-TextFooler	66.0	-22.4%	90.6	-1.1%	57.9	3.6%
		Fast-BAE	88.2	3.8%	89.0	-2.8%	67.7	21.1%
	RoBERTa	Natural	95.2	-	94.4	-	74.5	-
		Fast-TextFooler	82.4	-13.4%	91.0	-3.6%	68.7	-7.8%
		Fast-BAE	72.9	-23.4%	88.6	-6.1%	71.7	-3.8%
BAE	BERT	Natural	60.5	-	52.6	-	37.8	-
		Fast-TextFooler	46.7	-22.8%	51.5	-2.1%	34.4	-9.0%
		Fast-BAE	52.4	-13.4%	43.8	-16.7%	31.3	-17.2%
	RoBERTa	Natural	65.5	-	56.4	-	44.4	-
		Fast-TextFooler	56.8	-13.3%	54.7	-3.0%	38.0	-14.4%
		Fast-BAE	42.3	-35.4%	48.3	-14.4%	28.7	-35.4%
PWWS	BERT	Natural	87.5	-	82.1	-	67.9	-
		Fast-TextFooler	70.9	-19.0%	80.4	-2.1%	65.4	-3.7%
		Fast-BAE	87.1	-0.5%	81.3	-1.0%	72.2	6.3%
	RoBERTa	Natural	96.6	-	83.8	-	77.9	-
		Fast-TextFooler	84.4	-12.6%	81.9	-2.3%	73.1	-6.2%
		Fast-BAE	73.5	-23.9%	79.8	-4.8%	70.7	-9.2%
PSO	BERT	Natural	43.8	-	81.6	-	40.3	-
		Fast-TextFooler	16.5	-62.3%	73.2	-10.3%	26.4	-34.5%
		Fast-BAE	29.9	-31.7%	75.4	-7.6%	34.4	-14.6%
	RoBERTa	Natural	34.8	-	88.0	-	35.7	-
		Fast-TextFooler	12.9	-62.9%	81.6	-7.3%	21.6	-39.5%
		Fast-BAE	13.1	-62.4%	77.5	-11.9%	20.3	-43.1%

Table 4.4: Attack success rate of attacks from literature, including original TextFooler (Jin et al., 2019), BAE (Garg and Ramakrishnan, 2020), PWWS (Ren et al., 2019), and PSO (Zang et al., 2020). Δ% column represents the percent change between natural training and the different training methods.

Model	Method	IMDB		MR		Yelp	
		Standard Accuracy	Yelp Accuracy	Standard Accuracy	Yelp Accuracy	Standard Accuracy	IMDB Accuracy
BERT	Natural	93.97	92.13	85.40	**90.60**	96.34	88.31
	`Fast-TextFooler`	**94.49**	**92.50**	**85.71**	88.45	**96.68**	**89.24**
	`Fast-BAE`	93.05	90.67	83.80	85.32	95.85	85.01
	SSMBA	93.94	91.59	85.33	89.49	96.28	88.54
	Backtranslation	93.97	91.73	85.63	89.46	96.46	88.77
RoBERTa	Natural	95.26	94.09	87.52	93.42	97.26	**91.94**
	`Fast-TextFooler`	**96.57**	94.41	**88.03**	93.45	**97.45**	91.86
	`Fast-BAE`	94.71	**94.48**	86.49	92.93	96.84	90.44
	SSMBA	95.25	94.11	86.46	93.03	97.16	91.90
	Backtranslation	95.31	93.84	87.78	**93.77**	97.25	91.76

Table 4.5: Accuracy on in-domain and out-of-domain datasets. We can see that adversarial training can helps model outperform both naturally trained models and models trained using data augmentation methods.

Model	Training Method	IMDB	MR	Yelp
BERT	Natural	7.78	33.43	12.78
	`Fast-TextFooler`	**10.74**	**34.25**	**13.18**
	`Fast-BAE`	9.12	32.17	11.14
	SSMBA	7.21	32.19	10.94
	Backtranslation	6.02	32.21	11.10
RoBERTa	Natural	0.35	0.39	-1.09
	`Fast-TextFooler`	**0.42**	0.01	-1.01
	`Fast-BAE`	0.09	**0.45**	-1.13
	SSMBA	0.26	-0.12	**-0.43**
	Backtranslation	-0.04	0.05	-1.06

Table 4.6: AOPC scores of the LIME explanations for each model. Higher AOPC scores indicates that the model is more interpretable.

4.6 Discussions

4.6.1 `Fast-TextFooler` vs `Fast-BAE` attack

We can see that model trained using `Fast-TextFooler` attack outperforms the model trained using `Fast-BAE` attack in standard accuracy and cross-domain accuracy in all but one case. This suggests that using counter-fitted embeddings can generate higher quality adversarial examples than masked language models. Since masked language models are only trained to predict words that are statistically most likely to appear, it is likely that it will propose words that do change the semantics of the text entirely. In fact, we observed that once a bad word replacement is proposed, it is often difficult to filter them out using constraints. An interesting direction for future work is to study whether masked language models can be used to generate only synonyms for fewer false positive adversarial examples.

Chapter 5

Conclusion

In this work, we have presented empirical evidence that adversarial training using carefully designed perturbations to the input space can improve model's adversarial robustness, standard accuracy, cross-domain generalization, and interpretability. We also have demonstrated that the improvement in adversarial robustness from adversarial training is transferable against other attacks from literature. To build adversarial attack that is feasible for adversarial training, we also have presented a comprehensive benchmarking of the many search algorithms used for adversarial attacks in NLP. These new findings demonstrate that adversarial training can be used to defend the model against many different attacks without worrying about potential trade-offs.

An interesting area of future work is to perform more in-depth investigation of the hyper-parameters for adversarial training - mainly γ, which controls how many adversarial examples we generate per epoch, and α, which controls the strength of adversarial loss. Another area of future work is to compare our adversarial training method with ones that perform embedding-level perturbations.

References

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649.

Naveed Akhtar and Ajmal Mian. 2018. Threat of adversarial attacks on deep learning in computer vision: A survey. *CoRR*, abs/1801.00553.

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating natural language adversarial examples. *arXiv preprint arXiv:1804.07998*.

Petr Bělohlávek, Ondřej Plátek, Zdeněk Žabokrtský, and Milan Straka. 2018. Using adversarial examples in natural language processing. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

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

Nicholas Carlini and David A. Wagner. 2016. Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Universal sentence encoder. *CoRR*, abs/1803.11175.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. 2013. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005.

Hanjie Chen and Yangfeng Ji. 2020. Learning variational word masks to improve the interpretability of neural text classifiers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4236–4251, Online. Association for Computational Linguistics.

Minhao Cheng, Jinfeng Yi, Huan Zhang, Pin-Yu Chen, and Cho-Jui Hsieh. 2018. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. *CoRR*, abs/1803.01128.

J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Zhendong Dong, Qiang Dong, and Changling Hao. 2010. Hownet and its computation of meaning. In *Proceedings of the 23rd International Conference on Computational Linguistics: Demonstrations*, COLING '10, page 53–56, USA. Association for Computational Linguistics.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2017. Hotflip: White-box adversarial examples for text classification. In *ACL*.

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

Siddhant Garg and Goutham Ramakrishnan. 2020. Bae: Bert-based adversarial examples for text classification.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. SMART: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2177–2190, Online. Association for Computational Linguistics.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2019. Is bert really robust? natural language attack on text classification and entailment. *ArXiv*, abs/1907.11932.

Harini Kannan, Alexey Kurakin, and Ian J. Goodfellow. 2018. Adversarial logit pairing. *CoRR*, abs/1803.06373.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2016a. Adversarial examples in the physical world. *CoRR*, abs/1607.02533.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2016b. Adversarial machine learning at scale. *CoRR*, abs/1611.01236.

Qi Lei, Lingfei Wu, Pin-Yu Chen, Alex Dimakis, Inderjit S. Dhillon, and Michael J Witbrock. 2019. Discrete adversarial attacks and submodular optimization with applications to text classification. In *Proceedings of Machine Learning and Systems 2019*, pages 146–165.

Dianqi Li, Yizhe Zhang, Hao Peng, Liqun Chen, Chris Brockett, Ming-Ting Sun, and Bill Dolan. 2021. Contextualized perturbation for textual adversarial attack.

Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020. Bert-attack: Adversarial attack against bert using bert.

Xiaodong Liu, Hao Cheng, Pengcheng He, Weizhu Chen, Yu Wang, Hoifung Poon, and Jianfeng Gao. 2020a. Adversarial training for large neural language models.

Xiaodong Liu, Hao Cheng, Pengcheng He, Weizhu Chen, Yu Wang, Hoifung Poon, and Jianfeng Gao. 2020b. Adversarial training for large neural language models.

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

Ilya Loshchilov and Frank Hutter. 2017. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

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

George A. Miller. 1995. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41.

Takeru Miyato, Shin ichi Maeda, Masanori Koyama, and Shin Ishii. 2018. Virtual adversarial training: A regularization method for supervised and semi-supervised learning.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2015. Deepfool: a simple and accurate method to fool deep neural networks. *CoRR*, abs/1511.04599.

John Morris, Eli Lifland, Jin Yong Yoo, and Yanjun Qi. 2020a. TextAttack: A framework for adversarial attacks in natural language processing. *ArXiv*, abs/2005.05909.

John X. Morris, Eli Lifland, Jack Lanchantin, Yangfeng Ji, and Yanjun Qi. 2020b. Reevaluating adversarial examples in natural language.

Nikola Mrksic, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gasic, Lina Maria Rojas-Barahona, Pei hao Su, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. 2016. Counter-fitting word vectors to linguistic constraints. In *HLT-NAACL*.

Nathan Ng, Kyunghyun Cho, and Marzyeh Ghassemi. 2020. SSMBA: Self-supervised manifold based data augmentation for improving out-of-domain robustness. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1268–1283, Online. Association for Computational Linguistics.

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook fair's wmt19 news translation task submission. In *Proc. of WMT*.

Dong Nguyen. 2018. Comparing automatic and human evaluation of local explanations for text classification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1069–1078, New Orleans, Louisiana. Association for Computational Linguistics.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2020. Adversarial NLI: A new benchmark for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 115–124, Ann Arbor, Michigan. Association for Computational Linguistics.

Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2015. The limitations of deep learning in adversarial settings. *CoRR*, abs/1511.07528.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Danish Pruthi, Mansi Gupta, Bhuwan Dhingra, Graham Neubig, and Zachary C. Lipton. 2020. Learning to deceive with attention-based explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4782–4793, Online. Association for Computational Linguistics.

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

Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John C. Duchi, and Percy Liang. 2019. Adversarial training can hurt generalization. *CoRR*, abs/1906.06032.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1085–1097, Florence, Italy. Association for Computational Linguistics.

Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938.

W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K. Müller. 2017. Evaluating the visualization of what a deep neural network has learned. *IEEE Transactions on Neural Networks and Learning Systems*, 28(11):2660–2673.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Ali Shafahi, W. Ronny Huang, Christoph Studer, Soheil Feizi, and Tom Goldstein. 2018. Are adversarial examples inevitable? *CoRR*, abs/1809.02104.

Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John P. Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. 2019. Adversarial training for free! *CoRR*, abs/1904.12843.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations*.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. 2019. Robustness may be at odds with accuracy.

Eric Wallace, Jens Tuyls, Junlin Wang, Sanjay Subramanian, Matt Gardner, and Sameer Singh. 2019. AllenNLP Interpret: A framework for explaining predictions of NLP models. In *Empirical Methods in Natural Language Processing*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In the Proceedings of ICLR.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Cihang Xie, Mingxing Tan, Boqing Gong, Alan Yuille, and Quoc V. Le. 2020. Smooth adversarial training.

Qizhe Xie, Zihang Dai, Eduard H. Hovy, Minh-Thang Luong, and Quoc V. Le. 2019. Unsupervised data augmentation. *CoRR*, abs/1904.12848.

Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. 2020. Word-level textual adversarial attacking as combinatorial optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6066–6080, Online. Association for Computational Linguistics.

Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. 2019a. Theoretically principled trade-off between robustness and accuracy. *CoRR*, abs/1901.08573.

Huangzhao Zhang, Hao Zhou, Ning Miao, and Lei Li. 2019b. Generating fluent adversarial examples for natural languages. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5564–5569, Florence, Italy. Association for Computational Linguistics.

Tianyi Zhang*, Varsha Kishore*, Felix Wu*, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*.

Wei Emma Zhang, Quan Z. Sheng, and Ahoud Abdulrahmn F. Alhazmi. 2019c. Generating textual adversarial examples for deep learning models: A survey. *CoRR*, abs/1901.06796.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 649–657. Curran Associates, Inc.

Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. 2019. Freelb: Enhanced adversarial training for language understanding. *CoRR*, abs/1909.11764.

Appendix A

A.1 Search Benchmark Figures for LSTM Models

Figures A.1 shows how the number of words in the input affects runtime for each algorithm against LSTM models. Figure A.2 shows the attack success rate of each search algorithm as the maximum number of queries permitted to perturb a single sample varies from 0 to $20{,}000$ for Yelp dataset and 0 to 3000 for MR and SNLI.

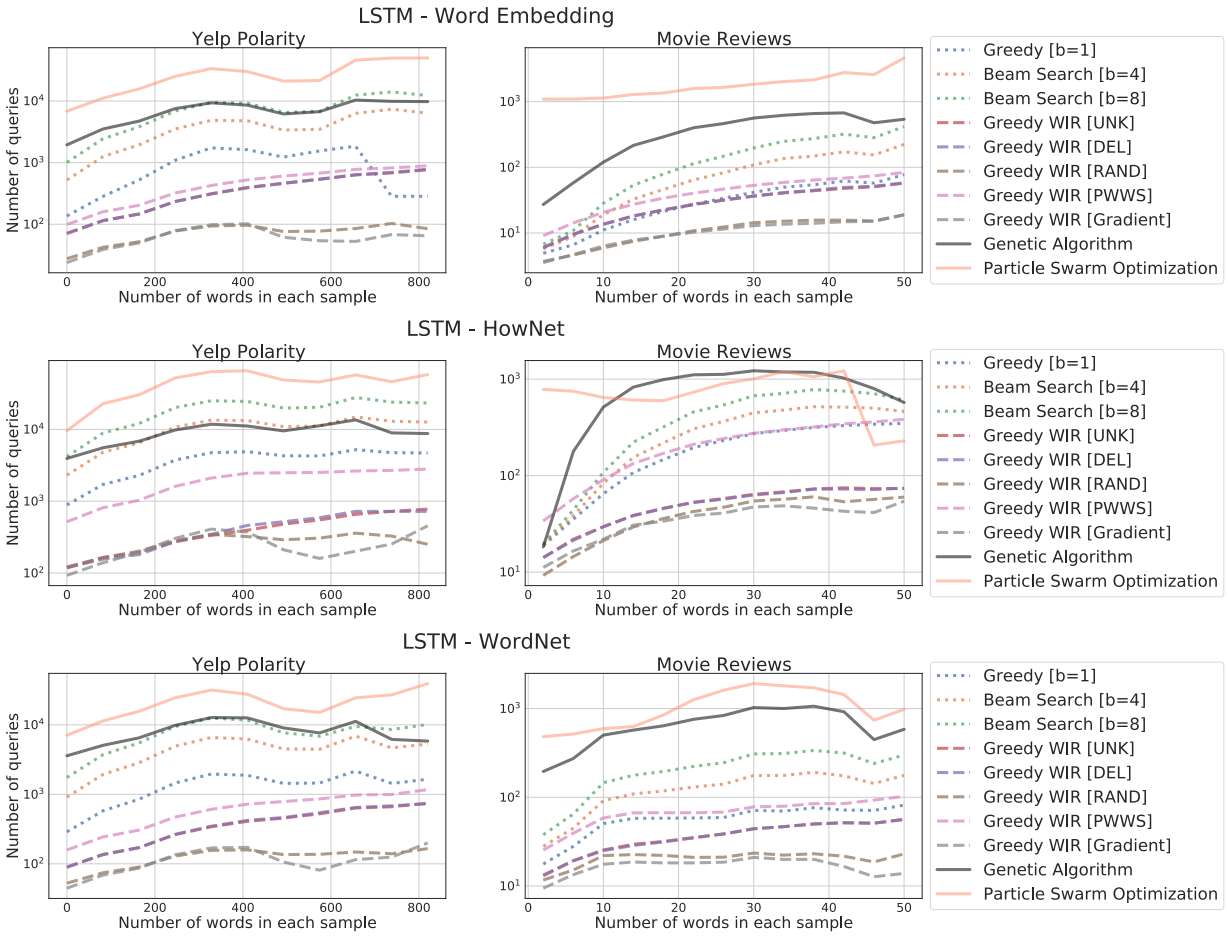

Figure A.1: Number of queries vs. length of input text.

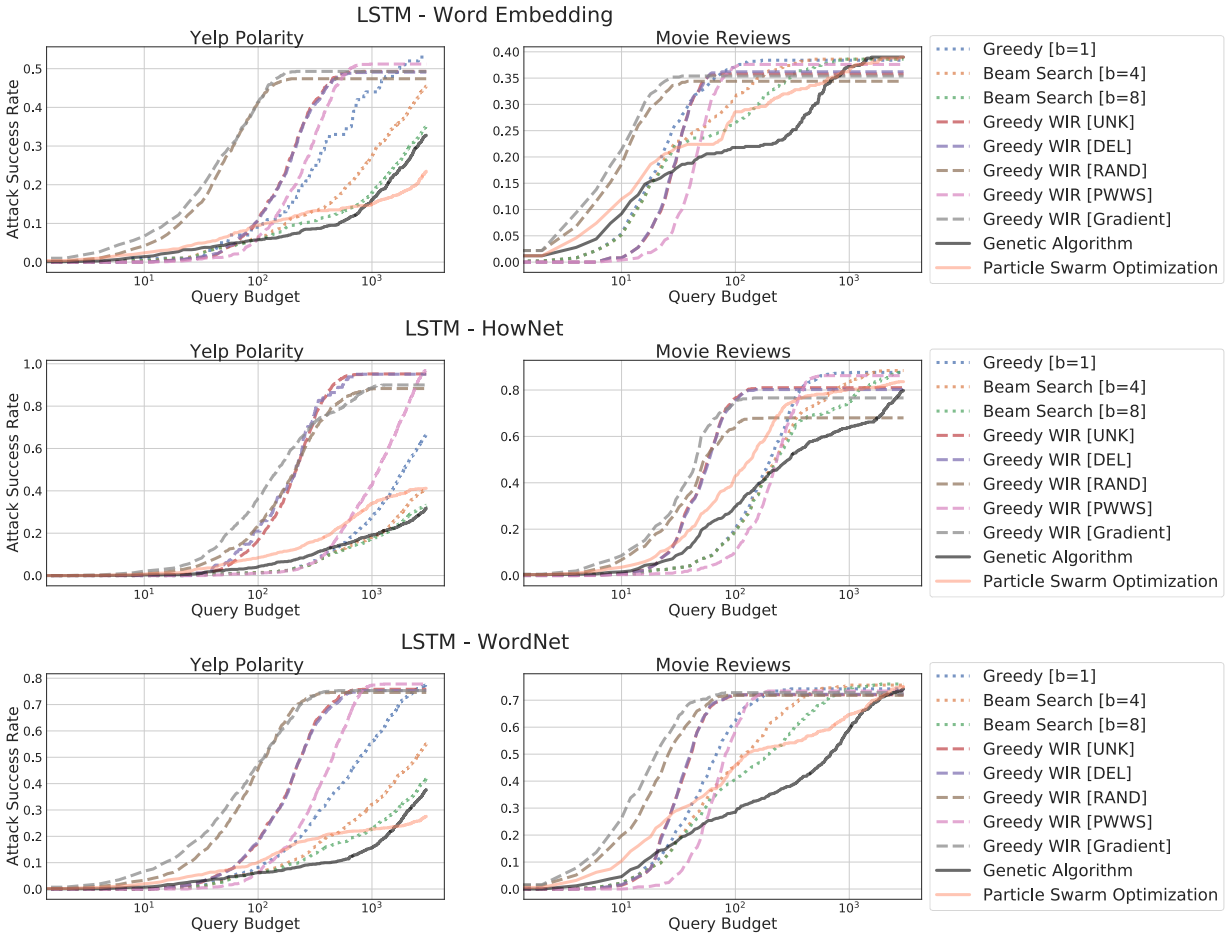

Figure A.2: Attack success rate by query budget for each search algorithm and dataset.