Weight Perturbation as Defense against Adversarial Word Substitutions

Jianhan Xu[1,2], **Linyang Li**[1,2], **Jiping Zhang**[1,2],**Xiaoqing Zheng**[1,2], **Kai-Wei Chang**[3], **Cho-Jui Hsieh**[3], **Xuanjing Huang**[1,2]

[1]School of Computer Science, Fudan University, Shanghai, China
[2]Shanghai Key Laboratory of Intelligent Information Processing
[3]Department of Computer Science, University of California, Los Angeles, USA
{jianhanxu20,zhengxq}@fudan.edu.cn
{kwchang,chohsieh}@cs.ucla.edu

Abstract

The existence and pervasiveness of textual adversarial examples have raised serious concerns to security-critical applications. Many methods have been developed to defend against adversarial attacks for neural natural language processing (NLP) models. Adversarial training is one of the most successful defense methods by adding some random or intentional perturbations to the original input texts and making the models robust to the perturbed examples. In this study, we explore the feasibility of improving the adversarial robustness of NLP models by performing perturbations in the parameter space rather than the input feature space. The weight perturbation helps to find a better solution (i.e., the values of weights) that minimizes the adversarial loss among other feasible solutions. We found that the weight perturbation can significantly improve the robustness of NLP models when it is combined with the perturbation in the input embedding space, yielding the highest accuracy on both clean and adversarial examples across different datasets.

1 Introduction

Deep neural networks (DNNs) have achieved impressive results in a wide range of domains, but they were found to be vulnerable to adversarial examples maliciously crafted by adding a small perturbation to original examples (Szegedy et al., 2014). Many studies have demonstrated the vulnerability of DNNs on various natural language processing (NLP) tasks, including machine translation (Zhao et al., 2018; Cheng et al., 2020), dialogue systems (Cheng et al., 2019) and text classification (Liang et al., 2018; Zhao et al., 2018; Gao et al., 2018; Ren et al., 2019; Jin et al., 2020). These methods attack an NLP model by replacing, scrambling, and erasing characters or words under certain semantic and syntactic constraints.

The existence and pervasiveness of textual adversarial examples have raised serious concerns, especially when NLP models are deployed to security-sensitive applications. Many methods have been proposed to defend against adversarial attacks for neural NLP models, including adversarial data augmentation (Zheng et al., 2020; Si et al., 2021), adversarial training (Madry et al., 2018; Zhu et al., 2020) and certified defense (Jia et al., 2019; Huang et al., 2019; Ye et al., 2020). Most of them improve the adversarial robustness of NLP models by applying some perturbations on input data and making the models robust to these perturbations in the input space. For example, one of the most effective methods is adversarial training that applies a min-max optimization into the training process by adding (usually gradient-guided) perturbations to the input embeddings (Miyato et al., 2017; Sato et al., 2018; Zhu et al., 2020). By augmenting these perturbed examples with the original training data, the models are robust to such perturbations. However, it is infeasible to enumerate and explore all possible inputs that would be fed to models by adversaries. In this study, we want to explore the feasibility of enhancing the robustness of neural NLP models by performing weight perturbations in the parameter space. The weight perturbation is useful to find a better solution in the parameter space (i.e., the weights) that minimizes the adversarial loss among other feasible solutions.

Adversarial weight perturbation (Wu et al., 2020; Foret et al., 2021) has been investigated in the image domain, but our preliminary experiments show that their methods to implement weight perturbation cannot be trivially applied to NLP models. The existing weight perturbation results in inferior robustness and requires a long training time due to the discrete nature of texts. We found that the weight perturbation works better for NLP models when it is combined with the perturbation in the input feature space. Based on this finding, we propose a *mixed adversarial training method* with *accumulated weight perturbation*, named MAWP. The mixed adversarial training was designed to boost the model's robustness by combining the weight perturbation with the traditional adversarial training (i.e., perturbation in the input embedding space like FreeLB (Zhu et al., 2020)). In this way, the resulting models can benefit more from the weight perturbation by exposing them to the input perturbation during the training process. The accumulated weight perturbation was mainly introduced to accelerate the training process while the model's robustness can be further improved. The accumulated perturbation takes the smoothed form of a weighted sum of gradient descents calculated in the previously-performed weight perturbations, which carries the global gradient information and gives a clear signal in which direction the parameters should move to aggressively if the successive gradients point in a similar direction. Through extensive experiments, we demonstrate that our method can boost the robustness of NLP models to a great extent while suffering no or little performance drop on the clean data across three different datasets.

2 Related Work

2.1 Textual Adversarial Defense

The goal of adversarial defenses is to learn a model capable of achieving high accuracy on both clean and adversarial examples. Recently, many defense methods have been developed to defend against textual adversarial attacks, which can roughly be divided into two categories: *empirical* (Miyato et al., 2017; Sato et al., 2018; Zhou et al., 2021; Dong et al., 2021) and *certified* (Jia et al., 2019; Huang et al., 2019; Ye et al., 2020) methods.

Adversarial data augmentation is one of the most effective empirical defenses (Ren et al., 2019; Jin et al., 2020; Li et al., 2020) for NLP models. During the training time, they replace a word with one of its synonyms to create adversarial examples, and the models are trained on the dataset augmented with these adversarial examples. By augmenting these adversarial examples with the original training data, the model is robust to such perturbations. Zhou et al. (2021) and Dong et al. (2021) relax a set of discrete points (a word and its synonyms) to a convex hull spanned by the word embeddings of all these points, and use a convex hull formed by a word and its synonyms to capture word substitutions. Adversarial training (Miyato et al., 2017; Zhu et al., 2020) is another one of the most successful empirical defense methods by adding norm-bounded adversarial perturbations to word embeddings and minimizes the resultant adversarial loss.

The downside of existing empirical methods is that failure to discover an adversarial example does not mean that another more sophisticated attack could not find one. To address this problem, some certified defenses (Jia et al., 2019; Huang et al., 2019; Ye et al., 2020) have been introduced to guarantee the robustness to certain specific types of attacks. However, the existing certified defense methods make an unrealistic assumption that the defenders can access the synonyms used by the adversaries. They would be easily broken by more sophisticated attacks by using synonym sets with large sizes (Jin et al., 2020) or generating synonyms dynamically with BERT (Li et al., 2020).

Most of the existing defense methods improve the robustness by making the models adapt to the training set augmented with the adversarial examples crafted by adding adversarial perturbations to discrete tokens or distributed embeddings. However, it is infeasible to enumerate all possible inputs that would be fed to the models by adversaries. In contrast, we have full control over the values of the model's parameters. Therefore, we propose to improve the robustness of neural NLP models by performing weight perturbations in the parameter space rather than in the input space.

2.2 Weight Perturbation

Weight perturbation has been explored to improve the generalization of models in the image domain. Graves (2011) first investigated the method to apply the perturbations on the weights of neural networks and introduced a stochastic variational method to improve the generalization. Following this direction, Foret et al. (2021) proposed an optimization method, named Sharpness-Aware Minimization (SAM), to seek the values of parameters that yield a uniformly low loss in their neighborhood.

Recently, researchers from the computer vision community have also tried to improve the model's robustness by weight perturbation. He et al. (2019) presented a Parametric Noise Injection method that intentionally injects trainable noises on the activations and weights of neural networks. Wu et al. (2020) showed the correlation of model's performance with the direction and scale of weight perturbation by investigating the weight loss landscapes of multiple adversarial training techniques. They also proposed an Adversarial Weight Perturbation

(AWP) method, which can be incorporated into existing adversarial training methods to narrow down the gap in robustness between training and test sets.

However, the existing weight perturbation methods cannot trivially be applied to NLP models due to the discrete nature of texts. To address the problem we faced when implementing the weight perturbation to train NLP models, we propose a mixed adversarial training method to further improve the adversarial robustness of neural NLP models and introduce an accumulated weight perturbation to speed up the training process. This study is among the first ones to study how to apply the adversarial weight perturbations in the text domain.

3 Preliminary

In the following, we first introduce the traditional adversarial training that performs perturbation in the input feature space, and then give a brief review of the adversarial weight perturbation. Before diving into the details, we need to set up some notations. A training dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ with n instances consists of a set of the feature vector representation $\boldsymbol{x} \in \mathbb{R}^d$ of an input text x and its corresponding label $y \in \{1, ..., C\}$ pairs, where d is the size of feature vectors and C is the number of classes. Given a neural text classifier with a set of trainable weights $\boldsymbol{w}$ and a loss function of $L(\boldsymbol{w}, \boldsymbol{x}, y)$, the regular training aims to find the values of weights $\boldsymbol{w}$ that minimizes the empirical risk of $\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}[L(\boldsymbol{w}, \boldsymbol{x}, y)]$. In adversarial training, we denote the adversarial perturbation to input feature vectors $\boldsymbol{x}$ as $\boldsymbol{\delta}$, the weight perturbation to model's weights $\boldsymbol{w}$ as $\boldsymbol{\epsilon_w}$, and the number of ascent steps as $k \in \{1, ..., K\}$.

3.1 Traditional Adversarial Training

Traditional adversarial training can be formulated as a min-max optimization problem (Madry et al., 2018) as follows:

$$\min_{\boldsymbol{w}} \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}} \left[\max_{\|\boldsymbol{\delta}\|_F \le \epsilon} L(\boldsymbol{w}, \boldsymbol{x} + \boldsymbol{\delta}, y) \right], \tag{1}$$

where $\boldsymbol{\delta}$ is constrained in a Frobenius norm ball with a radius ϵ. As pointed out by Zhu et al. (2020), the outer minimization can be achieved by Stochastic Gradient Descent (SGD) method, and the inner maximization can be accomplished by Projected Gradient Descent (PGD)-based attack algorithm. Specifically, PGD-based algorithms take the following step (with a step size α) at k-th iteration under the constraint of Frobenius norm:

$$\boldsymbol{\delta}_{k+1} = \prod_{\|\boldsymbol{\delta}\|_F \le \epsilon} \left(\boldsymbol{\delta}_k + \frac{\alpha g(\boldsymbol{\delta}_k)}{\|g(\boldsymbol{\delta}_k)\|_F} \right), \tag{2}$$

where $g(\boldsymbol{\delta}_k) = \nabla_{\boldsymbol{\delta}_k} L(\boldsymbol{w}, \boldsymbol{x} + \boldsymbol{\delta}_k, y)$ denotes the loss gradient with respect to $\boldsymbol{\delta}_k$, and $\prod_{\|\boldsymbol{\delta}\|_F \le \epsilon}$ is the projection of input perturbation $\boldsymbol{\delta}$ within the Frobenius norm ball with the radius ϵ.

3.2 Adversarial Weight Perturbation

We here give a brief introduction of adversarial weight perturbation (Wu et al., 2020; Foret et al., 2021) in the image domain. Given an adversarial example $\boldsymbol{x}'$ of a clean one $\boldsymbol{x}$, the adversarial weight perturbation seeks the values of parameters $\boldsymbol{w}$ that have the lowest training loss within the surrounding neighborhood, which can be formulated as follows:

$$\min_{\boldsymbol{w}} \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}} \left[\max_{\|\boldsymbol{\epsilon_w}\|_2 \le \rho} L(\boldsymbol{w} + \boldsymbol{\epsilon_w}, \boldsymbol{x}', y) \right], \tag{3}$$

where ρ is the radius of weight perturbation under the l_2-norm. Since it is hard to directly maximize the values of $\boldsymbol{\epsilon_w}$, the optimal values can be approximated via the first-order Taylor expansion of $L(\boldsymbol{w} + \boldsymbol{\epsilon_w}, \boldsymbol{x}', y)$ as follows:

$$\boldsymbol{\epsilon}^*_{\boldsymbol{w}} \approx \arg\max_{\|\boldsymbol{\epsilon_w}\|_2 \le \rho} \boldsymbol{\epsilon}_{\boldsymbol{w}}^T \nabla_{\boldsymbol{w}} L(\boldsymbol{w}, \boldsymbol{x}', y). \tag{4}$$

By this approximation, the value of $\boldsymbol{\epsilon_w}$ can be estimated as $\rho \nabla_{\boldsymbol{w}} L(\boldsymbol{w}, \boldsymbol{x}', y)/\|\nabla_{\boldsymbol{w}} L(\boldsymbol{w}, \boldsymbol{x}', y)\|_2$. Specifically, Wu et al. (2020) performs the adversarial weight perturbations at layer level by setting $\boldsymbol{\epsilon_w}$ to $\eta\|\boldsymbol{w}\|_2 \cdot \nabla_{\boldsymbol{w}} L(\boldsymbol{w}, \boldsymbol{x}', y)/\|\nabla_{\boldsymbol{w}} L(\boldsymbol{w}, \boldsymbol{x}', y)\|_2$. Once the values of $\boldsymbol{\epsilon_w}$ are obtained, they update $\boldsymbol{w}$ based on the gradient of $\nabla_{\boldsymbol{w}} L(\boldsymbol{w}, \boldsymbol{x}', y)|_{\boldsymbol{w}+\boldsymbol{\epsilon_w}}$ in order to make the models generalize well on the adversarial sample $\boldsymbol{x}'$.

4 Method

In the following, we first introduce our accumulated weight perturbation method that is designed to accelerate the training process when fine-tuning a pre-trained language model. After then, we discuss how to combine it with adversarial training to further improve the robustness of NLP models.

4.1 Accumulated Weight Perturbation

Adversarial weight perturbation works by searching for the worst case within the neighborhood of current weights with respect to the training loss and finding a better solution in the neighborhood

Algorithm 1 A mixed adversarial training algorithm with accumulated weight perturbation

```
Input:
    $K$: the number of ascent steps;
    $\mathcal{D}$: a training dataset $(\boldsymbol{x}_i, y_i)_{i=1}^n$;
    $\alpha$: the size of ascent steps;
    $H$: the demensionality of the hidden layers;
    $\boldsymbol{w}$: the weights of a model;
    $\tau$: a learning rate;
    $\eta$: the size of weight perturbation $\boldsymbol{\epsilon_w}$;
    $\eta_2$: the size of accumulated weight perturbation $\boldsymbol{\epsilon_g}$;
    $U$: a uniform distribution with the bound of $\sigma$.
Output: the resulting weights $\boldsymbol{w}$.
 1: Initialize $\boldsymbol{w}$
 2: for epoch $= 1, \ldots, N$ do
 3:     $\boldsymbol{\epsilon_g} \leftarrow \mathbf{0}$
 4:     for minibatch $B \subset \mathcal{D}$ do
 5:         $\boldsymbol{w}_0 \leftarrow \boldsymbol{w}$, $g_0 \leftarrow 0$
 6:         $\boldsymbol{\delta}_0 \leftarrow \frac{1}{\sqrt{H}} U(-\sigma, \sigma)$
 7:         for $k = 0 \ldots (K-1)$ do
 8:             // Calculate weight perturbation $\boldsymbol{\epsilon}_{\boldsymbol{w}_k}$
 9:             $\boldsymbol{d}_{\boldsymbol{w}_k} \leftarrow \nabla_{\boldsymbol{w}_k} L(\boldsymbol{w}_k, \boldsymbol{x} + \boldsymbol{\delta}_k, y)$
10:             $\boldsymbol{\epsilon}_{\boldsymbol{w}_k} \leftarrow \eta \|\boldsymbol{w}_k\|_2 \cdot \frac{\boldsymbol{d}_{\boldsymbol{w}_k}}{\|\boldsymbol{d}_{\boldsymbol{w}_k}\|_2}$
11:             $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k + \boldsymbol{\epsilon}_{\boldsymbol{w}_k}$
12:             // Accumulate gradient $g_k$ for $\boldsymbol{w}_{k+1}$
13:             $g_{k+1} \leftarrow g_k + \frac{1}{K} \mathbb{E}_{(\boldsymbol{x},y) \in B} \nabla_{\boldsymbol{w}_{k+1}} L(\boldsymbol{w}_{k+1}, \boldsymbol{x} + \boldsymbol{\delta}_k, y)$
14:             // Update $\boldsymbol{\delta}_k$ via input perturbation
15:             $g_{adv} \leftarrow \nabla_{\boldsymbol{\delta}_k} L(\boldsymbol{w}_{k+1}, \boldsymbol{x} + \boldsymbol{\delta}_k, y)$
16:             $\boldsymbol{\delta}_{k+1} \leftarrow \prod_{\|\boldsymbol{\delta}\|_2 \le \epsilon} (\boldsymbol{\delta}_k + \alpha \frac{g_{adv}}{\|g_{adv}\|_2})$
17:         end for
18:         // Calculate accumulated perturbation $\boldsymbol{\epsilon_g}$
19:         $\boldsymbol{\epsilon_s} \leftarrow \sum_{k=0}^{K-1} \boldsymbol{\epsilon}_{\boldsymbol{w}_k}$
20:         $\boldsymbol{\epsilon_g} \leftarrow \prod_{\|\boldsymbol{\epsilon_g}\|_2 \le \eta_2} (\boldsymbol{\epsilon_s} + \frac{\boldsymbol{\epsilon_g} \|\boldsymbol{\epsilon_s}\|_2}{\|\boldsymbol{\epsilon_g}\|_2 + \epsilon_0})$
21:         $\boldsymbol{w} \leftarrow \boldsymbol{w} - \tau g_K - \boldsymbol{\epsilon_g}$
22:     end for
23: end for
24: return $\boldsymbol{w}$
```

through minimizing the adversarial loss. Note that the gradient calculated to find the worse case by the weight perturbation can be reused to optimize the value of the current weight, which would be updated in the reverse direction of the gradient already computed. For example, starting from a point of weight we can calculate its gradient with respect to the loss function to locate the worse case around this point to perform the weight perturbation. The normalized version of this gradient also can be used to update the model's parameters by the gradient descent. By reusing such a gradient calculated at each step of weight perturbation, we can improve both the generalization and robustness with less computational cost.

We also found experimentally that the robustness of models can be further improved by introducing a global term that takes the form of the accumulated gradients obtained at previous perturbation steps. The accumulated gradient carries the global information and gives a clear signal in which direction the parameters should move to aggressively if the gradients obtained at different steps point in a similar direction.

At each step k, we can perform the weight perturbation $\boldsymbol{\epsilon}_{\boldsymbol{w}_k}$ that finds the worse case in the neighborhoods of weight $\boldsymbol{w}_k$. Meanwhile, an accumulated weight perturbation is also calculated which takes the smoothed form of a weighted sum of gradient descents calculated in the previously-performed weight perturbations. Specifically, for each minibatch B, the values of weight perturbation $\boldsymbol{\epsilon_s}$ are obtained by summing up all the perturbations as $\boldsymbol{\epsilon_s} = \sum_{k=0}^{K-1} \boldsymbol{\epsilon}_{\boldsymbol{w}_k}$. Then, we can calculate the accumulated weight perturbation $\boldsymbol{\epsilon_g}$ as follows:

$$\boldsymbol{\epsilon_g} = \prod_{\|\boldsymbol{\epsilon_g}\|_2 \le \eta_2} \left(\boldsymbol{\epsilon_s} + \frac{\boldsymbol{\epsilon_g} \cdot \|\boldsymbol{\epsilon_s}\|_2}{\|\boldsymbol{\epsilon_g}\|_2 + \epsilon_0} \right), \tag{5}$$

where η_2 is the radius of the accumulated weight perturbation, ϵ_0 a constant introduced for numerical stability in the computation, and $\prod_{\|\boldsymbol{\epsilon_g}\|_2 \le \eta_2}$ the projection function.

4.2 Mixed Adversarial Training Method

We here describe our adversarial weight perturbation and how to combine it with FreeLB (Free Large-Batch) (Zhu et al., 2020), a popular adversarial training algorithm. The algorithm of FreeLB adds adversarial perturbations to word embeddings and minimizes the resultant adversarial loss inside different regions around input samples. It also adds norm-bounded adversarial perturbations to the input sentences' embeddings using a gradient-based method and enlarges the batch size with diversified adversarial samples under such norm constraints. By the mixed adversarial training, NLP models can benefit more from the adversarial weight perturbation by exposing the models to the input perturbation during the training process.

Specifically, at the k-th ascent step, we calculate the weight perturbation $\boldsymbol{\epsilon}_{\boldsymbol{w}_k}$ based on the input perturbations $\boldsymbol{\delta}_k$ and the weights $\boldsymbol{w}_k$ as follows:

$$\boldsymbol{\epsilon}_{\boldsymbol{w}_k} = \eta \|\boldsymbol{w}_k\|_2 \cdot \frac{\nabla_{\boldsymbol{w}_k} L(\boldsymbol{w}_k, \boldsymbol{x} + \boldsymbol{\delta}_k, y)}{\|\nabla_{\boldsymbol{w}_k} L(\boldsymbol{w}_k, \boldsymbol{x} + \boldsymbol{\delta}_k, y)\|_2}. \tag{6}$$

The input perturbation δ_0 is initialized from a uniform distribution, and $\delta_0 \sim \frac{1}{\sqrt{H}} U(-\sigma, \sigma)$. After calculating the weight perturbation $\boldsymbol{\epsilon}_{\boldsymbol{w}_k}$, the perturbed weights of $\boldsymbol{w}_{k+1}$ can be updated by $\boldsymbol{w}_k + \boldsymbol{\epsilon}_{\boldsymbol{w}_k}$. Following the gradient accumulating operation in FreeLB, we compute the gradients of

Datasets	Methods	*Clean%*	TextFooler		BERT-Attack		TextBugger		TextFooler*		BERT-Attack*		TextBugger*	
			Aua%	*#Query*	*Aua%*	*#Query*	*Aua%*	*#Query*	*Aua%*	*#Query*	*Aua%*	*#Query*	*Aua%*	*#Query*
SST-2	Base	92.24	11.77	101.22	11.10	128.23	28.00	52.44	19.93	102.81	18.97	107.94	21.63	99.99
	PGD	90.78	10.87	114.16	8.67	131.56	33.73	49.87	29.37	106.73	22.53	106.29	29.60	103.88
	FreeLB++	92.35	13.33	112.86	11.40	136.90	33.07	51.10	33.83	110.43	27.63	111.22	33.93	109.16
	TA-VAT	93.08	13.89	117.06	11.56	133.47	32.89	53.45	28.00	106.11	22.33	106.84	27.33	106.64
	InfoBERT	**93.14**	10.56	111.95	10.33	138.74	33.33	53.09	26.67	105.50	22.45	111.87	28.89	106.15
	PGD-AWP	91.14	11.87	113.53	9.93	129.50	32.33	50.27	29.17	106.68	22.50	106.33	27.90	104.66
	FreeLB-AWP	92.35	14.80	118.97	13.67	142.30	35.63	53.89	35.37	121.32	29.00	113.61	36.73	129.45
	MAWP	91.96	**31.30**	**146.01**	**24.57**	**184.02**	**43.07**	**88.02**	**41.47**	**174.01**	**32.60**	**167.06**	**39.23**	**187.00**
AGNEWS	Base	94.50	14.20	320.72	21.87	433.90	38.23	178.74	23.90	345.58	36.73	380.99	28.10	371.26
	PGD	94.56	30.43	413.93	25.80	456.81	57.03	169.74	50.03	416.89	42.60	389.03	56.67	416.02
	FreeLB++	**95.33**	28.20	410.36	29.83	**494.79**	55.17	185.76	47.60	421.38	**47.10**	**415.16**	53.63	437.54
	TA-VAT	94.97	28.44	404.81	28.00	470.20	52.00	185.56	43.11	405.67	44.22	403.80	49.44	422.24
	InfoBERT	95.04	17.11	351.46	23.33	448.84	46.00	**187.95**	30.22	367.01	37.67	389.81	34.78	391.28
	PGD-AWP	94.38	28.53	407.12	23.77	446.69	57.60	165.08	48.40	410.16	39.83	380.61	55.70	405.38
	FreeLB-AWP	94.39	**32.03**	**425.13**	29.03	481.79	**58.57**	178.73	**51.57**	**425.95**	46.07	406.96	**57.43**	435.43
	MAWP	95.23	31.37	423.78	**29.97**	481.29	58.40	178.99	50.70	424.20	46.23	407.66	57.33	**436.20**
IMDB	Base	92.09	8.53	866.73	6.10	878.57	18.27	576.04	26.90	672.57	26.60	526.46	26.80	691.31
	PGD	92.70	10.33	1059.83	7.33	862.28	17.41	590.15	40.73	788.64	24.93	562.40	34.60	769.36
	FreeLB++	**93.31**	15.80	1298.17	10.33	1149.16	26.08	757.12	48.20	937.28	37.53	641.63	44.67	1005.66
	TA-VAT	92.84	12.60	1270.00	9.77	1105.95	28.11	830.73	45.77	885.73	35.53	638.21	41.40	1008.66
	InfoBERT	92.60	10.67	882.74	8.44	906.67	16.33	596.63	28.67	713.80	25.67	567.49	26.89	774.45
	PGD-AWP	93.18	12.83	1258.87	8.77	1068.43	22.22	733.80	47.47	879.47	33.80	628.32	42.60	915.04
	FreeLB-AWP	93.25	18.80	1353.93	13.47	1184.38	28.58	778.54	49.23	897.46	37.53	636.53	44.93	948.74
	MAWP	93.24	**35.97**	**1594.49**	**15.90**	**1501.68**	**39.90**	**1033.27**	**58.40**	**2522.56**	**43.20**	**1795.51**	**56.07**	**3167.35**

Table 1: The experimental results of different defense methods on SST-2, AGNEWS, and IMDB datasets. The best performance is highlighted in bold fonts. The symbol * indicates the attack algorithms on which we impose some constraints for fair comparison by ensuring the quality of adversarial examples (see Section 5.2 for details).

g_{k+1} with respect to the perturbed weights $\boldsymbol{w}_{k+1}$, the perturbed inputs $\boldsymbol{x} + \boldsymbol{\delta}_k$ and the accumulated gradient g_k in $k-1$-th step:

$$g_{k+1} = g_k + \frac{1}{K} \nabla_{\boldsymbol{w}_{k+1}} L(\boldsymbol{w}_{k+1}, \boldsymbol{x} + \boldsymbol{\delta}_k, y) \qquad (7)$$

where g_0 is initialized to 0. When we calculate the accumulated gradient g_{k+1}, it is free for us to calculate the input perturbation without additional cost. Therefore, we can calculate the input perturbation $\boldsymbol{\delta}_{k+1}$ based on the perturbed weights $\boldsymbol{w}_{k+1}$ and the former input perturbation $\boldsymbol{\delta}_k$ as follows:

$$\boldsymbol{\delta}_{k+1} = \prod_{\|\boldsymbol{\delta}\|_2 \le \epsilon} \left(\boldsymbol{\delta}_k + \alpha \frac{\nabla_{\boldsymbol{\delta}_k} L(\boldsymbol{w}_{k+1}, \boldsymbol{x} + \boldsymbol{\delta}_k, y)}{\|\nabla_{\boldsymbol{\delta}_k} L(\boldsymbol{w}_{k+1}, \boldsymbol{x} + \boldsymbol{\delta}_k, y)\|_2} \right). \qquad (8)$$

We list the proposed MAWP in Algorithm 1. After the weights $\boldsymbol{w}_0$ and the input perturbations $\boldsymbol{\delta}_0$ are initialized, we calculate the weight perturbations $\boldsymbol{\epsilon}_{\boldsymbol{w}_k}$ and add them to the model's weight $\boldsymbol{w}_k$ at each ascent step. We then calculate the gradient g_k (Line 14) and the next input perturbations $\boldsymbol{\delta}_{k+1}$ (Line 16). Finally, the accumulated weight perturbations $\boldsymbol{\epsilon}_{\boldsymbol{g}}$ are calculated to update the model's weights with the adversarial loss.

5 Experiments

We conducted three sets of experiments. The first one is to evaluate our MAWP in both clean accuracy and adversarial robustness on several datasets under three representative attack algorithms, compared to seven baseline methods. The goal of the second one is to investigate the impact of the number of ascent steps and training epochs on the performance. In the third experiment, we would like to better understand the interpretability of adversarial training via visualizations.

Three widely-used text classification datasets were used for evaluation: Stanford Sentiment Treebank (SST-2) (Socher et al., 2013), AG-News corpus (AGNEWS) (Zhang et al., 2015) and Internet Movie Database (IMDB) (Maas et al., 2011). SST-2 has about $67,000$ sentences for binary categories, and IMDB consists of about $50,000$ movie reviews for positive and negative sentiment classification, and AGNEWS is a text classification dataset pertaining to four categories containing about $30,000$ news articles. For fair comparison, we used a BERT-based model (Devlin et al., 2019) as the base model for all the defense methods.

5.1 Implementation Details

We implemented MAWP based on Huggingface Transformers[1]. We chose to use PGD (Madry et al., 2018), FreeLB++ (Li et al., 2021), TA-VAT (Li and Qiu, 2020), and InfoBERT (Wang et al., 2020) as baselines. They are widely-used defense methods against textual adversarial attacks or have been

[1]https://github.com/huggingface/transformers

proposed most recently. We also developed two strong baselines. One is the combination of AWP (Wu et al., 2020) (a weight perturbation method proposed for computer vision tasks) and PGD, denoted as PGD-AWP. Another is the combination of AWP with FreeLB, denoted by FreeLB-AWP. These two baselines were also designed for the ablation study and used to compare our mixed adversarial training method. For fair comparison, we also implemented PGD-AWP and FreeLB-AWP as describe in Algorithm 1 except using AWP instead of the proposed weight perturbation.

The size of the weight perturbation η was set to 1×10^{-5} for all the training methods compared, and that of the accumulated perturbation η_2 was set to 2×10^{-5}. All the experimental results are obtained over three runs with different random initialization.

5.2 Attack Algorithms and Evaluation Metrics

Three representative attack algorithms were used to evaluate the adversarial robustness of models: TextFooler (Jin et al., 2020), BERT-Attack (Li et al., 2020) and TextBugger (Li et al., 2019). We use these attack algorithms reimplemented by TextAttack toolkit [2] (Morris et al., 2020). TextFooler and BERT-Attack adversarially perturb text inputs by synonym-based substitutions, whereas TextBugger can perform adversarial perturbation to inputs at both character and word levels. TextFooler generates synonyms using 50 nearest neighbors in the word embedding space, while BERT-Attack uses BERT to generate synonyms dynamically, and thus no defender can know in advance the synonyms used to replace original words by BERT-Attack.

Following Li et al. (2021), we used three metrics to evaluate our methods and other competitors: *clean accuracy* (the accuracy of models on clean examples) is denoted as *Clean*%, *accuracy under attack* (the accuracy of models on adversarial examples under a certain attack) denoted as *Aua*%; and *number of queries* (the average number of queries an attacker needs to perform successful attacks) denoted as *#Query*.

The clean accuracy is calculated on the entire test set while the other two metrics are evaluated on the $1,000$ examples randomly sampled from the test set. In Table 1, we also report the experimental results under the attack algorithms imposed with some constraints suggested by Li et al. (2021) to ensure the quality of adversarial examples generated. For all the attack algorithms, the maximum percentage of words that are allowed to be modified is set to 0.2 on SST-2, 0.3 on AGNEWS, and 0.1 on IMDB. We set the minimum semantic similarity between original sample and adversary to 0.84, set the maximum number of one word's synonyms to 50 and set the maximum number of queries to the victim model for each sample text to $50L$, where L serves as the length of sample text. For a fair comparison, the number of ascent steps K was set to 10 for all adversarial training methods considered.

5.3 Experimental Results

From the numbers reported in Table 1, a handful of trends are readily apparent: (1) MAWP consistently outperforms all the competitors by a significant margin on the adversarial data across three different attacks on SST-2 and IMDB datasets, and it also achieves comparable results on the predication accuracy of clean examples; (2) Although the gap in the accuracy under attack between MAWP and other baseline methods is slightly reduced under the attack algorithms imposed by some constraints as recommended by (Li et al., 2021), the adversaries require much more queries to find adversarial examples for the models trained with MAWP. Note that the greater the average number of queries required by the adversaries is, the more difficult it is for the defense model to be compromised; (3) MAWP outperforms most of the competitors and achieves a relatively high clean accuracy of 95.23 on the test set of AGNEWS.

5.4 Impact of Different Training Epochs

The accumulated weight perturbation was introduced to accelerate the training process and further improve the adversarial robustness of models. To evaluate its effectiveness, we implemented a variant of MAWP, denoted as "MAWP w/o Accum", in which the accumulation operation is not used (i.e., reset the value of $\boldsymbol{\epsilon_g}$ to 0 at every minibatch).

As shown in Figure 1, we found that: (1) Both MAWP and its variant outperform FreeLB++ and FreeLB-AWP in the accuracy under attack especially after 10 epochs; (2) MAWP requires fewer training epochs than its variant of "MAWP w/o Accum" in order to achieve the same or similar *Aua*%; (3) The robustness of the model trained with FreeLB-AWP still can be improved even after 35 epochs while the *Aua*% of the model trained

[2] https://github.com/QData/TextAttack

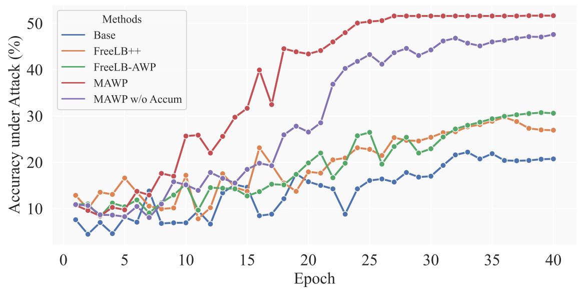

Figure 1: Accuracy under attack versus the number of training epochs. The experiments were conducted on SST-2 under the attack algorithm of TextFooler.

with FreeLB++ drops obviously at the end of the training process.

5.5 Impact of the Number of Ascent Steps

We would like to understand how the choice of the number of ascent steps K impacts the adversarial robustness of models. MAWP was compared to FreeLB++ with different numbers of ascent steps, and the results on SST-2 dataset are reported in Table 2.

Methods	Step K	*Clean%*	TextFooler*	Bert-Attack*	TextBugger*
FreeLB++	3	92.53	24.67	22.00	26.07
	5	**93.10**	27.53	24.87	28.90
	10	92.17	33.83	27.63	33.93
	15	91.14	**36.20**	**28.27**	**37.40**
MAWP	3	**92.53**	30.27	27.27	29.47
	5	**92.53**	36.80	30.27	35.03
	10	92.07	**42.60**	**34.23**	**40.00**
	15	91.21	38.43	29.10	36.80

Table 2: The impact of different ascent steps in clean accuracy and adversarial robustness on the validation set of SST-2.

MAWP outperforms FreeLB++ in all the cases of K under three different attack algorithms. As the number of ascent steps grows, the clean accuracy of the two models drops to 91.14 and 91.21 respectively, and the performance in adversarial robustness also drops slightly. It shows that too much perturbation does harm the model's clean accuracy and adversarial robustness. Therefore, we chose to set $K = 10$ in all the experiments except those reported in this subsection.

5.6 Input Loss Landscape

To give a reasonable explanation of the effect of an enlarged number of ascent steps, we visualized the input loss landscapes produced by the models trained with FreeLB++ and MAWP with the step $K \in \{5, 10, 15\}$. The visualization was provided

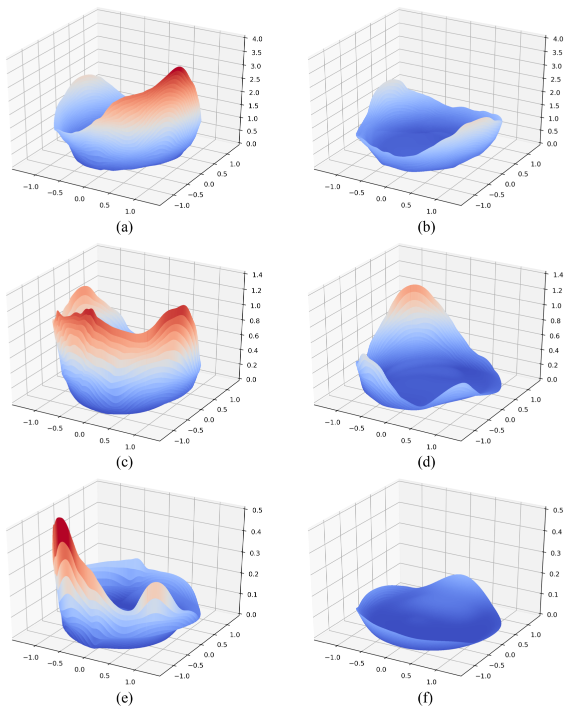

Figure 2: The input loss landscapes produced by the models trained with FreeLB++ and MAWP. Sub-figures (a), (c), and (e) are the loss landscapes produced by the models trained with FreeLB++-5, FreeLB++-10, and FreeLB++-15 respectively, while sub-figure (b), (d), (f) are the loss landscapes produced by those trained with MAWP-5, MAWP-10, and MAWP-15 respectively.

by the models trained on SST-2 dataset.

Specifically, we perturb the original input embedding $\boldsymbol{x}$ to $\boldsymbol{x} + \alpha\boldsymbol{\delta}_1 + \beta\boldsymbol{\delta}_2$, where $\boldsymbol{\delta}_1$ and $\boldsymbol{\delta}_2$ denote two random Gaussian direction vectors with normalization, and α and β are two scalar parameters. Then, the corresponding input loss landscapes can be obtained and are shown in Figure 2. As we can see from Figure 2-(a), (c) and (e), the input loss landscapes will gradually become more flattened as the number of ascent steps K grows when the BERT-based model is trained by FreeLB++ (an enhanced adversarial training method of FreeLB). The similar trend also can be observed from Figure 2-(b), 2(d) and 2(f) when MAWP is used to train the models. As the number of ascent steps K grows, the region with lower loss will become larger, leading to more robust models. However, too strong perturbations without the norm-bounded constraints will distort the decision boundary of the models, which reduces the models' generalization and robustness.

To sum up, a larger number of ascent steps will make the input landscape flatter and enable the model less impacted by the adversarial perturbations. However, the model with a too flat landscape (i.e., an over-smooth model) will result in poor generalization and robustness.

5.7 The Features Captured with and without Weight Perturbation

We want to gain a deeper understanding of why the weight perturbation will lead to more robust models. We examined the differences in the feature vectors produced by the resulting models between original and perturbed examples. Given a neural model f, for an original example x we obtain a perturbed example $\hat{x}$ that stays very close to the decision boundary of the model f and $f(x) = f(\hat{x})$. Note that the smaller the difference, the harder a model can be compromised by the adversaries, and the more robust the model will be generally.

We chose to use TextFooler* as the attack algorithm when generating such perturbed examples. Note that TextFooler* was imposed by some constraints highly recommended by (Li et al., 2021). We used this variant of TextFooler because we want the results yielded by different defense methods can be compared in a more controlled manner. We define the following distance function $\mathcal{G}(x, \hat{x})$ to measure the difference between the feature vectors of x and $\hat{x}$:

$$\mathcal{G}(x, \hat{x}) = \frac{\|h^j(x) - h^j(\hat{x})\|_2}{\|h^j(x)\|_2} \tag{9}$$

where $h^j(x)$ denotes the averaged feature vector extracted from the j-th layer of the BERT-based model for an input x. We chose to use l_2-norm for calculating the distances since the size of perturbation in the embedding space is usually measured based on l_2-norm for almost all the textual attack algorithms.

We first investigated a set of an input x and its quasi-adversarial example $\hat{x}$ (stays very close to the decision boundary but does not make the model change its prediction yet) produced by different defense methods where $\hat{x}$ was generated against the BERT-based model. The reported differences are averaged over all the examples in SST-2 test set. As shown in Figure 3-(a), we found that: (1) The distances between the features of x and $\hat{x}$ generated at the 0-th layer (i.e., embeddings) are quite similar for almost all the models except InfoBERT, which aggressively compresses the embeddings; (2) At the deeper layers, the differences in this distance gradually increase among the models. More

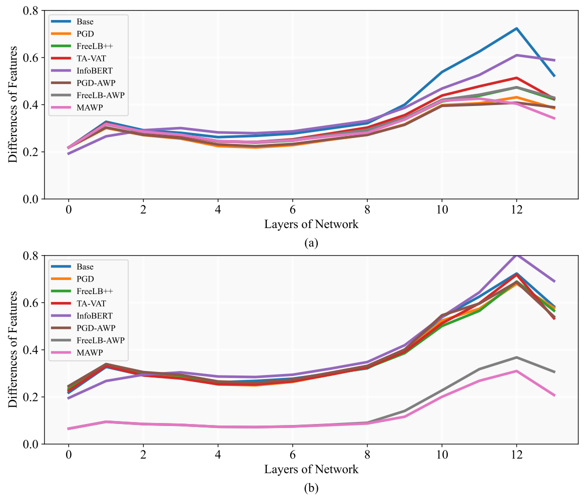

Figure 3: The differences in the feature vector representations between original and perturbed examples produced at each BERT layer on SST-2, where the number "0" denotes the word embedding layer, "13" the output layer, and the numbers in between the hidden layers. (a) The perturbed examples were generated against the BERT-based model trained without any adversarial training method; (b) The perturbed examples were generated against respective models under the text-time attack.

adversarially robust models have a lower distance between the features of x and $\hat{x}$ generated at every layer, especially for the last few layers.

We also examined the differences between the feature vectors of original examples and perturbed ones generated against respective models (i.e., under test-time attacks). As shown in Figure 3-(b), the model trained with MAWP archived a relatively low distance in the generated feature space compared to other adversarial training methods, indicating that MAWP can keep such a distance smaller in the feature space produced at every network layer, which makes the model trained with MAWP more resistant to the perturbations imposed in the inputs.

6 Conclusion

This study is among the first ones to explore the feasibility of improving the adversarial robustness of neural NLP models by performing the perturbations in the parameter space (i.e., weights) rather than the input feature space (i.e., word embeddings). We experimentally demonstrate that the weight perturbation can be used to find a better solution in the parameter space by minimizing the adversarial loss with a multi-step, gradient-guided optimization method. We also show that the proposed method is complementary to existing adversarial training methods, and the combination of our method and FreeLB achieved state-of-the-art accuracy on both clean and adversarial examples on multiple benchmark datasets.

Limitations

We evaluated the proposed mixed adversarial training method with accumulated weight perturbation (MAWP) under the word substitution-based attacks only in this study. We are aware that there are a wide range of textual adversarial attacks, including adding, deleting or modifying characters or words or other language units under certain semantics-preserving constraints. In the future, we would like to investigate how well the MAWP can defend against other types of adversarial attacks. In the current implementation of the MAWP, the weight perturbation needs to be calculated and applied at each adversarial training step, which requires a relatively longer training time. It is also planned to implement the weight perturbations for training NLP models in a more efficient way.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments. This work was supported by National Science Foundation of China (No. 62076068), Shanghai Municipal Science and Technology Major Project (No. 2021SHZDZX0103), and Zhangjiang Lab. Chang is supported in part by Cisco and Sloan fellowship. Hsieh is supported in part by NSF IIS-2008173 and IIS-2048280.

References

Minhao Cheng, Wei Wei, and Cho-Jui Hsieh. 2019. Evaluating and enhancing the robustness of dialogue systems: A case study on a negotiation agent. In *NAACL*.

Minhao Cheng, Jinfeng Yi, Pin-Yu Chen, Huan Zhang, and Cho-Jui Hsieh. 2020. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

Xinshuai Dong, Anh Tuan Luu, Rongrong Ji, and Hong Liu. 2021. Towards robustness against natural language word substitutions. In *ICLR*.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. 2021. Sharpness-aware minimization for efficiently improving generalization. In *ICLR*.

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

Alex Graves. 2011. Practical variational inference for neural networks. In *NIPS*.

Zhezhi He, Adnan Siraj Rakin, and Deliang Fan. 2019. Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack. In *2019 CVPR*.

Po-Sen Huang, Robert Stanforth, Johannes Welbl, Chris Dyer, Dani Yogatama, Sven Gowal, Krishnamurthy Dvijotham, and Pushmeet Kohli. 2019. Achieving verified robustness to symbol substitutions via interval bound propagation. In *EMNLP*, Hong Kong, China.

Robin Jia, Aditi Raghunathan, Kerem Göksel, and Percy Liang. 2019. Certified robustness to adversarial word substitutions. In *EMNLP*.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is BERT really robust? a strong baseline for natural language attack on text classification and entailment. *AAAI*.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2019. Textbugger: Generating adversarial text against real-world applications. *Proceedings 2019 Network and Distributed System Security Symposium*.

Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020. BERT-ATTACK: Adversarial attack against BERT using BERT. In *EMNLP*, Online.

Linyang Li and Xipeng Qiu. 2020. Tavat: Token-aware virtual adversarial training for language understanding. *arXiv preprint arXiv:2004.14543*.

Zongyi Li, Jianhan Xu, Jiehang Zeng, Linyang Li, Xiaoqing Zheng, Qi Zhang, Kai-Wei Chang, and Cho-Jui Hsieh. 2021. Searching for an effective defender: Benchmarking defense against adversarial word substitution.

Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. 2018. Deep text classification can be fooled. In *IJCAI'18*. AAAI Press.

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*, Portland, Oregon, USA.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *ICLR*. OpenReview.net.

Takeru Miyato, Andrew M. Dai, and Ian Goodfellow. 2017. Adversarial training methods for semi-supervised text classification.

John Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp. In *EMNLP*.

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

Motoki Sato, Jun Suzuki, Hiroyuki Shindo, and Yuji Matsumoto. 2018. Interpretable adversarial perturbation in input embedding space for text.

Chenglei Si, Zhengyan Zhang, Fanchao Qi, Zhiyuan Liu, Yasheng Wang, Qun Liu, and Maosong Sun. 2021. Better robustness by more coverage: Adversarial training with mixup augmentation for robust fine-tuning.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks.

Boxin Wang, Shuohang Wang, Yu Cheng, Zhe Gan, Ruoxi Jia, Bo Li, and Jingjing Liu. 2020. InfoBERT: Improving robustness of language models from an information theoretic perspective. *arXiv preprint arXiv:2010.02329*.

Dongxian Wu, Shu-Tao Xia, and Yisen Wang. 2020. Adversarial weight perturbation helps robust generalization. In *NeurIPS*.

Mao Ye, Chengyue Gong, and Qiang Liu. 2020. SAFER: A structure-free approach for certified robustness to adversarial word substitutions. In *ACL*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *NIPS*.

Zhengli Zhao, Dheeru Dua, and Sameer Singh. 2018. Generating natural adversarial examples. In *ICLR*.

Xiaoqing Zheng, Jiehang Zeng, Yi Zhou, Cho-Jui Hsieh, Minhao Cheng, and Xuanjing Huang. 2020. Evaluating and enhancing the robustness of neural network-based dependency parsing models with adversarial examples. In *ACL*.

Yi Zhou, Xiaoqing Zheng, Cho-Jui Hsieh, Kai-Wei Chang, and Xuanjing Huang. 2021. Defense against synonym substitution-based adversarial attacks via Dirichlet neighborhood ensemble. In *ACL*.

Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. 2020. FreeLB: Enhanced adversarial training for natural language understanding. In *ICLR*.