LexicalAT: Lexical-Based Adversarial Reinforcement Training for Robust Sentiment Classification

Jingjing Xu[1]*, Liang Zhao[2]*, Hanqi Yan[2], Qi Zeng[1], Yun Liang[3], Xu Sun[1,2]
[1] MOE Key Lab of Computational Linguistics, School of EECS, Peking University
[2] Center for Data Science, Beijing Institute of Big Data Research, Peking University
[3] Center for Energy-efficient Computing and Applications, Peking University
{jingjingxu,zhao1iang,hanqi.yan,pkuzengqi}@pku.edu.cn
{ericlyun,xusun}@pku.edu.cn

Abstract

Recent work has shown that current sentiment classification models are fragile and sensitive to simple perturbations. In this work, we propose a novel adversarial training approach, LexicalAT, to improve the robustness of current sentiment classification models. The proposed approach consists of a generator and a classifier. The generator learns to generate examples to attack the classifier while the classifier learns to defend these attacks. Considering the diversity of attacks, the generator uses a large-scale lexical knowledge base, WordNet, to generate attacking examples by replacing some words in training examples with their synonyms (e.g., sad and unhappy), neighbor words (e.g., fox and wolf), or super-superior words (e.g., chair and armchair). Due to the discrete generation step in the generator, we use policy gradient, a reinforcement learning approach, to train the two modules. Experiments show LexicalAT outperforms strong baselines and reduces test errors on various neural networks, including CNN, RNN, and BERT.[1]

1 Introduction

Sentiment classification is a fundamental research area in natural language processing (Pang et al., 2002; Glorot et al., 2011; Lai et al., 2015; Kiritchenko and Mohammad, 2018; Liu et al., 2018; Chen et al., 2018). With the development of deep learning, neural networks have obtained state-of-the-art results on many sentiment classification datasets (Kim, 2014; Dong et al., 2014; Tang et al., 2015). However, despite the promising results, recent work has shown that these models easily fail in adversarial examples[2] with little perturbations on real examples. This phenomenon shows that current sentiment classification models have poorly learned the true underlying patterns that determine the correct label. The over-fitting problem still needs to be further explored.

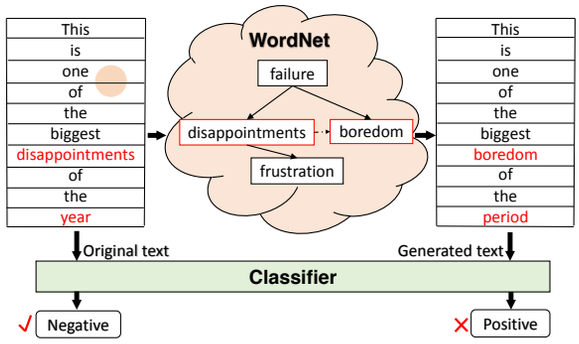

Figure 1: An attacking example for sentiment classification generated by the proposed approach. A pre-trained classifier correctly predicts the label of the original text but fails on the generated text.

Several approaches have been proposed in recent years for the attack problem. These studies can be roughly classified into two categories, data augmentation based approaches and adversarial training based approaches. The key idea of the former approaches is to assist the training of the classifier by augmenting the training data with pre-designed examples (Wang and Yang, 2015; Jia and Liang, 2017; Iyyer et al., 2018). Adversarial training based approaches (Miyato et al., 2017) aim to improve the generalization ability by adding random noises to word embeddings. Although these methods are good pioneering work, they either heavily rely on human knowledge or suffer from low diversity of attacks, which limits the robustness to diverse words and expressions.

In this work, we propose a lexical-based adversarial reinforcement training framework, LexicalAT, for robust sentiment classification. Compar-

*Equal Contribution.

[1]The code will be released at https://github.com/lancopku/LexicalAT

[2]Adversarial examples are intentionally designed by attackers to cause the model to make a mistake.

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pages 5518–5527, Hong Kong, China, November 3–7, 2019.

ing to previous studies, our approach can generate diversely attacking examples with self-learned policies. Our framework contains a generator and a classifier. The generator provides examples to attack the classifier while the classifier learns to defend these attacks. To generate diverse examples, we propose to use a large lexical knowledge base, WordNet, to add perturbations on training examples by replacing some words with their synonyms (e.g., sad and unhappy), neighbor words (e.g., fox and wolf), or super-superior words (e.g., chair and armchair)[3], as shown in Figure 1. Specifically, the output of the generator is a sequence of actions deciding which words should be replaced and their replacements. By involving the attacking examples into the training, the classifier will be more robust and powerful.

Since the generator has discrete generation steps, the gradient-based approach cannot be directly used to back-propagate the errors. We consider policy gradient, a reinforcement learning approach, to train the generator. With the feedback from the classifier as the reward, the generator is encouraged to generate tougher examples for the classifier. In return, with the increasing number of hard examples for training, the classifier becomes more robust and powerful. We evaluate the proposed approach on four popular sentiment classification datasets. Experiments show that LexicalAT outperforms strong baselines on various models and various datasets.

- We propose a lexical-based adversarial reinforcement training approach, LexicalAT, to improve the robustness of sentiment classification models.
- To the best of our knowledge, it is the first work of combining a knowledge base and adversarial learning. The knowledge base contributes to diverse example generation and the adversarial learning develops the attacking policy.
- Experiments show that LexicalAT outperforms strong baselines and improves results on various models, including CNN, RNN, and BERT.

[3]These relations are optional. If a relation has a high risk of changing labels in specific datasets, it is feasible to drop it.

2 Related Work

In this work, we focus on single-label sentiment classification where the input is a word sequence and the output is a single label. In many sentiment classification datasets, neural networks have achieved promising results, even comparable or super to humans. However, several studies have noted that these models are vulnerable and the performance is very sensitive to simple perturbations (Szegedy et al., 2014; Huang et al., 2017; Yuan et al., 2017).

Based on these findings, some studies have been proposed to improve the robustness of neural networks. These studies can be roughly classified into two categories, data augmentation based approaches and adversarial training based approaches. The main idea of the data augmentation based approaches is to augment the training data with pre-designed adversarial examples. Iyyer et al. (2018) propose a paraphrase method to generate syntactically adversarial examples for machine translation tasks. To explore semantically adversarial examples, Kobayashi (2018) replaces input words in real examples with the word predicted by a label-conditional language model. In these approaches, the attacking policy is task-specific and elaborately designed by humans. Unlike these approaches, Miyato et al. (2017) propose to use an adversarial training framework to attack the classifier by adding perturbations to the word embedding layer. However, the unchanged input text makes it hard to improve the robustness to diverse words and expressions.

In this work, we propose a new adversarial reinforcement training framework that aims to generate diverse attacks with self-learned policies. We build a generator that acts as a policy learner to automatically learn to attack the classifier. To generate diverse examples, we include WordNet, a large-scale lexical knowledge base, into the generator for word replacement.

3 Approach

3.1 Overview

Figure 2 shows the overall structure of LexicalAT. Given a sentence, the generator first generates a sequence of actions to replace some words with substitutes in WordNet and build a new example. Then, the new example is sent to the classifier to get the action reward. If the generated example

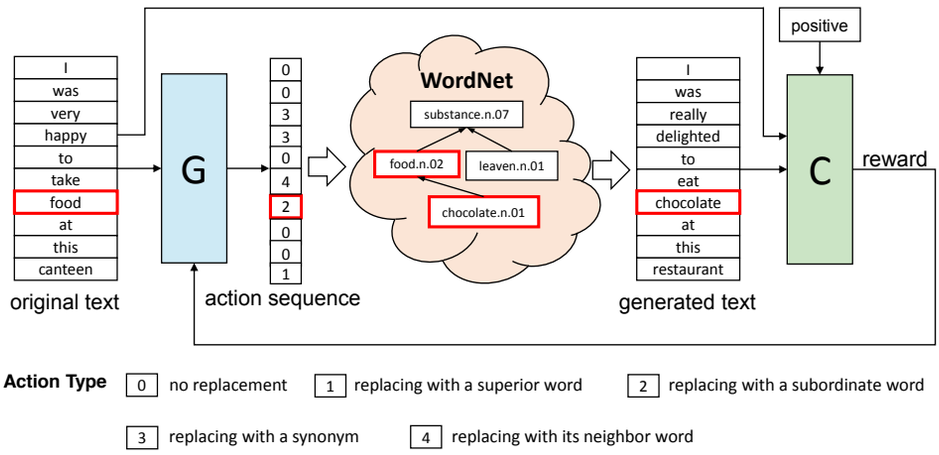

Figure 2: An illustration about the LexicalAT framework. The generator (G) generates an action sequence to replace some words in a training example with new words in WordNet. The classifier (C) calculates the reward of the generated example and returns it to the generator. **G** maximizes the expectation of the reward of generated examples by policy gradient and C minimizes the cross entropy loss between real and generated examples.

successfully confuses the classifier and decreases the probability of the original label, we regard it as a good example and give it a high reward. In this way, the generator is encouraged to generate tough examples for the classifier and the challenging generated examples are used in the training process of the classifier. By alternatively training the generator and the classifier, the classifier is trained to be more robust and powerful.

3.2 Generator

The generator generates attacking examples by adding noises on real examples with WordNet. WordNet is a large English lexical database. Nouns, verbs, adjectives and adverbs are grouped into cognitive synonyms sets (synsets). Each synset represents a distinct concept. As shown in Table 1, WordNet has two basic relations:

- Synonymy. It is the most basic relation, because WordNet uses sets of synonyms (synsets) to represent word senses.
- Hyponymy and hypernymy (super-subordinate). They are transitive relations between synsets. Because there is usually only one hypernym, the semantic relation hierarchically organizes the meanings of nouns.

The generation process is defined as a sequence labeling task for simplification. It takes a text sequence as input and an action sequence as output. Specifically, we define five actions representing the replacement decision, whether a word

Relation	Syntactic Category	Examples
Synonymy (similar)	N, V, Aj, Av	(pipe, tube) (rise, ascend) (sad, unhappy)
Hyponymy and Hypernymy (super-subordinate)	N	(sugar maple, maple) (maple, tree) (tree, plant)
Note: N = Nouns, Aj = Adjectives, V = Verbs, Av = Adverbs		

Table 1: Basic relations in WordNet.

Action	Description	Example
0	No replacement	N/A
1	Replacing with a super word	hamburger vs. sandwich
2	Replacing with a subordinate word	fish vs. salmon
3	Replacing with a synonymy	disappointed vs. frustrated
4	Replacing with a neighbor word	elephant vs.donkey

Table 2: Actions defined in this work. The neighbor word is the word that shares the same super word with the word to be replaced.

should be replaced or its replacement type, as shown in Table 2. For example, if a word is labeled with action "2", we choose a word from its subordinate words with the highest frequency as the replacement. Formally, assume that the input is a sequence of words $\boldsymbol{x} = \{x_1, x_2, \cdots, x_n\}$ where n is the length of input text. This module generates a sequence of replacement actions $\boldsymbol{a} = \{a_1, a_2, \cdots, a_n\}$. Then, by querying WordNet based on $\boldsymbol{x}$ and $\boldsymbol{a}$, we can get a new sentence. Since the proposed framework is independent of the structure of the generator, for simplification we use a traditional sequence labeling model, Bi-Directional Long-Short Term Memory Network

(Bi-LSTM) (Hochreiter and Schmidhuber, 1997), as implementation.

3.3 Classifier

In this work, we focus on single-label sentiment classification. The input is a sequence of words and the output is a label from a pre-defined set $Y = \{y_1, y_2, \cdots, y_k\}$, where k is the number of labels. To evaluate the effectiveness of the proposed approach on various settings, we implement two widely-used sentiment classification networks, Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN) (Kim, 2014), and one state-of-the-art pre-trained model, BERT (Devlin et al., 2018).

In the RNN-based classifier, the input word embeddings are fed into a single Long-Short Term Memory Network (LSTM). Then, a feed-forward layer transfers the last hidden vector of LSTM into the probability distribution of labels.

In the CNN-based classifier, we first feed the word embeddings into a convolutional layer with four convolutional filters. Then, the concatenated filter output applies a max-pooling and is fed into a two-layer feed-forward network with ReLU, followed by the softmax function.

In the BERT-based classifier, we use a model pre-trained on a large-scale language modeling dataset for initialization. Following the work (Devlin et al., 2018), we take the final hidden state for the first token in the input as sentence representation for classification. We use the released model and code, BERT-large, as implementation, which is based on 24 transformer layers.[4]

3.4 Adversarial Reinforcement Training

Due to the discrete choice in generation steps, we use policy gradient, a reinforcement learning approach, to adversarially train the two modules. The generator can be viewed as the agent which interacts with the classifier that acts as environment. The generator improves itself by maximizing the reward returned from the environment.

Given a real example $(\boldsymbol{x}, y)$, the generator first samples an action sequence $\hat{\boldsymbol{a}}$ based on the following probability distribution:

$$\hat{\boldsymbol{a}} \sim p_{\mathbf{G}}(\boldsymbol{a}|\boldsymbol{x}, \boldsymbol{\theta}), \tag{1}$$

where $\boldsymbol{\theta}$ represents the parameter of the generator.

[4]The pre-trained model is provided by goo.gl/language/bert

Algorithm 1 Adversarial reinforcement training.

Require: Generator $\mathbf{G}$; classifier $\mathbf{C}$; training corpus $\mathcal{D}$
Initialize $\mathbf{G}$ and $\mathbf{C}$ with random weights
Pre-train $\mathbf{C}$ using Eq. (6) on $\mathcal{D}$
repeat
  for warm-steps **do**
    Sample an action sequence $\hat{\boldsymbol{a}}$ using Eq. (1)
    Build a new example $(\boldsymbol{x'}, y)$ using $\hat{\boldsymbol{a}}$
    Compute reward $r(\hat{\boldsymbol{a}})$ by Eq. (3)
    Update parameters of $\mathbf{G}$ by Eq. (4)
  end for
  for rl-steps **do**
    Sample an action sequence $\hat{\boldsymbol{a}}$ using Eq. (1)
    Build a new example $(\boldsymbol{x'}, y)$ using $\hat{\boldsymbol{a}}$
    Compute reward $r(\hat{\boldsymbol{a}})$ by Eq. (3)
    Update parameters of $\mathbf{G}$ by Eq. (4)
    Update parameters of $\mathbf{C}$ by Eq. (5)
    Update parameters of $\mathbf{C}$ by Eq. (6)
  end for
until convergence

Based on the sampled action sequence $\hat{\boldsymbol{a}}$, we get a new example $(\boldsymbol{x'}, y)$ after word replacement.

Then, we feed the real example $(\boldsymbol{x}, y)$ and the generated example $(\boldsymbol{x'}, y)$ into the classifier to get the replacement reward $r(\hat{\boldsymbol{a}})$. Specifically, we define the reward as the absolute difference of the probability of y between the real example and the generated example:

$$r(\hat{\boldsymbol{a}}) = \log p_{\mathbf{C}}(y|\boldsymbol{x}; \boldsymbol{\phi}) - \log p_{\mathbf{C}}(y|\hat{\boldsymbol{x}}; \boldsymbol{\phi}), \tag{2}$$

where $\boldsymbol{\phi}$ is the parameter of the classifier. If the generated example successfully confuses the classifier and decreases the probability of y, we regard it as a good example and give it a high reward. In practice, we use the following equation to get a new reward to train the generator steadily:

$$r'(\hat{\boldsymbol{a}}) = r(\hat{\boldsymbol{a}}) - \mathbb{E}_{P_G(\boldsymbol{a}|\boldsymbol{x},\boldsymbol{\theta})}(r(\boldsymbol{a})), \tag{3}$$

where $\mathbb{E}_{P_G(\boldsymbol{a}|\boldsymbol{x},\boldsymbol{\theta})}(r(\boldsymbol{a}))$ is the expectation of $r(\boldsymbol{a})$.

Then, the reward is fed back to the generator. Formally, the expected gradient of the parameter $\boldsymbol{\theta}$ can be approximated as:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}}\mathcal{L} \approx & - r'(\hat{\boldsymbol{a}}) \nabla_{\boldsymbol{\theta}} \log p_{\mathbf{G}}(\hat{\boldsymbol{a}}|\boldsymbol{x}; \boldsymbol{\theta}) \\ & - \mathbb{E}_{p_{\mathbf{G}}(\hat{\boldsymbol{a}}|\boldsymbol{x};\boldsymbol{\theta})} r'(\hat{\boldsymbol{a}}), \end{aligned} \tag{4}$$

where $\hat{\boldsymbol{a}}$ is the generated action sequence. The second term $\mathbb{E}_{p_{\mathbf{G}}(\hat{\boldsymbol{a}}|\boldsymbol{x};\boldsymbol{\theta})} r'(\hat{\boldsymbol{a}})$ means we approximate the expectation with sampling in practice.

The generated example $(\boldsymbol{x'}, y)$ is then used to train the classifier by minimizing the following cross-entropy loss:

$$\mathcal{L} = - \log p_{\mathbf{C}}(y|\boldsymbol{x'}, \boldsymbol{\phi}). \tag{5}$$

Furthermore, to prevent the classifier from forgetting the knowledge of training data, we also use teacher forcing (Goyal et al., 2016) to train the classifier. After each reinforcement training step, we run a teacher forcing step by directly using the real example $(\boldsymbol{x}, y)$ to train the classifier:

$$\mathcal{L} = -\log p_{\mathbf{C}}(y|\boldsymbol{x}, \boldsymbol{\phi}). \quad (6)$$

Considering that reinforcement learning process requires the two modules with the initial learning ability, we pre-train the generator and the classifier before the reinforcement learning stage. The classifier is directly pre-trained on real training data until convergence. Due to the lack of supervisory signals, we build a warm-up stage to pre-train the generator. In the warm-up stage, the feedback of the pre-trained classifier is used to train the generator.

4 Experiments

We evaluate LexicalAT on four sentiment classification datasets. We first introduce datasets, implementation details, and baselines. Then, we show experiment results and provide detailed analysis.

4.1 Datasets

SST-2 & SST-5. The Stanford sentiment treebank (Socher et al., 2013) is a single-sentence classification dataset built on movie reviews. Based on the difference of sentiment granularity, the human annotators design two label sets. Following existing work (Kobayashi, 2018), we run experiments on both label sets. For clarify, we call them SST-2 and SST-5.

RT. The rating inference dataset (Pang and Lee, 2005) is another sentiment classification dataset. The data is from internet movie reviews and has two types of labels.

Yelp. The dataset is built upon reviews from website Yelp.[5] Each review has a rating label varying from 1 to 5. We randomly select 100K for training, 10K for validation, and 10K for testing. It is used to verify whether the proposed approach is applied to tasks with large-scale data.

The dataset statistics are shown in Table 3. For SST-2 and SST-5, we use the standard split in their work. For RT, due to the lack of the standard split, we randomly divide all examples into 90% for the training set and 10% for the test set. To build the development set, we randomly take out 10% from the training set.

[5]https://www.yelp.com/dataset/challenge

Dataset	#Class	Avg. #w	Train	Dev	Test
SST2	2	19	6,920	872	1,821
SST5	5	18	8,544	1,101	2,210
RT	2	21	8,608	964	1,089
Yelp	5	89	100,000	10,000	10,000

Table 3: Dataset statistics. "Class" is the number of pre-defined labels. "Avg. #w" is the average word number in the input text. "Train", "Dev", and "Test" represent the sizes of the training set, the development set, and the test set.

4.2 Baselines

We compare our proposed approach with the following robustness-driven approaches.

SynDA. It is a synonymy based data augmentation approach (Zhang et al., 2015). It uses an English thesaurus, obtained from WordNet, to randomly replace some words in real examples with their synonymys to build new examples.

ConDA. It is a contextual data augmentation approach (Kobayashi, 2018). They build adversarial examples by randomly replacing words in real examples with the words that are predicted by a bidirectional language model at the word positions.

VAT. It is an adversarial training based approach (Miyato et al., 2017) for robust text classification. It adds perturbations to recurrent neural networks to improve the robustness. We use the released code for implementation.

4.3 Experiment settings

Based on the performance on the development sets, we set batch size to 64 except yelp whose batch size is 256. We use the Adam optimizer to train the modules. The details of model-specific hyper-parameter settings are shown in Table 4. In the pre-training stage, we train the classifier until convergence. In the warm-up stage, we train the generator by 3 epoch. In the reinforcement training stage, we set the maximum epcoh to 100 and adopt the early stopping mechanism.

4.4 Results and Discussion

The results of the proposed approach and baselines are shown in Table 5.

	Hyper-Parameter	Setting
Generator	Embedding size	300
	Embedding dropout	0.4
	Hidden size	300
	BiLSTM layer	1
	Learning rate	1e-3
CNN Classifier	Embedding size	300
	CNN layer	1
	CNN output dropout	0.1
	CNN dense layer dropout	0.4
	Filter kernel size	[2,3,4,5]
	Filter kernel dimension	300
	Learning rate	1e-4
RNN Classifier	Embedding size	300
	Embedding dropout	0.4
	Hidden size	300
	LSTM layer	1
	LSTM output dropout	0.1
	Learning rate	1e-4
BERT Classifier	Transformer layer	24
	Embedding size	1,024
	Hidden size	1,024
	Head	16
	Learning rate	2e-5
	Fine-turning epoch	3

Table 4: Settings of model-specific hyper-parameters.

Approach	SST-2	SST-5	RT	Yelp
RNN(our implemented)	80.61	40.54	75.85	60.94
RNN (Kobayashi, 2018)	80.30	40.20	*	*
+SynDA (Zhang et al., 2015)	80.20	40.50	*	*
+ConDA (Kobayashi, 2018)	80.10	41.10	*	*
+VAT (Miyato et al., 2017)	81.16	37.38	75.94	59.69
+LexicalAT (proposed)	**81.60**	**41.99**	**76.12**	**61.18**
Approach	**SST-2**	**SST-5**	**RT**	**Yelp**
CNN(our implemented)	80.62	40.81	75.85	60.77
CNN (Kobayashi, 2018)	79.50	41.30	*	*
+SynDA (Zhang et al., 2015)	80.00	40.70	*	*
+ConDA (Kobayashi, 2018)	80.80	**42.10**	*	*
+VAT (Miyato et al., 2017)	*	*	*	*
+LexicalAT (Proposed)	**81.58**	41.67	**76.22**	**61.86**
Approach	**SST-2**	**SST-5**	**RT**	**Yelp**
BERT(our implemented)	92.60	55.07	88.57	66.76
+SynDA (Zhang et al., 2015)	*	*	*	*
+ConDA (Kobayashi, 2018)	*	*	*	*
+VAT (Miyato et al., 2017)	*	*	*	*
+LexicalAT (proposed)	**93.03**	**55.38**	**88.68**	**67.50**

Table 5: Comparisons between LexicalAT and baselines on four datasets.[6] LexicalAT outperforms strong baselines on all datasets, including three small datasets and a large dataset, Yelp.

As expected, LexicalAT, the proposed approach, substantially outperforms the naive baselines (RNN, CNN and BERT). Since RNN and CNN are directly trained on training sets, they tend to suffer from over-fitting. Therefore, it is reasonable that LexicalAT brings improvements to RNN and CNN models. BERT has better generalization ability because it has been pre-trained on a large scale corpus. Even so, LexicalAT still gains accuracy improvements over BERT by 0.43 on SST-2, 0.31 on SST-5, 0.11 on RT, and 0.74 on Yelp, respectively. These results show that the proposed approach is universal and well applied to various models, even the state-of-the-art pre-trained model BERT.

By contrast, SynDA, the synonymy based data augmentation approach, does not bring significant performance improvements over the naive baselines. The main difference between LexicalAT and SynDA is that SynDA uses the random replacement policy to generate new examples, while LexicalAT uses the dynamic replacement policy learned by the proposed adversarial reinforcement training framework. The performance gap between LexicalAT and SynDA shows that the proposed adversarial reinforcement training framework is effective for learning the attacking policy toward the weaknesses of the classifier. Furthermore, the proposed approach beats VAT under various datasets. Since VAT only adds random noises on word embeddings without changing the input text, it does not augment the training data with new words and expressions, and thus limits the robustness improvement. In summary, with adversarial reinforcement training, lexicalAT is capable of learning the attacking policy toward the weaknesses of the classifier. With WordNet, lexicalAT can generate examples with diverse expressions, which improves the classifier robustness. These advantages make the proposed approach perform well and achieve the best performance on various datasets.

To illustrate the training process, we plot the performance on the SST-5 development set in Figure 3. LexicalAT converges to a higher accuracy than the naive baseline RNN. It shows that the adversarial reinforcement training mechanism is robust and can converge to good results.

Furthermore, we compare the performance of lexicalAT and naive baselines on defending attacks, taking RNN and CNN on SST-2, SST-5, and RT as examples. The results are shown in Table 6. "RNN-Attacking Set" and "CNN-Attacking Set" are two sets generated by two different gen-

[6]The results of SynDA and ConDA are from the work of Kobayashi (2018).

SST-2	Classifier (RNN)	Classifier (CNN)	LexicalAT (RNN)	LexicalAT (CNN)
Test Set	80.61	80.62	81.60	81.58
RNN-Attacking Set	69.91	65.62	76.44	73.70
CNN-Attacking Set	68.81	68.04	74.62	76.28
SST-5	**Classifier (RNN)**	**Classifier (CNN)**	**LexicalAT (RNN)**	**LexicalAT (CNN)**
Test Set	40.54	40.81	41.99	41.67
RNN-Attacking Set	35.16	35.43	38.73	38.91
CNN-Attacking Set	34.98	36.60	37.65	38.96
RT	**Classifier (RNN)**	**Classifier (CNN)**	**LexicalAT (RNN)**	**LexicalAT (CNN)**
Test Set	75.85	75.85	76.12	76.22
RNN-Attacking Set	69.05	68.78	71.44	70.61
CNN-Attacking Set	62.90	61.89	69.88	71.17

Table 6: Comparisons of different classifiers on defending attacks. The classifiers trained in lexicalAT, lexicalAT (RNN) and lexicalAT (CNN), have better defending ability. "Classifier (RNN)" and "Classifier (CNN)" are two different naive baselines trained on real examples. "Lexical (RNN)" and "Lexical (CNN)" are two classifiers trained in the framework of lexicalAT. "Test Set" is the standard test set. "RNN-Attacking Set" and "CNN-Attacking Set" are two attacking sets generated by two generators in the framework of lexicalAT based on real test examples. The two generators are trained based on the interaction with RNN-based and CNN-based classifiers, respectively.

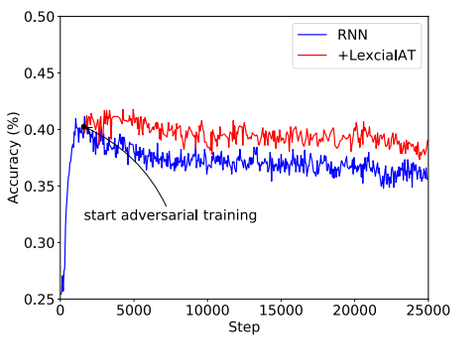

Figure 3: Learning curves of RNN and LexicalAT on the development set of SST-5.

Setting	SST-2	SST-5	RT	Average
all words (RNN)	81.60	41.14	75.76	66.17
-super words	80.72	41.17	75.02	65.64
-subordinate words	80.56	41.67	75.48	65.90
-synonymy words	**80.45**	41.99	76.12	66.19
-neighbor words	80.56	**40.91**	**74.66**	**65.38**
Setting	**SST-2**	**SST-5**	**RT**	**Average**
all words (CNN)	80.18	41.86	74.84	65.63
-super words	79.96	41.67	76.22	65.95
-subordinate words	81.58	41.49	75.39	66.15
-synonymy words	**79.24**	41.67	74.74	**65.22**
-neighbor words	81.33	**40.68**	**74.47**	65.49

Table 7: Effects of different replacement actions. As we can see, neighbor and synonymy words contribute most to the performance.

erators in lexicalAT based on real test examples. The two generators are trained based on the interaction with RNN-based and CNN-based classifiers, respectively. The naive baselines, RNN and CNN, have weak robustness performances and the results drop largely on attacking examples, even a 15% accuracy decrease for "Classifier (CNN)" on SST-2. In comparison, the classifiers trained in lexicalAT, lexicalAT (RNN) and lexicalAT (CNN), have better defending ability.

4.5 Analysis

What is the effect of each action? To show the effect of different actions, we take RNN and CNN on SST-2, SST-5, and RT as examples and conduct experiments by dropping one action at a time. As Table 7 shows, only some actions are useful for the robustness improvement. The average performance becomes better after dropping subordinate words under two different settings. Surprisingly, the neighbor words largely contribute to the performance. Without neighbor words, the average accuracies are dropped from 66.17 to 65.38 for RNN, and from 65.63 to 65.22 for CNN. Neighbor words share the same super word, like "fox" and "wolf". In WordNet, neighbor words usually have similar semantic meanings. By replacing a word with its similar word, the semantic diversity can be largely enhanced. Furthermore, the semantic similarity can reduce the risk of changing the label when modifying the input text. Therefore, it is a good choice to include this relation in models unless the replacement exactly impacts the original label in specific tasks.

Is the learned attacking policy universal or model-specific? To explore this question, we

	Classifier (RNN)	Classifier (CNN)
None	80.81	80.62
Policy-RNN	**81.44**	80.89
Policy-CNN	81.22	**81.27**

Table 8: Results of different classifiers trained with the examples generated by different attacking policies on SST-2 dataset. "Policy-RNN" and "Policy-CNN" are the learned policies from the interaction with RNN-based and CNN-based classifiers in the proposed framework.

conduct the following experiments.

We train two generators based on the feedback of the classifiers with different structures (e.g., RNN and CNN) on SST-2. For clarify, we call the learned attacking policies in the two generators as policy-RNN and policy-CNN, as shown in Table 8. Then, we put the examples generated by policy-RNN and policy-CNN into the training data to train the classifiers with different structures.

For the RNN-based classifier, two attacking policies both outperform the naive RNN baseline, which shows the universality of the learned attacking policy. Furthermore, policy-RNN brings more improvements than policy-CNN does on the RNN-based classifier. It demonstrates that the learned attacking policy has model-specific features. Similar result happens to the CNN-based classifier.

In summary, different classification models share some weaknesses and have their own unique vulnerability. Considering unique weaknesses are difficult to be summarized by human knowledge, our approach is an effective way to automatically learn the attacking policy toward the classifier weaknesses.

4.6 Case Study

Table 9 presents the adversarial examples generated by the generator on SST-5. Even the simple perturbations can confuse the classifier, which shows the weak robustness of the classifier.

4.7 Error Analysis

Although the proposed approach improves the robustness of current classifiers, it is important to note that there still have several problems to be further explored.

First, some generated examples contain low-quality phrases, such as the bottom one in Table 9. This problem is due to the inappropriate word replacement. In the future work, we would like to explore how to generate diverse and high-quality examples by taking sentence context, replaced word features, and replacing word features into consideration.

Original Text	The **movie** is without **intent**.
Generated Text	The **film** is without **spirit**.
Original Text	The **script** is smart not **cloying**.
Generated Text	The **dialogue** is smart not **saccharine**.
Original Text	This is a gorgeous **film** vivid with **color music** and **life**.
Generated Text	This is a gorgeous **movie** vivid with **gloss sound** and **spirit**.
Original Text	Hollywood ending is the most **disappointing** woody allen **movie** ever.
Generated Text	Hollywood ending is the most **fail** woody allen **film** ever.

Table 9: Adversarial examples generated by our approach.

Second, we regard the examples that confuse the classifier as good attacking examples and give them high reward to train the generator. However, not all confusing examples are useful for robustness improvements. To optimize the training time, in the future we will explore an advanced LexicalAT to automatically distinguish between "useful examples" and "useless examples".

5 Conclusion and Future Work

In this work, we propose a new adversarial training approach, LexicalAT, to improve the robustness of current sentiment classification models. The key idea is to use WordNet and adversarial reinforcement training to automatically learn the diversely attacking policy. We evaluate LexicalAT on four representative sentiment classification datasets. Experiments demonstrate that the proposed approach has better generality and reduces test errors on various neural networks, including CNN, RNN, and BERT.

In the future work, we would like to build an advanced version of LexicalAT from the following perspectives. First, to address the problem of low-quality phrases in generated text, we will explore how to keep the syntactic correctness of the generated text. Second, we would like to figure out the detailed effects of the generated examples on the robustness of sentiment classification models. By removing useless examples, we can obtain higher training speed.

Acknowledgments

We thank all reviewers for providing the thoughtful and constructive suggestions. This work was supported in part by National Natural Science Foundation of China (No. 61673028). Xu Sun is the corresponding author of this paper.

References

Xilun Chen, Yu Sun, Ben Athiwaratkun, Claire Cardie, and Kilian Q. Weinberger. 2018. Adversarial deep averaging networks for cross-lingual sentiment classification. *TACL*, 6:557–570.

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

Li Dong, Furu Wei, Chuanqi Tan, Duyu Tang, Ming Zhou, and Ke Xu. 2014. Adaptive recursive neural network for target-dependent twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 2: Short Papers*, pages 49–54.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 513–520.

Anirudh Goyal, Alex Lamb, Ying Zhang, Saizheng Zhang, Aaron C. Courville, and Yoshua Bengio. 2016. Professor forcing: A new algorithm for training recurrent networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4601–4609.

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

Sandy Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. 2017. Adversarial attacks on neural network policies. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1875–1885.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 2021–2031.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751.

Svetlana Kiritchenko and Saif Mohammad. 2018. Examining gender and race bias in two hundred sentiment analysis systems. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics, *SEM@NAACL-HLT, New Orleans, Louisiana, USA, June 5-6, 2018*, pages 43–53.

Sosuke Kobayashi. 2018. Contextual augmentation: Data augmentation by words with paradigmatic relations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*.

Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent convolutional neural networks for text classification. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 2267–2273.

Qi Liu, Yue Zhang, and Jiangming Liu. 2018. Learning domain representation for multi-domain sentiment classification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 541–550.

Takeru Miyato, Andrew M. Dai, and Ian J. Goodfellow. 2017. Adversarial training methods for semi-supervised text classification. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

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

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing, EMNLP 2002, Philadelphia, PA, USA, July 6-7, 2002*.

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 *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Duyu Tang, Bing Qin, and Ting Liu. 2015. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1422–1432.

William Yang Wang and Diyi Yang. 2015. That's so annoying!!!: A lexical and frame-semantic embedding based data augmentation approach to automatic categorization of annoying behaviors using #petpeeve tweets. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 2557–2563.

Xiaoyong Yuan, Pan He, Qile Zhu, Rajendra Rana Bhat, and Xiaolin Li. 2017. Adversarial examples: Attacks and defenses for deep learning. *CoRR*, abs/1712.07107.

Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 649–657.