Self-Supervised Contrastive Learning with Adversarial Perturbations for Robust Pretrained Language Models

Anonymous ACL submission

Abstract

In this paper, we present an approach to improve the robustness of BERT language models against word substitution-based adversarial attacks by leveraging adversarial perturbations for self-supervised contrastive learning. We create an efficient word-level adversarial attack, and use it to finetune BERT on adversarial examples generated on the fly during training. In contrast with previous works, our method improves model robustness without using any labeled data. Experimental results show that our method improves robustness of BERT against four different word substitution-based adversarial attacks, and combining our method with adversarial training gives higher robustness than adversarial training alone. As our method improves the robustness of BERT purely with unlabeled data, it opens up the possibility of using large text datasets to train robust language models.

1 Introduction

Pretrained language models such as BERT (Devlin et al., 2019, inter alia) have had a tremendous impact on many NLP tasks. However, several researchers have demonstrated that these models are vulnerable to adversarial attacks, which fool the model by adding small perturbations to the model input (Jia and Liang, 2017).

A prevailing method to improve model robustness against adversarial attacks is adversarial training (Madry et al., 2018) In NLP, adversarial training in the input space has been challenging, as existing natural language adversarial attacks are too slow to generate adversarial examples on the fly during training (Alzantot et al., 2018; Ebrahimi et al., 2018; Ren et al., 2019). While some recent work (Wang et al., 2021b) have started exploring efficient input space adversarial training (e.g., for text classification), scaling adversarial training to pretrained language models like BERT has been challenging.

In this work, we propose an approach to adversarially finetune BERT-like models without using any labeled data. In order to achieve this, we rely on self-supervised contrastive learning (Chen et al., 2020). Self-supervised contrastive learning has recently gained attention in the community and contrastive learning has been used to learn better representations for text classification (Giorgi et al., 2021; Kim et al., 2021; Gao et al., 2021). However, how to use these methods to improve model robustness remains an open question.

We combine self-supervised contrastive learning with adversarial perturbations by using adversarial attacks to generate hard positive examples for contrastive learning. To efficiently create adversarial examples, we leverage an adversarial attack, that is capable of generating multiple adversarial examples in parallel. The attack adversarially creates hard positive examples for contrastive learning by iteratively replacing words to follow the direction of the contrastive loss (see fig. 2).

Experiments show that our method can improve the robustness of pretrained language models without looking at the labels. Additionally, by combining our method with adversarial training, we are able to obtain better robustness than conducting adversarial training alone (see section 4.5). Our study of the vector representations of clean examples and their corresponding adversarial examples indeed explains that our method improves model robustness by pulling clean examples and adversarial examples closer.

Our contribution in this paper are two-fold. On the one hand, we improve the robustness of the pretrained language model BERT by using self-supervised contrastive learning with adversarial perturbations (see section 3.2). On the other hand, we create for BERT a word-level adversarial attack.

1In contrast, previous attacks generate adversarial examples one by one (Alzantot et al., 2018; Ebrahimi et al., 2018; Ren et al., 2019).
which makes adversarial training with on-the-fly generated adversarial examples efficient (see section 3.3 and see appendix D for speed comparisons). Additionally, we also show that our method is capable of using out-of-domain data to improve model robustness (see table 2 and section 4.5). This opens an opportunity for using large-scale unlabeled data to train robust language models.

2 Related Work

2.1 Adversarial Training for NLP

Adversarial training improves model robustness by augmenting clean examples with adversarial examples during training. Previous works on adversarial training for natural language mainly focus on perturbations in the vector space, while actual adversarial attacks create adversarial examples by changing natural language symbols. For example, Zhu et al. (2020) and Liu et al. (2020) improve model generalization ability by adversarial training on the word embedding space, without mentioning model robustness. However, they either ignore model robustness, or only test model robustness against the adversarial dataset ANLI, without paying attention to actual adversarial attacks. Other works conduct adversarial training in the word space (Alzantot et al., 2018; Ren et al., 2019). Still, they can only do adversarial training on a limited number of pre-generated adversarial examples due to the low efficiency of the attacks. A recent work (Wang et al., 2021b) conducts adversarial training efficiently in the word space, but their method is limited to non-contextualized models.

Our work also differs from previous works in natural language adversarial training. On the one hand, as opposed to previous works, which are supervised, we propose a self-supervised learning scheme to improve the robustness of pretrained language models. On the other hand, while previous works mostly focus on adversarial training in embedding space, we conduct efficient adversarial training with pretrained language models at the word level.

2.2 Contrastive Learning for NLP

Contrastive learning was first proposed in the image domain to improve model performance in a self-supervised fashion (He et al., 2020; Chen et al., 2020). These methods bring representations of similar examples closer and push representations of dissimilar examples further apart. Additionally, researchers also find that by adding adversarial perturbations during contrastive learning, image classification models become more robust against adversarial attacks (Kim et al., 2020).

In NLP, previous works on contrastive learning mainly focus on improving model generalization. Gunel et al. (2021) boost performance of RoBERTa by adding supervised signals during fine-tuning on downstream tasks. Lee et al. (2021) tackle the “exposure bias” problem in text generation by adding adversarial signals during contrastive learning. Other similar works include Pan et al. (2021), Giorgi et al. (2021), and Gao et al. (2021). Although these works have demonstrated the usefulness of contrastive learning in NLP applications, few have addressed the robustness of NLP models, particularly pretrained language models, against natural language adversarial attacks. Recently, Wang et al. (2021a) claimed that their method improves model robustness against adversarial sets. However, such sets are pre-generated and are less challenging than adversarial examples generated on the fly by actual adversarial attacks (Jin et al., 2020; Ren et al., 2019). In this paper, we focus on improving the robustness of pretrained language models against word substitution-based adversarial attacks. We present the details of our method in section 3.

3 Methodology

In this section, we describe our method for self-supervised contrastive learning with adversarial perturbations. Specifically, section 3.1 gives the background and motivation of our problem, and section 3.2 describes the adversarial contrastive learning framework. Finally, in section 3.3, we describe the adversarial attack used in contrastive learning.

3.1 Background and Motivation

Imagine we have a batch of \( N \) samples: \( \{(X_1, y_1), (X_2, y_2), \ldots, (X_N, y_N)\} \), where \( X_i = \{w_1, w_2, \ldots, w_L\} \) is an example text consisting of \( L \) words and \( y_i \) is the corresponding label. Let \( X_i \) be the current input to encoder \( f(\cdot) \), \( c(\cdot) \) be the softmax classification layer, and \( a(\cdot) \) be a natural language adversarial attack. Thus, we have:

\[
X'_i = a(X_i), h_i = f(X_i), h'_i = f(X'_i), \hat{y}_i = c(h_i), \hat{y}'_i = c(h'_i),
\]

where \( h_i, h'_i \in \mathbb{R}^d \) are fixed-size representations of \( X_i \) and \( X'_i \), and \( \hat{y}_i \) and \( \hat{y}'_i \) are predicted labels of \( X_i \) and \( X'_i \), respectively.
Assuming the attack successfully fools the model, we have \( \hat{y}_i \neq \hat{y}_i' \). Our assumption is that although \( X_i \) and \( X_i' \) are very similar to each other on the word level, the distance of their representations \( h_i \) and \( h_i' \) are larger such that the softmax classifier \( c(\cdot) \) predicts \( X_i \) and \( X_i' \) to be of different classes.

To obtain a robust model, we optimize the encoder such that \( h_i \) and \( h_i' \) become similar to each other. We achieve this goal by conducting self-supervised contrastive learning with adversarial perturbations, during which we use an attack to create hard positive examples, maximizing the contrastive loss.

### 3.2 Self-Supervised Contrastive Learning with Adversarial Perturbations

Following previous works on self-supervised contrastive learning (He et al., 2020; Chen et al., 2020), we formulate our learning objectives as follows. Consider we have a batch of \( n \) examples and \( X_i \) is the \( i \)-th input, we first obtain \( X_i' = t(X_i) \) as an augmentation of \( X_i \) by transformation \( t(\cdot) \). We call \( X_i \) and \( X_i' \) a pair of positive examples. All other examples in the same batch are considered negative examples of \( X_i \) and \( X_i' \).

To take advantage from using more negative examples, we use MoCo (He et al., 2020) as our framework, in which we employ an encoder \( f_q \) for the positive examples, and another momentum encoder \( f_k \) for the negative examples. We then have \( h_i = f_q(X_i) \) and \( h_i' = f_k(X_i') \), where \( h_i, h_i' \in \mathbb{R}^d \) are representations of \( X_i \) and \( X_i' \), respectively. During training, \( f_q \) and \( f_k \) are initialized the same. We update \( f_k \) momentarily by \( \theta_k \leftarrow m \cdot \theta_k + (1 - m) \cdot \theta_q \), where \( \theta_k \) and \( \theta_q \) denote the parameters of \( f_k \) and \( f_q \), respectively. We also use \( g_q(\cdot) \) and \( g_k(\cdot) \) to map \( h_i \) and \( h_i' \) to \( z_i, z_i' \in \mathbb{R}^c \), respectively, where \( g_q(\cdot) \) and \( g_k(\cdot) \) are MLPs with one hidden layer of sigmoid activation.

Contrastive learning pulls the clean example \( z_i \) and the hard positive example \( z_i' \) together. (c) After contrastive learning, the clean example \( z_i \), the hard positive example \( z_i' \), and the adversarial example \( z_{i\text{adv}} \) are close. We omit MLP in this figure for simplicity. We use a different color to show another example from the dataset. See section 3 for details. Note that the adversarial example \( X_{i\text{adv}} \) and its corresponding vector \( z_{i\text{adv}} \) are not used in contrastive learning. We nevertheless show \( X_{i\text{adv}} \) and \( z_{i\text{adv}} \) for illustration purposes.

### 3.2 Self-Supervised Contrastive Learning with Adversarial Perturbations

Following previous works on self-supervised contrastive learning (He et al., 2020; Chen et al., 2020), we formulate our learning objectives as follows. Consider we have a batch of \( n \) examples and \( X_i \) is the \( i \)-th input, we first obtain \( X_i' = t(X_i) \) as an augmentation of \( X_i \) by transformation \( t(\cdot) \). We call \( X_i \) and \( X_i' \) a pair of positive examples. All other examples in the same batch are considered negative examples of \( X_i \) and \( X_i' \).

To take advantage from using more negative examples, we use MoCo (He et al., 2020) as our framework, in which we employ an encoder \( f_q \) for the positive examples, and another momentum encoder \( f_k \) for the negative examples. We then have \( h_i = f_q(X_i) \) and \( h_i' = f_k(X_i') \), where \( h_i, h_i' \in \mathbb{R}^d \) are representations of \( X_i \) and \( X_i' \), respectively. During training, \( f_q \) and \( f_k \) are initialized the same. We update \( f_k \) momentarily by \( \theta_k \leftarrow m \cdot \theta_k + (1 - m) \cdot \theta_q \), where \( \theta_k \) and \( \theta_q \) denote the parameters of \( f_k \) and \( f_q \), respectively. We also use \( g_q(\cdot) \) and \( g_k(\cdot) \) to map \( h_i \) and \( h_i' \) to \( z_i, z_i' \in \mathbb{R}^c \), respectively, where \( g_q(\cdot) \) and \( g_k(\cdot) \) are MLPs with one hidden layer of sigmoid activation.

Contrastive learning pulls the clean example \( z_i \) and the hard positive example \( z_i' \) together. (c) After contrastive learning, the clean example \( z_i \), the hard positive example \( z_i' \), and the adversarial example \( z_{i\text{adv}} \) are close. We omit MLP in this figure for simplicity. We use a different color to show another example from the dataset. See section 3 for details. Note that the adversarial example \( X_{i\text{adv}} \) and its corresponding vector \( z_{i\text{adv}} \) are not used in contrastive learning. We nevertheless show \( X_{i\text{adv}} \) and \( z_{i\text{adv}} \) for illustration purposes.

### 3.2 Self-Supervised Contrastive Learning with Adversarial Perturbations

Following previous works on self-supervised contrastive learning (He et al., 2020; Chen et al., 2020), we formulate our learning objectives as follows. Consider we have a batch of \( n \) examples and \( X_i \) is the \( i \)-th input, we first obtain \( X_i' = t(X_i) \) as an augmentation of \( X_i \) by transformation \( t(\cdot) \). We call \( X_i \) and \( X_i' \) a pair of positive examples. All other examples in the same batch are considered negative examples of \( X_i \) and \( X_i' \).

To take advantage from using more negative examples, we use MoCo (He et al., 2020) as our framework, in which we employ an encoder \( f_q \) for the positive examples, and another momentum encoder \( f_k \) for the negative examples. We then have \( h_i = f_q(X_i) \) and \( h_i' = f_k(X_i') \), where \( h_i, h_i' \in \mathbb{R}^d \) are representations of \( X_i \) and \( X_i' \), respectively. During training, \( f_q \) and \( f_k \) are initialized the same. We update \( f_k \) momentarily by \( \theta_k \leftarrow m \cdot \theta_k + (1 - m) \cdot \theta_q \), where \( \theta_k \) and \( \theta_q \) denote the parameters of \( f_k \) and \( f_q \), respectively. We also use \( g_q(\cdot) \) and \( g_k(\cdot) \) to map \( h_i \) and \( h_i' \) to \( z_i, z_i' \in \mathbb{R}^c \), respectively, where \( g_q(\cdot) \) and \( g_k(\cdot) \) are MLPs with one hidden layer of sigmoid activation.

Contrastive learning pulls the clean example \( z_i \) and the hard positive example \( z_i' \) together. (c) After contrastive learning, the clean example \( z_i \), the hard positive example \( z_i' \), and the adversarial example \( z_{i\text{adv}} \) are close. We omit MLP in this figure for simplicity. We use a different color to show another example from the dataset. See section 3 for details. Note that the adversarial example \( X_{i\text{adv}} \) and its corresponding vector \( z_{i\text{adv}} \) are not used in contrastive learning. We nevertheless show \( X_{i\text{adv}} \) and \( z_{i\text{adv}} \) for illustration purposes.

### 3.2 Self-Supervised Contrastive Learning with Adversarial Perturbations

Following previous works on self-supervised contrastive learning (He et al., 2020; Chen et al., 2020), we formulate our learning objectives as follows. Consider we have a batch of \( n \) examples and \( X_i \) is the \( i \)-th input, we first obtain \( X_i' = t(X_i) \) as an augmentation of \( X_i \) by transformation \( t(\cdot) \). We call \( X_i \) and \( X_i' \) a pair of positive examples. All other examples in the same batch are considered negative examples of \( X_i \) and \( X_i' \).

To take advantage from using more negative examples, we use MoCo (He et al., 2020) as our framework, in which we employ an encoder \( f_q \) for the positive examples, and another momentum encoder \( f_k \) for the negative examples. We then have \( h_i = f_q(X_i) \) and \( h_i' = f_k(X_i') \), where \( h_i, h_i' \in \mathbb{R}^d \) are representations of \( X_i \) and \( X_i' \), respectively. During training, \( f_q \) and \( f_k \) are initialized the same. We update \( f_k \) momentarily by \( \theta_k \leftarrow m \cdot \theta_k + (1 - m) \cdot \theta_q \), where \( \theta_k \) and \( \theta_q \) denote the parameters of \( f_k \) and \( f_q \), respectively. We also use \( g_q(\cdot) \) and \( g_k(\cdot) \) to map \( h_i \) and \( h_i' \) to \( z_i, z_i' \in \mathbb{R}^c \), respectively, where \( g_q(\cdot) \) and \( g_k(\cdot) \) are MLPs with one hidden layer of sigmoid activation.

Contrastive learning pulls the clean example \( z_i \) and the hard positive example \( z_i' \) together. (c) After contrastive learning, the clean example \( z_i \), the hard positive example \( z_i' \), and the adversarial example \( z_{i\text{adv}} \) are close. We omit MLP in this figure for simplicity. We use a different color to show another example from the dataset. See section 3 for details. Note that the adversarial example \( X_{i\text{adv}} \) and its corresponding vector \( z_{i\text{adv}} \) are not used in contrastive learning. We nevertheless show \( X_{i\text{adv}} \) and \( z_{i\text{adv}} \) for illustration purposes.
attack described in section 3.3 to create pairs of examples that are similar on the word level but at the same time are distant from each other in the representation space.

We illustrate method in fig. 1. In fig. 1 (b) and (c), by conducting contrastive learning and using the Geometry Attack generated adversarial examples as hard positives, the vector representations obtained from the model become invariant to the adversarial attacks.

![Figure 2: An illustration of one iteration in Geometry Attack for contrastive loss. See section 3.3 for details.](image)

### 3.3 Creating Hard Positive Examples by Geometry Attack

We describe how we create adversarial examples for contrastive loss during self-supervised contrastive learning (see fig. 1 (b)). Inspired by Meng and Wattenhofer (2020), who leverage geometry of representations to generate natural language adversarial examples for text classification tasks, we also use the geometry of pretrained representations to create adversarial examples for contrastive loss. The created adversarial examples are used as positive examples of the original examples in our contrastive learning framework, and at the same time are created to maximize the contrastive loss. Hence, we refer to adversarial examples created by the attack as hard positive examples.

The intuition of our attack is that we repeatedly replace words in the original texts such that in each iteration, the replaced word increases the contrastive loss as much as possible. To be specific, consider an example $X_t$, we then have:

1. **Determine Direction for Sentence Vector**
   Compute the gradients of $\ell_i$ with respect to $z_i$. In this step, we find the direction we should move from $z_i$ to increase the contrastive loss $\ell_i$. We have the gradient vector $v_{z_i} = \nabla z_i \ell_i$.

2. **Choose Original Word to be Replaced**
   Compute the gradients of $\ell_i$ with respect to input word embeddings of $X_t$. For words tokenized into multiple tokens, we take the average of the gradients of the tokens. In this step, we find the word $w_{t}$ which has the most influence in computing $\ell_i$.

Specifically, assuming we have $L$ words, then we choose $t = \arg \max_t \{||g_1||, ||g_2||, \ldots, ||g_L||\}$, where $g_k$ is the gradients of $\ell_i$ with respect to the embeddings of word $w_k, 1 \leq k \leq L$.

3. **Generate Candidate Set**
   Suppose we choose the word $w_t$ in step 2. In this step, we use a pre-trained BERT to choose the most probable candidates $w_t$ to replace it in the original text. We have the candidates set $= \{w_{t1}, w_{t2}, \ldots, w_{tT}\}$. Following Jin et al. (2020), we filter out semantically different words from the candidate set by discarding candidate words of which the cosine similarity of their embeddings are below a threshold $\epsilon$.

4. **Choose Replacement Word**
   Replace $w_t$ with words in the candidates set, resulting in text vectors $\{z_{i1}, z_{i2}, \ldots, z_{iT}\}$. We compute delta vector $r_{ij} \leftarrow z_{ij} - z_i$. The projection of $r_{ij}$ onto $v_{z_i}$ is: $p_{ij} \leftarrow \frac{r_{ij} \cdot v_{z_i}}{|v_{z_i}|}$. We select word $w_{tm}$, where $m = \arg \max_j |p_{ij}|$.

5. **Repetition**
   Replace $w_t$ with $w_{tm}$ in $X_t$, then we have $z_i \leftarrow z_{im}$. Repeat step 1-4 for $N$ iterations, where $N$ is a hyperparameter of our method. We expect $\ell_i$ to increase in each iteration.

Figure 2 illustrates an iteration of our attack, in which we have two options to choose from the candidate set. This attack can be easily implemented in a batched fashion, making it possible for us to generate adversarial examples on the fly during training. Furthermore, our attack is fast, which makes it possible to conduct contrastive learning with adversarial perturbations as well as adversarial training with adversarial examples generated on the fly. We give a speed comparison of our attack and other attacks in appendix D. We also give pseudocode of the attack in algorithm 1 of appendix A.

### 4 Experiments

#### 4.1 Datasets

We test how our method improves model robustness on four text classification datasets: AG’s News, Yelp, IMDB, and DBpedia (See appendix B for details).

#### 4.2 Evaluation Metrics

We report the attack success rate and the replacement rate of the attacks as the evaluation metrics. To prevent the model accuracy on clean examples from confounding the results, we define the success
rate of an attack on all correctly classified examples in the test set. Lower success rates indicate higher robustness. The replacement rate refers to the percent of original words replaced in the clean example. Higher replacement rates indicate that the attack needs to replace more words to fool the model, and thus means that the model is more robust.

4.3 Adversarial Attacks for Evaluating Robustness

We use four word substitution-based adversarial attacks to evaluate the model robustness. Geometry Attack We use the same attack described in section 3.3 to generate adversarial examples for sentence classification tasks by replacing contrastive loss with cross-entropy classification loss. We set the maximum number of replaced words to 20.

TextFooler, PWWS, and BAE-R We use the default implementations from TextAttack (Morris et al., 2020).

All these attacks will give up and terminate once the maximum number of replaced words (sometimes also called perturbation budget) is reached.

4.4 Experimental Designs

We have the following hypotheses for our approach:

H1: Self-supervised contrastive learning improves model robustness against adversarial attacks. Moreover, using adversarial perturbations during contrastive learning further improves robustness.

H2: Combining self-supervised contrastive learning with adversarial training gives higher robustness than conducting adversarial training alone.

H3: Our contrastive learning method is capable of using out-of-domain data to improve the model robustness.

H4: By optimizing eq. (1), our method pulls the representations of the clean samples and their corresponding hard positive examples closer in the vector space while pushing other different examples further. In this way, the representations of clean examples and their adversarial examples are also closer in the vector space.

We validate this hypothesis by conducting a vector space study. See section 4.5 for details.

Note that to avoid confusing adversarial examples generated during contrastive learning and adversarial examples generated during finetuning, we refer to the former as hard positive examples (see section 3.3).

4.5 Results

Table 1 shows the experimental results for validating H1 and H2. For each dataset, when evaluating the model robustness, we use the same perturbation budget across different settings. Note that although the replacement rates vary across different settings of the same dataset, the perturbation budget for the same attack is the same in these settings. By using the same perturbation budget, we ensure that the success rates of the attacks provide us with a fair evaluation of the robustness of the model (Wang et al., 2021b; Ren et al., 2019).
Table 1: Experimental results for H1 and H2. In-Domain setting is used. We **bold** the best results, while the second best is in *italic*.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Pretrain</th>
<th>Finetune</th>
<th>Acc. (%)</th>
<th>Success Rate (%)</th>
<th>Replaced (%)</th>
<th>Geometry</th>
<th>TextFooler</th>
<th>PWWS</th>
<th>BAE-R</th>
<th>Geometry</th>
<th>TextFooler</th>
<th>PWWS</th>
<th>BAE-R</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG</td>
<td>NP</td>
<td>FTC</td>
<td>94.2</td>
<td>96.2</td>
<td>87.6</td>
<td>63.6</td>
<td>17.9</td>
<td>19.6</td>
<td>25.7</td>
<td>20.9</td>
<td>18.4</td>
<td>7.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCL</td>
<td>FTC</td>
<td>94.4</td>
<td>20.7</td>
<td>25.1</td>
<td>26.1</td>
<td>10.7</td>
<td>20.5</td>
<td>29.3</td>
<td>22.3</td>
<td>21.2</td>
<td>7.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BTCL</td>
<td>FTC</td>
<td>94.4</td>
<td>80.6</td>
<td>84.6</td>
<td>63.1</td>
<td>17.7</td>
<td>18.1</td>
<td>24.6</td>
<td>20.9</td>
<td>17.9</td>
<td>7.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCL</td>
<td>FTC</td>
<td>94.3</td>
<td>76.5</td>
<td>80.7</td>
<td>55.9</td>
<td>14.1</td>
<td>19.1</td>
<td>26.7</td>
<td>22.6</td>
<td>16.2</td>
<td>7.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ADV</td>
<td>94.4</td>
<td>18.7</td>
<td>23.5</td>
<td>24.7</td>
<td>9.7</td>
<td>20.6</td>
<td>29.3</td>
<td>22.2</td>
<td>21.2</td>
<td>7.2</td>
<td></td>
</tr>
<tr>
<td>Yelp</td>
<td>NP</td>
<td>FTC</td>
<td>97.1</td>
<td>94.6</td>
<td>94.3</td>
<td>97.0</td>
<td>42.1</td>
<td>10.6</td>
<td>10.4</td>
<td>7.4</td>
<td>12.8</td>
<td>11.3</td>
<td>8.8</td>
</tr>
<tr>
<td></td>
<td>ADCL</td>
<td>FTC</td>
<td>96.4</td>
<td>38.8</td>
<td>52.4</td>
<td>62.7</td>
<td>22.2</td>
<td>11.0</td>
<td>10.1</td>
<td>7.7</td>
<td>12.8</td>
<td>11.3</td>
<td>8.8</td>
</tr>
<tr>
<td></td>
<td>BTCL</td>
<td>FTC</td>
<td>97.1</td>
<td>92.3</td>
<td>91.6</td>
<td>94.8</td>
<td>39.2</td>
<td>10.4</td>
<td>10.5</td>
<td>7.4</td>
<td>13.4</td>
<td>11.2</td>
<td>8.2</td>
</tr>
<tr>
<td>IMDB</td>
<td>NP</td>
<td>FTC</td>
<td>97.0</td>
<td>88.6</td>
<td>88.2</td>
<td>91.1</td>
<td>37.8</td>
<td>13.4</td>
<td>17.1</td>
<td>11.2</td>
<td>13.4</td>
<td>11.2</td>
<td>8.2</td>
</tr>
<tr>
<td></td>
<td>ADCL</td>
<td>FTC</td>
<td>96.1</td>
<td>35.6</td>
<td>50.1</td>
<td>61.0</td>
<td>21.0</td>
<td>13.4</td>
<td>17.1</td>
<td>11.2</td>
<td>13.4</td>
<td>11.2</td>
<td>8.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ADV</td>
<td>92.3</td>
<td>98.7</td>
<td>99.0</td>
<td>99.2</td>
<td>54.0</td>
<td>3.5</td>
<td>6.5</td>
<td>4.3</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
</tr>
<tr>
<td>DBpedia</td>
<td>NP</td>
<td>FTC</td>
<td>92.5</td>
<td>93.3</td>
<td>96.6</td>
<td>95.1</td>
<td>52.0</td>
<td>7.4</td>
<td>12.7</td>
<td>9.3</td>
<td>3.6</td>
<td>3.6</td>
<td>3.6</td>
</tr>
<tr>
<td></td>
<td>ADCL</td>
<td>FTC</td>
<td>92.5</td>
<td>93.3</td>
<td>96.6</td>
<td>95.1</td>
<td>52.0</td>
<td>4.5</td>
<td>7.4</td>
<td>4.4</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ADV</td>
<td>91.9</td>
<td>84.2</td>
<td>87.8</td>
<td>87.8</td>
<td>48.0</td>
<td>3.7</td>
<td>8.7</td>
<td>5.1</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
</tr>
</tbody>
</table>

**H1:** To validate H1, we focus on rows with the FTC setting during finetuning. We can observe that models without any contrastive pretraining (NP) are the most vulnerable to adversarial attacks. For example, the success rate of the Geometry Attack for AG’s News dataset is 86.2% for the NP model. In contrast, for BTCL and ADCL, the success rate of the Geometry Attack is at least 5.6% lower than this setting. This shows that _self-supervised contrastive learning does improve model robustness._

Additionally, we can also see from table 1 that ADCL improves the model robustness more than BTCL. For example, in the IMDB dataset, the model pretrained with ADCL is 9.1% more robust than the model pretrained with BTCL (93.3% → 84.2%), showing that _using adversarial perturbations during contrastive learning further improves model robustness against adversarial attacks_. Hence, we do not combine BTCL with ADV in later experiments for simplicity.

To further understand how contrastive learning improves the model robustness, we study the transferability of the adversarial examples between models without any contrastive pretraining (NP) and the models pretrained with ADCL. To be specific, the models are first pretrained using either NP or ADCL, and then finetuned on clean examples (FTC). Then, we use a NP model to generate adversarial examples on the test set of each dataset, and then test the corresponding model pretrained with ADCL on these adversarial examples. And vice versa.

Table 3 shows the results. We can see that adversarial examples generated by models pretrained with ADCL have much higher success rates on models without any contrastive pretraining (NP). For example, for the AG’s News dataset, the success rates increase by 32.1%, 35.3%, 33.8%, and 22.1% for Geometry Attack, TextFooler, BAE-R, and PWWS, respectively. This demonstrates that by self-supervised contrastive learning with adversarial perturbations, the models become more robust against attacks.

**H2:** To validate H2, we compare two settings of NP + ADV and ADCL + ADV. We note that when compared with conducting adversarial training alone (NP + ADV), _combining our self-supervised contrastive learning method with adversarial training (ADCL + ADV) constantly results in higher robustness_. In other words, the adversarial attacks have lower success rates and higher replacement rate in ADCL + ADV models than in NP + ADV models. For instance, for the IMDB dataset, the ADCL + ADV model is 2.7% more robust than the NP + ADV model, when both models are tested against the Geometry Attack (Success rates of Geometry attack: ADCL + ADV: 48.7%, NP + ADV: 51.4%; Replacement rates: ADCL + ADV: 8.1%, NP + ADV: 7.4%).

Note that when test NP + ADV models and ADCL + ADV models against the other three ad-
Table 2: Comparison of Out-of-Domain with In-Domain. We use the DBpedia dataset as the out-of-domain dataset for AG’s News and IMDB. Models are finetuned on clean examples after pretraining (FTC). Best results are bolded, while the second best are in italic.

Table 3: Transferability of adversarial examples. The models are pretrained under either NP or ADCL, and then finetuned on clean examples. NP → ADCL: We generate adversarial examples using the model pretrained with NP, then test the model pretrained with ADCL on these adversarial examples. The same applies to ADCL → NP.

versarial attacks, ADCL + ADV models do not show an advantage over NP + ADV models in terms of replacement rates, despite that ADCL + ADV models still constantly make lower success rates against the adversarial attacks. We argue that this is because we use the Geometry Attack for adversarial training during finetuning, and the adversarial examples from the Geometry Attack might not fully match the distribution from the other attacks. Nevertheless, we can still conclude that ADCL + ADV models are more robust than NP + ADV models.

Our experiments also show that during contrastive learning, the queue size (see section 3.2) has an impact on the final performance. We give the detailed analysis in appendix C.

**H3:** For the Out-of-Domain setting, we use the DBpedia dataset as the out-of-domain dataset for the AG’s News and IMDB datasets, mainly because (1) Computational limits: While using larger datasets such as BookCorpus or Wikipedia might be more useful, conducting self-supervised contrastive learning on these datasets exceeds the limits of our computational infrastructure; (2) The DBpedia dataset is several times larger than AG’s News and IMDB. This should give us a glimpse of what it looks like when we scale self-supervised contrastive learning with adversarial perturbations to even larger out-of-domain datasets; (3) The DBpedia dataset (topic classification on Wikipedia) has a different task and domain compared to the AG’s News dataset (news classification from a newspaper) and IMDB dataset (sentiment classification on movie reviews). This discrepancy allows us to understand how out-of-domain datasets could help.

Table 2 shows our results. We can see that models pretrained with ADCL under the Out-of-Domain setting are more robust than models without any pretraining at all (NP). This shows that our method can improve model robustness using out-of-domain data. For instance, for the IMDB dataset, the success rate of TextFooler decreases from 98.7% for FT models to 92.3% for Out-of-Domain ADCL models. This shows that our method can improve the model robustness even if the dataset used for contrastive learning is from a completely different domain. Note that in table 2, after pretraining, we fine-tune the model on clean examples (FTC).

We also notice that models pretrained with ADCL under the Out-of-Domain setting are not as robust as models pretrained with ADCL under the In-Domain setting. This indicates we might need to use much larger unlabeled raw datasets to obtain more improvements.

**H4:** To validate this hypothesis, we study the vector representations of $M = 1000$ clean examples of the AG’s News dataset and their corresponding
adversarial examples. We obtain the adversarial examples by attacking a NP + FTC model.

Let \( v_1, v_2, \ldots, v_M \) and \( v'_1, v'_2, \ldots, v'_M \) be the vector representations of the clean examples and corresponding adversarial samples, respectively. For each setting, we evaluate three metrics:

- The average distance \( d_{pos} \) between each of the positive pairs \( v_i \) and \( v'_i \), where \( 1 \leq i \leq M \). Then we have \( d_{pos} = \frac{1}{M} \sum_{i=1}^{M} \text{distance}(v_i, v'_i) \).
- The average distance \( d_{neg} \) between negative pairs, which is \( d_{neg} = \frac{1}{2(M-1)} \sum_{i=1}^{M} \sum_{j=i+1}^{M} \mathbb{1}_{i \neq j} (\text{distance}(v_i, v_j) + \text{distance}(v'_i, v'_j)) \).
- The difference \( \delta = d_{neg} - d_{pos} \) between (a) and (b).

Furthermore, we evaluate the above metrics under the following settings:

- NP + FTC: Finetune the model on clean examples.
- ADCL + FTC: First do ADCL pretraining, and then finetune the model on clean examples.
- NP + ADV: Finetune the model with adversarial training.
- ADCL + ADV: First do ADCL pretraining. Then finetune with adversarial training.

Table 4 shows the results. We can see that our method (1) increases the distance between negative pairs in all settings; (2) decreases the distance between positive pairs in NP + FTC and ADCL + FTC models, while the distances between positive pairs barely change in NP + ADV and ADCL + ADV models; (3) increases \( \delta \) in all settings. The above observations validate H4 in section 4.4, and further explain that our method achieves higher robustness by pushing vector representations of clean examples and adversarial examples closer.

In fig. 3, we further give qualitative analysis on the distances between clean examples and adversarial examples of the AG’s News dataset by showing the t-SNE plot. We can see from the plot that the distances between the clean examples and the corresponding adversarial examples are closer when we apply ADCL pretraining, and that combining advcl with adv gives the smallest distance between supervised adversarial examples. Additional plots of other datasets are available in appendix H.

5 Conclusion and Future Work

In this paper, we improve the robustness of pre-trained language models against word substitution-based adversarial attacks by using self-supervised contrastive learning with adversarial perturbations. Our method is different from previous works as we can improve model robustness without accessing annotated labels. Furthermore, we also conduct word-level adversarial training on BERT with an efficient attack. Our adversarial training is different from previous works in that (1) it is on the word level; (2) we generate adversarial examples on the fly, instead of generating a fixed adversarial set beforehand. Experiments show that our method improves model robustness. We find that combining our method with adversarial training results in better robustness than conducting adversarial training alone. In the future, we plan to scale our method to even larger out-of-domain datasets.
**Ethical Considerations**

To the best of our knowledge, the data used in our work does not contain sensitive information. Although our models are evaluated on academic datasets in this paper, they could also be used in sensitive contexts, e.g. healthcare or legal scenarios. It is essential that necessary anonymization and robustness evaluation is undertaken before using our models in these settings.

**References**


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 *ACL*.


Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *ACL*.

Dong Wang, Ning Ding, Piji Li, and Haitao Zheng. 2021a. CLINE: Contrastive learning with semantic negative examples for natural language understanding. In *ACL*.


A Geometry Attack for Contrastive Loss

Algorithm 1 is the pseudocode of our Geometry Attack for contrastive loss. Refer to Section 3.3 for more details.

B Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Labels</th>
<th>Avg Len</th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG’s News</td>
<td>4</td>
<td>44</td>
<td>120K</td>
<td>7.6K</td>
</tr>
<tr>
<td>IMDB</td>
<td>2</td>
<td>292</td>
<td>25K</td>
<td>25K</td>
</tr>
<tr>
<td>DBPedia</td>
<td>14</td>
<td>67</td>
<td>560K</td>
<td>70K</td>
</tr>
<tr>
<td>Yelp</td>
<td>2</td>
<td>177</td>
<td>560K</td>
<td>38K</td>
</tr>
</tbody>
</table>

Table 5: Statistics of the datasets.

The statistics of each dataset are shown in Table 5. In our work, the maximum sequence length is set to 128 for AG’s News and DBpedia, 256 for Yelp, and 512 for IMDB. To save time during evaluating the model robustness against attacks, we randomly select a part of the test examples in each dataset for evaluation. Specifically, we select 1,000 samples from IMDB, 2,000 samples from Yelp, and 5,000 samples from DBpedia. We use all 7,600 samples from the AG’s News test set for evaluation.


IMDB (Maas et al., 2011): Binary sentiment classification dataset on positive and negative movie reviews.

Yelp: Yelp review dataset for binary sentiment classification. Following Zhang et al. (2015), reviews with star 1 and 2 are considered negative, and reviews with star 3 and 4 are considered positive.

DBpedia (Zhang et al., 2015): Topic classification dataset with 14 non-overlapping classes. Both content and title fields are used in our work.

C Effect of Queue Size

We conduct additional experiments to study the effect of queue size. We use a queue size of 8192, 16384, 32768, and 65536 under the setting of ADCL+FT for the AG’s News dataset. As is shown in Table 6, a larger queue size generally helps improve the model robustness. However, we also notice that when the queue size is too large (65536), the model robustness starts to decrease. We argue that this is because a too large queue size results in less frequent queue updates, which makes the vectors in the queue stale.

Table 6: Effect of queue size. We use the Geometry Attack to evaluate the robustness of each model. The FT model is finetuned without contrastive learning.

<table>
<thead>
<tr>
<th>Queue Size</th>
<th>Original Acc. (%)</th>
<th>Success (%)</th>
<th>Replaced (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>94.2</td>
<td>86.2</td>
<td>18.6</td>
</tr>
<tr>
<td>8192</td>
<td>94.4</td>
<td>77.8</td>
<td>18.9</td>
</tr>
<tr>
<td>16384</td>
<td>94.3</td>
<td>76.9</td>
<td>18.7</td>
</tr>
<tr>
<td>32768</td>
<td>94.3</td>
<td>76.5</td>
<td>19.1</td>
</tr>
<tr>
<td>65536</td>
<td>94.4</td>
<td>76.7</td>
<td>19.3</td>
</tr>
</tbody>
</table>

D Speed of Different Attacks

We show in Table 7 the average number of seconds each attack needs for one example. We obtain the average time by attacking 1000 examples and then taking the average. We can observe that the Geometry attack is at least four times faster than TextFooler, and 4 to 10 times faster than PWWS and BAE-R. The efficiency of the Geometry attack allows us to generate adversarial examples on the fly.

Table 7: Average number of seconds each attack needs for an example.

<table>
<thead>
<tr>
<th>Attack</th>
<th>AG’s News</th>
<th>IMDB</th>
<th>DBpedia</th>
<th>Yelp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry</td>
<td>0.44</td>
<td>2.02</td>
<td>0.69</td>
<td>1.16</td>
</tr>
<tr>
<td>TextFooler</td>
<td>2.48</td>
<td>8.69</td>
<td>2.89</td>
<td>4.86</td>
</tr>
<tr>
<td>PWWS</td>
<td>6.29</td>
<td>21.86</td>
<td>2.52</td>
<td>10.27</td>
</tr>
<tr>
<td>BAE-R</td>
<td>5.37</td>
<td>24.10</td>
<td>7.74</td>
<td>16.03</td>
</tr>
</tbody>
</table>

E Adversarial Training with Pre-generated Examples

We compare two different methods for adversarial training:

- **Pre-generated** We pre-generate for each example in the training set an adversarial example. We then augment the original training set with the adversarial examples. Finally, the model is finetuned on the augmented dataset.

- **On-the-fly** This setting is the same as ADV in Table 1, where we generate adversarial examples on the fly for each mini-batch during training.
Algorithm 1 Geometry Attack for Contrastive Loss

1: **Input**: Example \( X_i = \{w_1, w_2, \ldots, w_L\} \), encoder \( f \) and MLP \( g \)
2: **Output**: Adversarial example \( X'_i \)
3: Initialize \( z_i \leftarrow g(f(X_i)) \)
4: for \( \text{iter} = 1 \) to \( N \) do
5: \( \nu_{z_i} \leftarrow \nabla_{z_i} \ell_i \)
6: \( \mathbf{E} \leftarrow \text{BertEmbeddings}(X'_i) = \{e_1, e_2, \ldots, e_L\} \)
7: \( \mathbf{G} \leftarrow \nabla_{\ell_i} = \{g_1, g_2, \ldots, g_L\} \)
8: \( t \leftarrow \arg \max_i \|g_i\| \)
9: \( \mathbf{C} \leftarrow \text{BertForMaskedLM}(\{w_1, \ldots, w_{t-1}, [\text{MASK}], w_{t+1}, \ldots, w_L\}) \)
10: \( \mathbf{C} \leftarrow \text{Filter}(\mathbf{C}) /\!\!\!/ \text{construct candidates set } \mathbf{C} = \{w_{t_1}, w_{t_2}, \ldots, w_{t_T}\} \)
11: for each \( w_{t_j} \in \mathbf{C}, 1 \leq j \leq T \) do
12: \( X_{ij} \leftarrow \{w_1, \ldots, w_{t-1}, w_{t_j}, w_{t+1}, \ldots, w_L\} \)
13: \( r_{ij} \leftarrow \frac{z_i - g(f(X_{ij}))}{\|r_i\|} \)
14: \( p_{ij} \leftarrow \frac{z_i - g(f(X_{ij}))}{\|r_i\|} \)
15: end for
16: \( m \leftarrow \arg \max_j \|p_{ij}\| \)
17: \( X_i \leftarrow X_{im} \)
18: \( z_i \leftarrow z_{im} \)
19: end for
20: \( X'_i \leftarrow X_i \)
21: return \( X_i \)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Success Rate / Replaced (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-generated</td>
<td>55.3/17.1 ( \quad ) 59.4/22.6 ( \quad ) 42.0/17.4 ( \quad ) 16.5/7.3</td>
</tr>
<tr>
<td>On-the-fly</td>
<td>20.7/20.5 ( \quad ) 25.1/29.3 ( \quad ) 26.1/22.3 ( \quad ) 10.7/7.7</td>
</tr>
</tbody>
</table>

Table 8: Comparison between adversarial training with pre-generated adversarial examples and on-the-fly generated adversarial examples.

Table 8 shows the results on the AG’s News dataset. We can see that on all four attacks, adversarial training with on-the-fly generated adversarial examples gives higher robustness than adversarial training with pre-generated adversarial examples.

F Implementation Details

In our paper, we use PyTorch Lightning\(^5\) and HuggingFace Transformers\(^4\) in our implementation. We use BERT as the encoder \( f() \), and the representation of the [CLS] symbol in the last layer is used for \( h \). \( g() \) is a two-layer MLP, of which the output size \( c \) is 128. \( g() \) uses Tanh as activation function in the output layer. We use FP16 in training step to reduce GPU memory usage, and use FusedAdam from DeepSpeed\(^5\) as the optimizer. We enable DeepSpeed ZeRO Stage 2 to further speed up training. We conduct all our experiments on 8 RTX TITAN GPUs.

**Contrastive learning** For Geometry Attack for contrastive loss, to reach a balance between attack success rate and efficiency, the maximum number of iterations \( K \) is set to 10 for AG’s News, DBpedia, and Yelp, and 15 for IMDB dataset. We do not perturb words that were already perturbed in previous iterations. For an example \( X_i = \{w_1, w_2, \ldots, w_L\} \), at most \( \min\{K, 0.2 \cdot L\} \) words can be perturbed. For each word \( w_t, 1 \leq t \leq L \), the upper limit of the candidate set size \( T \) is set to 25. Due to the various maximum lengths in downstream datasets and GPU memory limits, we use different batch sizes for different datasets. During contrastive learning, the batch size is set to 1024 for AG’s News and DBpedia, 448 for Yelp, and 192 for IMDB.

**Fine-tuning** During finetuning, we train the model for two epochs for AG’s News and DBpedia, 3 for

\(^5\)https://www.deepspeed.ai/
Yelp, and 4 for IMDB. The learning rate is set to $2e^{-5}$ and is adjusted using linear scheduling.

**Adversarial training** For adversarial training, the number of training epochs is set to 3 with an additional first epoch of finetuning on clean examples. The adversarial examples are generated on the fly in each batch during training. For the Geometry Attack in adversarial training, at most $\min\{K, 0.4 \cdot \text{len}(X_i)\}$ words can be perturbed in an example. The upper limit of the candidate set size is set to 50.

**G Hard Positive Examples from Geometry Attack for Contrastive Loss**

In Table 9, we show hard positive examples generated by our Geometry Attack for contrastive loss from the AG’s News dataset.

<table>
<thead>
<tr>
<th>Original</th>
<th>Zurich employees plead guilty in probe new york (reuters) - two senior insurance underwriters at zurich american insurance co pleaded guilty on tuesday to misdemeanors related to bid-rigging in the insurance market.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adversarial</td>
<td>Zurich employees plead guilty in probe new york (reuters) - two senior insurance agents at zurich american insurance co testified guilty on tuesday to violations related to bid-rigging in the insurance market.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Original</th>
<th>It has a stunning lack of even rudimentary traces of realism. almost every war movie cliche appears in this film and is done badly. on the other hand , i wouldn’t have watched it to the end if it hadn’ t been so remarkably bad that it amused me.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adversarial</td>
<td>It has a stunning lack of even joyless traces of realism - almost every war movie cliche appears in this film and is done erroneously. on the other hand , i wouldn’t have watched it to the end if it hadn’ t been so remarkably bad that it flabbergasted me.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Original</th>
<th>Black watch troops move into position the first units of a black watch battlegroup are due to arrive today in their new positions south of baghdad as tony blair indicated that more british troops may replace them in the american - controlled zone before the end of the year.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adversarial</td>
<td>Black watch troops move into place the first units of a black watch operation are due to arrive today in their new positions south of baghdad as tony blair indicated that more british troops may replace them in the american - controlled zone before the end of the year.</td>
</tr>
</tbody>
</table>

Table 9: Hard positive examples generated by Geometry Attack for contrastive loss. Blue words in the original examples are replaced by red words in the adversarial examples.

**H Additional t-SNE plots**

We give t-SNE plots of the vector representations of clean examples and adversarial examples from Yelp, IMDB and DBpedia in fig. 4, fig. 5 and fig. 6, respectively.

Figure 4: t-SNE plot of the vector representations of clean examples and adversarial examples from the Yelp dataset.
Figure 5: t-SNE plot of the vector representations of clean examples and adversarial examples from the IMDB dataset.

Figure 6: t-SNE plot of the vector representations of clean examples and adversarial examples from the DBpedia dataset.