Adversarial Attack and Defense of Structured Prediction Models

Wenjuan Han[1]*, **Liwen Zhang**[2,3,4]*, **Yong Jiang**[5], **Kewei Tu**[2]†
[1]School of Computing, National University of Singapore, Singapore
[2]School of Information Science and Technology, ShanghaiTech University
[3]Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences
[4]University of Chinese Academy of Sciences
[5]Alibaba DAMO Academy, Alibaba Group
dcshanw@nus.edu.sg
{zhanglw1, tukw}@shanghaitech.edu.cn
yongjiang.jy@alibaba-inc.com

Abstract

Building an effective adversarial attacker and elaborating on countermeasures for adversarial attacks for natural language processing (NLP) have attracted a lot of research in recent years. However, most of the existing approaches focus on classification problems. In this paper, we investigate attacks and defenses for structured prediction tasks in NLP. Besides the difficulty of perturbing discrete words and the sentence fluency problem faced by attackers in any NLP tasks, there is a specific challenge to attackers of structured prediction models: the structured output of structured prediction models is sensitive to small perturbations in the input. To address these problems, we propose a novel and unified framework that learns to attack a structured prediction model using a sequence-to-sequence model with feedbacks from multiple reference models of the same structured prediction task. Based on the proposed attack, we further reinforce the victim model with adversarial training, making its prediction more robust and accurate. We evaluate the proposed framework in dependency parsing and part-of-speech tagging. Automatic and human evaluations show that our proposed framework succeeds in both attacking state-of-the-art structured prediction models and boosting them with adversarial training.

1 Introduction

Adversarial examples, which contain perturbations to the input of a model that elicit large changes in the output, have been shown to be an effective way of assessing the robustness of models in natural language processing (NLP) (Jia and Liang, 2017; Belinkov and Bisk, 2018; Hosseini et al., 2017; Samanta and Mehta, 2017; Alzantot et al., 2018; Ebrahimi et al., 2018; Michel et al., 2019; Wang et al., 2019). Adversarial training, in which models are trained on adversarial examples, has also been shown to improve the accuracy and robustness of NLP models (Goodfellow et al., 2015; Tramèr et al., 2017; Yasunaga et al., 2018). So far, most existing methods of generating adversarial examples only work for classification tasks (Jia and Liang, 2017; Wang et al., 2019) and are not designed for structured prediction tasks. However, since many structured prediction tasks such as part-of-speech (POS) tagging and dependency parsing are essential building blocks of many AI systems, it is important to study adversarial attack (generating adversarial examples) and defense (adversarial training) of structured prediction models.

There are multiple challenges that have to be addressed in building an efficient and effective attacker for structured prediction models in NLP. Zhang et al. (2019a) pointed out two major prob-

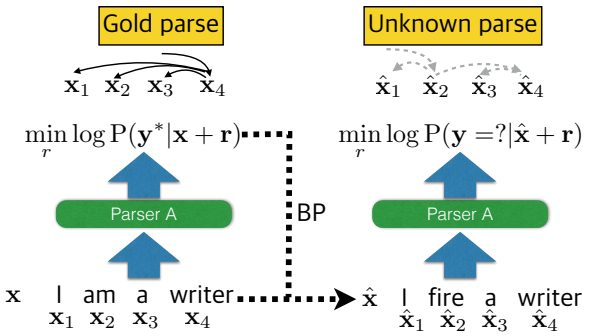

Figure 1: An example showing the challenges in attacking a dependency parser using gradient-based methods. A small perturbation to the sentence x changes one word from "am" to "fires". This change makes the perturbed example *I fires a writer* ungrammatical. Even if the perturbed example is "I fire a writer" that meets the rules of grammar, the true output structure is still different from the input sentence "I am a writer". More importantly, this true parse is unknown to the attacker, which hinders the next update step.

*Equal contributions.
†Corresponding author.

lems encountered by attackers of NLP tasks. First, since words are discrete, making small disturbances to words in the gradient direction is difficult. Secondly, there is no guarantee that the generated adversarial examples are fluent. In addition to these two problems, there is a unique challenge faced by attackers of structured prediction tasks. While small perturbations to images or even texts typically do not change their classification labels, small perturbations to sentences in structured prediction may very likely change the true output structures. In other words, many structured prediction tasks are very sensitive to small perturbations in the input sentence. Consequently, almost all the existing attacking methods are not directly applicable to structured prediction. To illustrate this challenge, we take adversarial attack of dependency parsing as an example (Figure 1). We use the fast gradient sign method (FGSM) (Goodfellow et al., 2015) as the attack method, which is a classic gradient-based attacker that perturbs the input by minimizing the likelihood of the true output. When applied to NLP tasks (Miyato et al., 2017), FGSM perturbs the embeddings of the words in the input sentence and then replaces individual words based on the new embeddings. However, there is no guarantee that the new sentence has the same parse tree as the original sentence. Once the true output parse tree becomes unknown, subsequent updates become impossible in FGSM, resulting in perturbation that might be insufficiently adversarial. In Figure 1, after just one step of perturbation, the sentence indeed has a different parse tree that is unknown to the attacker.

To address the aforementioned problems, we propose to attack structured prediction models with sequence-to-sequence (seq2seq) sentence generators. Before attack, the seq2seq generator is trained by reinforcement learning based on a novelty designed reward function that evaluates the output of the victim structured prediction model against an ensemble of multiple reference models of the same structured prediction task. During attack, the seq2seq generator is simply applied to input sentences to produce adversarial examples. Our framework has the following features.

- Our proposed attacker is a black-box attacker that does not need to know the internal details of the target model (such as the model structure, the hyper-parameters, the training strategy, the training dataset, and gradients over each layer). This ensures that our framework (including attack and defense) can be applied to almost any structured prediction models.
- In contrast to previous black-box attackers, our attacker is an *online* attacker. Once the seq2seq sentence generator is trained, it can generate adversarial examples directly from original sentences during attacks without any optimization procedure and also without the need to access the victim model. This significantly increases the efficiency of the attack.
- Most existing methods perform word or character level manipulations and hence cannot change the sentence length. We use a seq2seq generator to modify the whole sentence without this limitation.
- Our generator can utilize some recent pre-trained language models (e.g., BERT (Devlin et al., 2019), GPT-2 (Radford et al., 2019)) to improve quality of adversarial examples.

We evaluate our framework on the dependency parsing task and the POS tagging task. Both automatic and human evaluations show that our method outperforms previous approaches in attacking state-of-the-art structured prediction models as well as boosting these models with adversarial training for better accuracy and robustness. The code and the trained model can be found at `https://github.com/WinnieHAN/structure_adv`.

2 Background

2.1 Structured Prediction

Structured prediction in NLP aims to predict output variables that are mutually dependent or constrained given an input sentence. We represent the training data with N samples as $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)} : j = 1, ..., N\}$, where $\mathbf{x}^{(j)}$ is the j-th sentence and $\mathbf{y}^{(j)}$ is the corresponding structure. The set of all $\mathbf{x}^{(j)}$ is $\mathcal{X}$. For each $\mathbf{x}$ with length n, it can be written as a sequence of tokens $\{\mathbf{x}_i : i = 1, ..., n\}$. We also define $\mathbf{v}$ to represent the concatenation of all the word vectors in sentence $\mathbf{x}$.

A structured prediction model predicts the output $\mathbf{y}$ given an input sentence $\mathbf{x}$ by maximizing the log conditional probability:

$$\arg\max_{\mathbf{y} \in \mathcal{T}} \log P(\mathbf{y}|\mathbf{x}; \Theta)$$

where $\mathcal{T}$ is the set of all possible outputs and Θ is the set of parameters.

We train the model by minimizing the following loss:

$$\mathcal{L}(\Theta) = -\frac{1}{N} \sum_{(\mathbf{x}^{(j)}, \mathbf{y}^{(j)}) \in \mathcal{D}} \log P(\mathbf{y}^{(j)} | \mathbf{x}^{(j)}; \Theta)$$

2.2 Word-level Adversarial Attack

Goodfellow et al. (2015) proposed the fast gradient sign method (FGSM) in the image processing field, which uses the direction of the gradient to update image pixels and generate adversarial examples. Then Miyato et al. (2017) applied this approach to add perturbations in the word embedding space, though their approach cannot generate adversarial text examples. In order to solve the mapping problem from a modified word vector to a word, word level manipulation is used to replace original words (Papernot et al., 2016). In addition to the replacement manipulation, Samanta and Mehta (2017) introduced two new modification strategies: removal and addition.

2.3 Word-level Adversarial Attack for Structured Prediction

The gradient of the negative log likelihood with respect to the input in a structured prediction model can be leveraged to find adversarial examples. The original input sentence $\mathbf{x}$ is manipulated by adding or subtracting a small adversarial perturbation $\mathbf{r}$ to the vector $\mathbf{v}$. Adding $\mathbf{r}$ in the direction of the gradient means that the sentence is modified to decrease the log likelihood so that the model is less likely to predict the correct output. We use $\hat{\mathbf{x}}$ to represent $\mathbf{x}$ with perturbation.

The following formula describes the adversarial example:

$$\hat{\mathbf{x}} = search(\mathbf{x}, \mathbf{r}) = search(\mathbf{v}, \mathbf{r})$$

where we use $\mathbf{v}$ to represent the concatenation of all the word vectors in sentence $\mathbf{x}$. $search$ is a searching approach to find an adversarial example $\hat{\mathbf{x}}$ according to perturbed vector $\mathbf{v} + \mathbf{r}$ and $\mathbf{r}$ is calculated by maximizing the loss function as follows.

$$\mathbf{r} = \arg \max_{\mathbf{r}, ||\mathbf{r}|| \leq \epsilon} \{-\log P(\mathbf{y} | \mathbf{x} + \mathbf{r}; \Theta)\}$$

where ϵ is a hyper-parameter to control the scale of the perturbation.

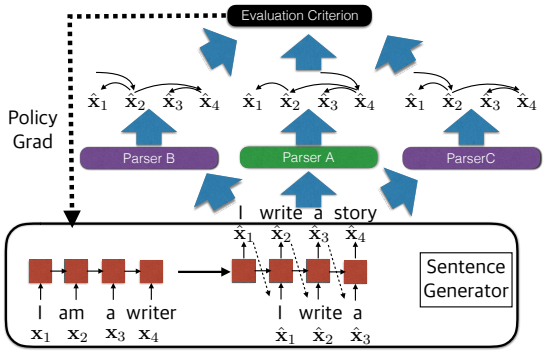

Figure 2: Our framework illustrated on the dependency parsing task. It consists of three parts: a seq2seq generator, an evaluation module (including the reference Parser B, the reference Parser C and the evaluation criteria), and the victim model A.

It is intractable to exactly solve the problem, so an approximate approach is proposed to compute $\mathbf{r}$ as follows:

$$\mathbf{r} = \frac{\epsilon \mathbf{g}}{||\mathbf{g}||}$$
$$\mathbf{g} = sign(\nabla_{\mathbf{v}} \log P(\mathbf{y} | \mathbf{v}; \Theta))$$

To generate natural and legible adversarial sentences, we search in the word embedding space and replace the original word with a word that is closest to the perturbed word vector. However, as discussed in section 1, this approach can only generate perturbed examples using one perturbation step for structured prediction. Moreover, this model cannot guarantee quality (e.g., fluency) of the generating sentences.

3 Sentence-level Adversarial Attack and Defense

We aim to mislead a structured prediction model by generating adversarial examples $\hat{\mathbf{x}}$ from the original examples $\mathbf{x}$ using a seq2seq generator. We train the generator using reinforcement learning following Williams (1992). The reward function for reinforcement learning evaluates whether the generated sentence could induce an incorrect output from the victim model, and the evaluation is facilitated by two reference models. In addition, the reward function also evaluates the quality of the generated sentence. Figure 2 illustrates the overall architecture of our proposed model, which mainly consists of three parts: a generator, an evaluation module, and the victim model. We use dependency parsing as our example structured prediction task and name the victim parser as parser A and the reference parsers as Parsers B and C.

In contrast to the word-level attackers described in section 2.3, our proposed method aims to do sentence-level attacks that are capable of generating a sentence of a different length and structure instead of merely making local word changes.

3.1 Evaluation Criterion for Structured Outputs

Since the structured output is very sensitive to perturbation of the input, the parse of the original sentence $\mathbf{x}$ cannot be treated as the ground truth of the generated adversarial sentence $\hat{\mathbf{x}}$. Without knowing the new ground truth, we would not know if the adversarial sentence can indeed mislead parser A to produce an incorrect prediction. Thus we make use of two reference parsers B and C to evaluate the prediction of parser A and help guide the generation of truly adversarial examples. Intuitively, if B and C produce the same parse tree, then it is more likely to be correct and can be used as ground-truth to evaluate parser A.

Given a generated sentence $\hat{\mathbf{x}}$, if the predicted parse tree $\mathbf{y}^A_{\hat{\mathbf{x}}}$ from parser A is greatly different from the predicted trees $\mathbf{y}^B_{\hat{\mathbf{x}}}$ and $\mathbf{y}^C_{\hat{\mathbf{x}}}$ from parsers B and C, while $\mathbf{y}^B_{\hat{\mathbf{x}}}$ and $\mathbf{y}^C_{\hat{\mathbf{x}}}$ agree with each other, then we think $\hat{\mathbf{x}}$ is a good adversarial example of parser A. The criterion is defined as follows:

$$s_p(\hat{\mathbf{x}}) = -f(\mathbf{y}^A_{\hat{\mathbf{x}}}, \mathbf{y}^B_{\hat{\mathbf{x}}}) - f(\mathbf{y}^A_{\hat{\mathbf{x}}}, \mathbf{y}^C_{\hat{\mathbf{x}}}) + f(\mathbf{y}^B_{\hat{\mathbf{x}}}, \mathbf{y}^C_{\hat{\mathbf{x}}}) \quad (1)$$

where $f(\mathbf{y}, \mathbf{y}^*)$ is a symmetric function that evaluates the difference between two parse trees $\mathbf{y}$ and $\mathbf{y}^*$. A higher value of $s_p(\hat{\mathbf{x}})$ means $\hat{\mathbf{x}}$ is more adversarial.

The primary criterion for selecting parsers B and C is their parsing accuracy. As we defined in Equation 1, the consensus prediction of parsers B and C is regarded as ground truth, no matter whether the prediction is actually right or wrong. Thus parsers B and C should have high accuracy and also different inductive biases so that they are unlikely to make the same mistake. In addition, B and C should not be too similar to parser A, because otherwise the first two terms in Equation 1 would become hard to optimize.

3.2 Evaluation Criteria for Sentence Quality

We consider two aspects of the sentence quality as follows:

- Fluency: Inspired by Holtzman et al. (2018); Xu et al. (2018); Pang et al. (2020), we use perplexity on GPT-2 (Radford et al., 2019), a large Transformer language model trained on massive texts, to evaluate the fluency of the generated sentences. We use the negative perplexity as a reward in the learning process.

$$s_f(\hat{\mathbf{x}}) = -PPL(\hat{\mathbf{x}})$$

- Meaning Preservation: Adversarial examples that differ too much from the original sentences are less effective in attacks because humans can easily identify them. We use BERTScore (Zhang et al., 2019b) as another reward in learning to evaluate the similarity between two sentences at the meaning level. We choose to use BERTScore because it correlates better with human judgments than traditional measures such as BLEU (Papineni et al., 2002).

$$s_m(\mathbf{x}, \hat{\mathbf{x}}) = BERTScore(\mathbf{x}, \hat{\mathbf{x}})$$

By maximizing these criteria, we hope to make the adversarial examples look more like human generated sentences and not differ too much from the original sentences in meaning.

3.3 Sentence Generator

We propose to use a seq2seq model (Wang et al., 2016) as the adversarial sentence generator, which has been widely used in machine translation, dialogue and many other areas. The seq2seq model specifies $P(\hat{\mathbf{x}}|\mathbf{x}; \Theta)$, the conditional probability of generating an adversarial sentence $\hat{\mathbf{x}}$ given an input sentence $\mathbf{x}$. We train the model by reinforcement learning guided by our aforementioned criteria. The objective function is the expected reward based on the sentences from the training corpus $\mathcal{X}$,

$$J(\Theta) = \sum_{\mathbf{x} \in \mathcal{X}} E_{\hat{\mathbf{x}} \sim P(\hat{\mathbf{x}}|\mathbf{x};\Theta)} s(\mathbf{x}, \hat{\mathbf{x}})$$

The reward $s(\mathbf{x}, \hat{\mathbf{x}})$ is composed of three parts.

$$s(\mathbf{x}, \hat{\mathbf{x}}) = \alpha s_p(\hat{\mathbf{x}}) + \beta s_f(\hat{\mathbf{x}}) + \gamma s_m(\mathbf{x}, \hat{\mathbf{x}}) \quad (2)$$

where α, β, γ are tunable hyper-parameters that control the balance between the three parts. We optimize the objective function with the REINFORCE algorithm (Williams, 1992).

To further encourage meaning preservation between $\mathbf{x}$ and $\hat{\mathbf{x}}$, we also pretrain the seq2seq model

as a denoising auto-encoder before reinforcement learning. Specifically, during pretraining we add noise to the hidden states of encoder and train the decoder to recover the input sentence. We employ the masking noise method that masks each word of the input sentence by a fixed probability and trains the denoising autoencoder to fill in these "blanks" (Vincent et al., 2008).

3.4 Defense against Adversarial Attack

Following Goodfellow et al. (2015), we use adversarial training to withstand attacks. More specifically, we enhance the victim model by injecting adversarial examples into the training data and retraining the model with the mixed data.

4 Experiments on Dependency Parsing

We first perform experiments on dependency parsing, a well-known structured prediction task.

4.1 Data

Our model does not need labeled data for training but we need a victim parser and two reference parsers in our experiments. We learn these parsers on an English dataset: Penn Treebank 3.0 (PTB, Marcus et al. (1994)). We also use the same data for training and evaluating our model.

4.2 Parser Selection

We choose the Deep Biaffine parser (Dozat and Manning (2017)), one of the state-of-the-art graph-based parsers, as the victim parser A. For the reference parsers, we choose two other well-known dependency parsers:

- **Parser B**: StackPTR from Ma et al. (2018)
- **Parser C**: BiST from Kiperwasser and Goldberg (2016)

The three parsers are trained with PTB. All the hyper-parameters of these parsers are the same as reported in their papers.

4.3 Evaluation Metrics

Our goal is to generate fluent sentences that are mispredicted by the victim model. Thus, we evaluate the adversarial examples produced by our model from 2 aspects: generation fluency and attacking efficiency (6 metrics).

Generation Fluency We use the perplexity on GPT-2 to evaluate the fluency of the generated sentences.

Attacking efficiency We evaluate the attacking success rates at the token level and sentence level. The token level attacking success rate is the percentage of words in the generated adversarial examples that are assigned the wrong head without considering the labels of the dependence type. It is also known as unlabeled attachment score (UAS). Sentence-level attacking success rate is the percentage of mispredicted sentences in the generated adversarial examples. Due the lack of golden parse trees of generated sentences, here we leverage the parses predicted by Parsers B and C as ground truth. The token level and sentence level each has three metrics: predictions of B as ground truth, predictions of C as ground truth, and consensus predictions of B and C as ground truth (discarding the sentences on which they disagree).

Human evaluation We conduct human evaluation of the fluency and attacking efficiency. All the volunteers have a background of linguistic study and are proficient in English. We further train the volunteers with the annotated English PTB treebank. From the adversarial examples generated by our method, we randomly sample 50 examples. During labeling, we ask two of them to label the sentences and the third skilled volunteers to double-check the evaluation results. For fluency, we ask them to rate the fluency of a sentence by an integer from 1 to 5. 5 indicates a sentence is fluent and has no grammatical errors. 1 indicates a sentence is full of grammatical errors and meaningless. For attacking efficiency, we ask them to manually annotate erroneous dependency edges and calculate the error rate in the same way as in automatic evaluation. The predictions of the Parsers B and C are given for reference.

4.4 Experimental Setup

We take the word-level approach in section 2.3 as our baseline, which uses a one-step update. Intuitively, this approach maintains the length of sentences and perturbs sentences by word-level replacement.

For our seq2seq generator, we use an attention-based three layers of BiLSTM with hidden vector dimension 1024. First, we pretrain the seq2seq generator for 3 epochs with unlabeled sentences from the PTB training set. The objective function for pretraining is negative conditional log likelihood. Then we train the seq2seq generator using reinforcement learning with hyper-parameter $\alpha = 1$,

	Generation Fluency (Perplexity ↓)	Token level Attacking Success Rate			Sentence level Attacking Success Rate		
		Parser B	*Parser C*	*Parsers B&C*	*Parser B*	*Parser C*	*Parsers B&C*
Origin	**156.02**	3.2	3.6	4.6	34.2	35.3	40.7
Baseline	217.02	3.7	10.1	11.5	37.0	64.1	67.4
Ours	174.16	**13.9**	**19.2**	**24.1**	**87.4**	**86.5**	**89.0**

Table 1: Experimental results on dependency parsing based on automatic evaluation. "Origin" shows the results of original sentences in the PTB test set. Lower perplexity is better.

	Generation Fluency ↑	Attacking Success Rate	
		Token	*Sentence*
Baseline	3.21	10.8	64
Ours	**3.84**	**18.3**	**72**

Table 2: Experimental results on dependency parsing based on human evaluation. Higher is better.

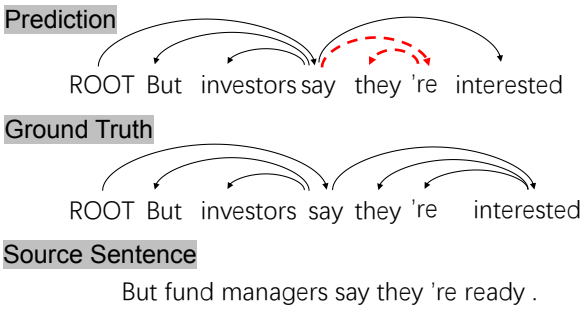

Figure 3: Case study of an adversarial example for dependency parsing task. The mispredicted dependencies of victim parser A are highlighted by dotted lines.

$\beta = 0.001$, $\gamma = 100$. Adam (Kingma and Ba, 2014) is used to optimize the parameters with the learning rate is $2e$-5. The minibatch size during reinforcement learning is 16. A detailed description of hyper-parameter settings can be found in Appendix A.

4.5 Experimental Results

Table 1 shows the automatic evaluation results. The attacking success rate improvement of our method over the baseline reflects the effectiveness of our reinforcement learning strategy. Particularly, our method improves the token level and sentence level attacking success rate 13.4% and 21.6% on Parsers B&C, respectively. It can also be seen that our proposed method maintains good fluency while making successful attacks. Human evaluation shown in Table 2 is consistent with automatic evaluation: our proposed method is significantly better than the baseline model at both generation fluency and attacking success rate. For better comparison, we ask volunteers to label the fluency score of the original sentences in PTB and obtain 4.64. We show an adversarial example in Figure 3.

		UAS
W/O Adv Train		95.42
Adv Train	*Baseline*	95.54
	BLLIP-BC	95.51
	BLLIP-ABC	95.46
	Ours	**95.63**

Table 3: Adversarial Training on different datasets for dependency parsing. *Adv Train*: adversarial training.

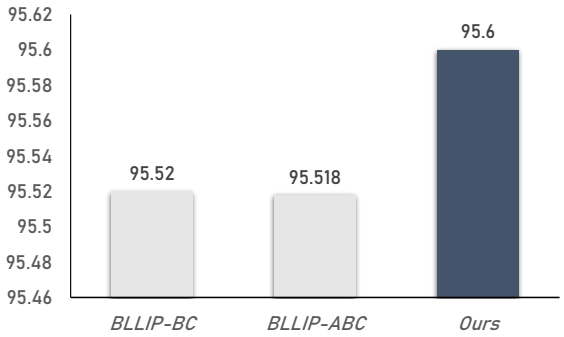

Figure 4: Average results of five time retrain using different datasets.

4.6 Adversarial Training

We then conduct experiments on adversarial training and summarize the results in Table 3. We add 2000 adversarial examples to the original training data and retrain the Biaffine parser[1]. We use the predicted parser Tree from Parsers B and C as the ground truth for these adversarial examples. If the parse trees from Parsers B and C are not the same, we drop the sentence. In addition to *W/O Adv Train* (result without adversarial training) and *Baseline* (retraining with adversarial examples produced by the word-level approach), we also experiment with the following two baseline methods of collecting 2000 additional training samples the BLLIP

[1]The candidate sentences are generated by the seq2seq generator using sentences in the training dataset as input. Then we drop the sentences that do **not** meet the criterion: reference parsers B and C predict the same parse trees that are different from the predictions of parser A (namely, the victim parser). Finally, we select the first 2000 sentences from the remaining 2044 sentences as the adversarial examples.

	Generation Fluency (Perplexity ↓)	Token level Attacking Success Rate			Sentence level Attacking Success Rate		
		Tagger B	*Tagger C*	*Tagger B&C*	*Tagger B*	*Tagger C*	*Tagger B&C*
Origin	156.02	1.9	2.1	3.2	30.6	35.5	45.7
Baseline	354.24	3.8	4.2	6.5	55.6	57.5	71.6
Ours	**142.59**	**9.2**	**7.3**	**14.5**	**78.1**	**73.3**	**89.0**

Table 4: Experimental results on POS tagging based on automatic evaluation. "Origin" shows the results of the original sentences. Lower perplexity is better.

	Generation Fluency ↑	Attacking Success Rate	
		Token	*Sentence*
Baseline	**3.98**	1.8	16
Ours	3.88	**8.1**	**52**

Table 5: Experimental results on POS tagging based on human evaluation. Higher means better.

dataset[2]:

- *BLLIP-BC*: Sampling sentences on which Parsers B and C predict the same parse trees.

- *BLLIP-ABC*: Sampling sentences on which Parsers B and C predict the same parse trees that are different from the predictions of Parser A.

We use the predicted parse trees from Parsers B and C as the ground truth for these two kinds of baselines. It can be seen that adversarial training, with adversarial examples leads to the largest performance gain over the "no adversarial training" baseline.

Although Table 3 shows that fine-tuning the victim parser A on our adversarial samples achieves better performance, the improvement is small. To investigate whether the improvement is significant or not, we retrain the parser A for five times with different random seeds. We also rerun the *BLLIP-BC* and *BLLIP-ABC* baselines (including the sampling step) for five times with different random seeds. The learning rate is $5e$-4. After training for 50 epochs, the average results are shown in Figure 4. It shows that our method outperforms the two baselines. We also perform Student's t-test:

- *BLLIP-BC* and *Ours*: t-value is -2.77 and p-value is 0.024.

- *BLLIP-ABC* and *Ours*: t-value is -3.39 and p-value is 0.010.

[2]Brown Laboratory for Linguistic Information Processing (BLLIP) 1987-89 WSJ Corpus Release 1. We choose the BLLIP corpus because it is collected from the same news article source as the WSJ corpus.

Both p-values are less than 0.05. That means the advantage of our method is statistically significant.

We also perform human evaluation on the retrained parser. The token level attacking success rate drops 1.3 points from 18.3 to 17.0, and the sentence level attacking success rate reduces from 72 to 70. We perform significance tests on the attacking success rate. The p-value is calculated by using the one-tailed sign test with bootstrap resampling on 50 samples following Chollampatt, Wang, and Ng (2019). We compare the attacking success rate with and without retraining. The p-values (5.42e-20 at the token level and 3.39e-21 at the sentence level) show that the improvement is significant.

5 Experiments on POS Tagging

5.1 Experimental Setup

In this section, we apply our method to the part-of-speech tagging task using the tagger from Ma and Hovy (2016) as the victim model. For the reference taggers, we choose two state-of-the-art taggers: Stanford POS tagger from Toutanova et al. (2003) and Senna tagger from Collobert et al. (2011). All the hyper-parameters of the three taggers are the same as reported in their papers. We conduct the experiments on the PTB dataset.

Similar to dependency parsing, the word level approach in section 2.3 is the baseline. For the adversarial example generator, we use the same structure and pretrain strategy as Section 4.4, except that the dimension of hidden state is set to 512. We train the sentence generator using reinforcement learning with hyper-parameter $\alpha = 1$, $\beta = 0.001$, $\gamma = 30$. Adam(Kingma and Ba, 2014) is used to optimize the parameters with learning rate $5e$-4. The minibatch size during reinforcement learning is 64. A detailed description of hyper-parameter settings can be found in Appendix B. We employ the same set of evaluation metrics as in section 4.3.

5.2 Experimental Results

We perform automatic evaluation over all the samples generated from the test dataset. As shown in

Prediction	NNP	NNS	VBD	VBG	.
Ground Truth	CC	NNS	VBD	VBG	.
Generated Sentence	But	stocks	went	falling	.
Original Sentence	Stocks kept falling .				
Prediction	NNP	NN	VBD	.	
Ground Truth	DT	NN	VBD	.	
Generated Sentence	The	market	went	.	
Original Sentence	Market crumbled .				

Figure 5: Case study of an adversarial example for POS tagging task. The mispredicted POS tags of victim tagger A are highlighted with underlines.

Table 4, attacking success rate and fluency of our proposed method are both above those of the baseline, which indicates the effectiveness of our proposed method. Particularly, our method improves the token level and sentence level attacking success rate 8.0% and 17.3%, respectively.

Similar to the dependency parsing task, Table 5 shows the result of human evaluation of 50 samples. According to human evaluation, the fluency of sentences generated by the two methods is similar, but the attacking success rate of our method is significantly higher than the baseline. Two example are shown in Figure 5.

We also conduct experiments on adversarial training with 1000 additional samples produced by our method. After retraining, the accuracy of Tagger A improves 0.13 point from 97.55 to 97.68 on PTB the test set. Similar to dependency parsing, we perform t-test to measure the statistical significance of the advantage of our method in POS tagging. The resulting p-value is 0.027.

6 Analysis

6.1 Selecting Reference Model

We mention in the Section 3.1 that the victim model and the two reference model should differ from each other as much as possible. In our previous experiments in Section 4 , we use three different types of parsers as the victim parser (Deep Biaffine) and reference parsers (StackPtr and BiST). Here we investigate the impact of making them similar. First, we make the two reference parsers similar to the victim parser, by training two Deep Biaffine parsers with different random seeds. We call this AllSame. Second, we make the two reference parsers similar to each other but different from the victim parser, by training two StackPtr parsers with different random seeds. We call this EvalSame.

Table 7 shows that AllSame tends to generate fluent sentences but the sentences are less adversarial. This can be explained by the fact that the similarity between the parsers make the first term of Equation 2 very small and the reward function is dominated by the two sentence quality terms. EvalSame can be seen to produce slightly higher token level attacking success rate but significantly lower generation fluency. Compared with AllSame and EvalSame, our standard method of using two different parsers as the reference models can reach a better attacking success rate, while keeping the sentences relatively fluent.

6.2 Applicability Analysis

We repeat our experiment of dependency parsing following the setup of Table 1 except for the choice of the victim parser and reference parsers. We use StackPTR as the victim model while the Deep Biaffine parser and BiST as Parser B and Parser C. Table 6 shows the automatic evaluation results. The results show similar trends to those in Table 1, suggesting that our approach is effective to different choices of the victim parser and reference parsers.

7 Related Work

Attack Design on Un-structured Prediction Model Following the success in the image processing area (Goodfellow et al., 2015), the idea of adding continuous perturbations to inputs has been applied to tasks in NLP (Sato et al., 2018; Gong et al., 2018). In order to solve the mapping problem from the modified word vector to the word, Papernot et al. (2016) built a special dictionary to select words to replace the original words. In addition to replacement manipulation, Samanta and Mehta (2017) introduced three modification strategies: removal and addition. Michel et al. (2019) leveraged atomic character-level operation. Some attack strategies to generate adversarial examples have been proposed in the sentence level setting. Zhao et al. (2018) searched adversarial examples in the continuous vector space and then used generative adversarial networks (Goodfellow et al., 2014) to map the fixed-length vectors to data instances. However, these attackers are only designed for classification tasks or generation tasks and can not be easily applied to structured prediction systems.

Attack Design on Structured Prediction Model There is also some prior work on attacking structured prediction models. Cisse et al. (2017) proposed to attack structured prediction models in the

	Generation Fluency (Perplexity ↓)	Token level Attacking Success Rate			Sentence level Attacking Success Rate		
		Parser B	*Parser C*	*Parsers B&C*	*Parser B*	*Parser C*	*Parsers B&C*
Baseline	377.36	4.5	15.9	17.5	40.7	74.5	74.90
Ours	**244.69**	**19.6**	**23.3**	**26.2**	**70.8**	**77.2**	**80.1**

Table 6: Experimental results on dependency parsing based on automatic evaluation with StackPTR as the victim model while the Deep Biaffine parser and BiST as Parser B and Parser C.

	Generation Fluency ↑	Attacking Success Rate	
		Token	*Sentence*
AllSame	**4.19**	11.4	64
EvalSame	3.54	13.6	62
Ours	3.84	**18.3**	**72**

Table 7: Results of human evaluation on different settings of the reference parsers. Higher is better.

image processing field, such as those for pose estimation and semantic segmentation. In a separate line of work, Zügner and Günnemann (2019) proposed to attack graph neural network for node classification.

8 Conclusion

Building an effective adversarial attacker for structured prediction models is challenging. The biggest challenge is the sensitivity of the output to small perturbations in the input in structured prediction. In this paper, we propose a novel framework to attack structured prediction models in NLP. Our framework consists of a structured-output evaluation criterion based on reference models and a seq2seq sentence generator. We propose to utilize reinforcement learning to train the sentence generator based on the evaluation criterion. Our attack experiments on dependency parsing and POS tagging show that our proposed framework can produce high-quality sentences that can effectively attack current state-of-the-art models. Our defense experiments show that adversarial training using the adversarial samples generated by our model can be used to improve the original model. We believe that our framework is general and can be applied to many other structured prediction tasks in NLP, such as neural machine translation, semantic parsing and so on.

Acknowledgments

Kewei Tu and Liwen Zhang are supported by the National Natural Science Foundation of China (61976139).

References

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2890–2896.

Yonatan Belinkov and Yonatan Bisk. 2018. Synthetic and natural noise both break neural machine translation. *International Conference on Learning Representations (ICLR)*.

Moustapha M Cisse, Yossi Adi, Natalia Neverova, and Joseph Keshet. 2017. Houdini: Fooling deep structured visual and speech recognition models with adversarial examples. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6977–6987. Curran Associates, Inc.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Timothy Dozat and Christopher D Manning. 2017. Deep biaffine attention for neural dependency parsing. *5th International Conference on Learning Representations (ICLR)*.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. Hotflip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36.

Zhitao Gong, Wenlu Wang, Bo Li, Dawn Song, and Wei-Shinn Ku. 2018. Adversarial texts with gradient methods. *arXiv preprint arXiv:1801.07175*.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron

Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. *International Conference on Learning Representations (ICLR)*.

Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. 2018. Learning to write with cooperative discriminators. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1638–1649.

Hossein Hosseini, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. 2017. Deceiving google's perspective api built for detecting toxic comments. *arXiv preprint arXiv:1702.08138*.

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*, pages 2021–2031.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1299–1304.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, pages 114–119. Association for Computational Linguistics.

Paul Michel, Xian Li, Graham Neubig, and Juan Pino. 2019. On evaluation of adversarial perturbations for sequence-to-sequence models. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3103–3114.

Takeru Miyato, Andrew M Dai, and Ian Goodfellow. 2017. Adversarial training methods for semi-supervised text classification. In *International Conference on Learning Representations (ICLR)*.

Bo Pang, Erik Nijkamp, Wenjuan Han, Linqi Zhou, Yixian Liu, and Kewei Tu. 2020. Towards holistic and automatic evaluation of open-domain dialogue generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3619–3629.

Nicolas Papernot, Patrick McDaniel, Ananthram Swami, and Richard Harang. 2016. Crafting adversarial input sequences for recurrent neural networks. In *MILCOM 2016-2016 IEEE Military Communications Conference*, pages 49–54. IEEE.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

Suranjana Samanta and Sameep Mehta. 2017. Towards crafting text adversarial samples. *arXiv preprint arXiv:1707.02812*.

Motoki Sato, Jun Suzuki, Hiroyuki Shindo, and Yuji Matsumoto. 2018. Interpretable adversarial perturbation in input embedding space for text. *arXiv preprint arXiv:1805.02917*.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for computational Linguistics.

Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2017. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103.

Dilin Wang, Chengyue Gong, and Qiang Liu. 2019. Improving neural language modeling via adversarial training. In *International Conference on Machine Learning*, pages 6555–6565.

Yequan Wang, Minlie Huang, Li Zhao, et al. 2016. Attention-based lstm for aspect-level sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 606–615.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Jingjing Xu, Xuancheng Ren, Junyang Lin, and Xu Sun. 2018. Dp-gan: diversity-promoting generative adversarial network for generating informative and diversified text. *arXiv preprint arXiv:1802.01345*.

Michihiro Yasunaga, Jungo Kasai, and Dragomir Radev. 2018. Robust multilingual part-of-speech tagging via adversarial training. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 976–986.

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

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019b. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Zhengli Zhao, Dheeru Dua, and Sameer Singh. 2018. Generating natural adversarial examples. *International Conference on Learning Representations (ICLR)*.

Daniel Zügner and Stephan Günnemann. 2019. Adversarial attacks on graph neural networks via meta learning. *International Conference on Learning Representations (ICLR)*.

A Dependency Parsing Experiment Details

During pretraining, the Deep Biaffine parser and the StackPtr parser is trained by Pytorch 0.4.1, the BiST parser is trained by Dynet.

Embedding	sskip
Embedding dim	100
POS Embedding dim	25
Word Embedding dropout	0.25
BiLSTM size	125
BiLSTM depth	2
MLP size	100
Batch size	32
Window	3
Optimizer	Adam
Learning rate	$1e$-1

Table 10: Hyper-parameters of pretraining the BiST parser.

Embedding	sskip
Embedding dim	100
Embedding dropout	0.33
BiLSTM size	512
BiLSTM depth	3
BiLSTM dropout	0.33
Arc MLP size	512
Arc MLP dropout	0.33
Label MLP size	128
Label MLP dropout	0.33
Batch size	32
Optimizer	Adam
Learning rate	$1e$-3

Table 8: Hyper-parameters of pretraining the Deep Biaffine parser. Here sskip is Structured SkipGram (Ling et al., 2015).

Embedding	sskip
Embedding dim	100
Embedding dropout	0.33
BiLSTM size	512
BiLSTM depth	3
BiLSTM dropout	0.33
Arc MLP size	512
Arc MLP dropout	0.33
Label MLP size	128
Label MLP dropout	0.33
Batch size	32
Optimizer	Adam
Learning rate	$1e$-3

Table 9: Hyper-parameters of pretraining the StackPtr parser.

Word Embedding	sskip
Word Embedding dim	100
BiLSTM depth	3
BiLSTM dim	1024
Hidden state dropout	0.5
Optimizer	Adam
Learning rate	$1e$-3
Epoch	3

Table 11: Hyper-parameters of pretraining our seq2seq sentence generator for dependency parsing.

α	1
β	0.001
γ	100
UNK weight	500
Optimizer	Adam
Learning rate	$2e$-5
Epoch	3

Table 12: Hyper-parameter of reinforcement training seq2seq sentence generator. UNK weight is a reward used to control the rate of UNK token. About 6 hours per epoch.

Retraining the Deep Biaffine parser We retrain the parser, all its hyper-parameter is same as the Table 8 but learning rate is $5e$-4.

B POS Tagging Experiment Details

the BiLSTM-CNN-CRF Tagger	
Embedding	sskip
Embedding dim	100
Embedding dropout	0.33
BiLSTM size	256
BiLSTM depth	1
Label MLP size	256
Label MLP dropout	0.5
Bigram	True
Batch size	16
Optimizer	Adam
Learning rate	$1e$-3

Table 13: Hyper-parameters during pretraining the BiLSTM-CNN-CRF Tagger.

Reference Tagger:

- Stanford POS tagger:
  http://nlp.stanford.edu/software/stanford-postagger-2015-04-20.zip
- Senna tagger:
  http://ronan.collobert.com/senna/senna-v3.0.tgz

During pretraining the seq2seq sentence generator, all hyper-parameters are same with Table 11 but BiLSTM dim is 512.

α	1
β	0.001
γ	30
UNK weight	0
Optimizer	Adam
Learning rate	$5e$-5
Epoch	3

Table 14: Hyper-parameter of reinforcement training seq2seq sentence generator. About 22 hours per epoch.

Retraining the BiLSTM-CNN-CRF Tagger We retrain the parser, all its hyper-parameter is same as the Table 13 but learning rate is $1e$-4.

C Hyper-Parameter Search

The criterion used to select all the hyper-parameters is the performance on the development data. We mainly tune the hyper-parameters of the text generator. For example, we choose the dimension of the hidden layer from 20 values in the range of 32 to 2048.