Grey-box Adversarial Attack And Defence For Sentiment Classification Supplementary Material

Ying Xu [*]
IBM Research
Australia

Xu Zhong [*]
IBM Research
Australia

Antonio Jimeno Yepes
IBM Research
Australia

Jey Han Lau
University of Melbourne
Australia

Abstract

We introduce a grey-box adversarial attack and defence framework for sentiment classification. We address the issues of differentiability, label preservation and input reconstruction for adversarial attack *and* defence in one unified framework. Our results show that once trained, the attacking model is capable of generating high-quality adversarial examples substantially faster (one order of magnitude less in time) than state-of-the-art attacking methods. These examples also preserve the original sentiment according to human evaluation. Additionally, our framework produces an improved classifier that is robust in defending against multiple adversarial attacking methods. Code is available at: https://github.com/ibm-aur-nlp/adv-def-text-dist

1 Reproducibility

1.1 Target model architectures

We constructed C-LSTM and C-CNN as target models for adversarial attack and C-BERT for transferability test.

For C-LSTM, we tuned a number of hyper-parameters including the learning rate $lr \in \{1e-2, 1e-3, 1e-4\}$, the batch size $bs \in \{32, 64, 128\}$, the number of hidden layers $l \in \{1, 2, 3\}$, the number of hidden units $u \in \{64, 128, 256\}$, and the number of attention units $u_a \in \{50, 150, 300\}$. After manual parameter tuning, we finally use the model with 2 hidden layers (256 units per layer) and one attention layer with 50 units that was trained with a learning rate of 1e-3 and batch size of 32.

For C-CNN, we tuned hyper-parameters including the learning rate $lr \in \{1e-2, 1e-3, 1e-4\}$, the batch size $bs \in \{32, 64, 128\}$, filter size for 1 convolutional layer $fs \in \{3, 5, 7\}$, filter sizes for 2 convolutional layers $fs \in \{[5, 3], [3, 3], [4, 2]\}$, the number of hidden units $u \in 64, 128, 256$, and the dropout keep probability $keep\_prob \in \{0.6, 0.8, 1.0\}$. After manual parameter tuning, we finally use the model with 2 convolutional layers of filter sizes [5, 3] and 256 units per hidden layer that is trained with a learning rate of 1e-3, batch size 32, and the dropout keep probability at 0.8.

For C-BERT, we used the default hyper-parameters for the BERT encoding layer, e.g. the batch size is set to 32 and the learning rate is set 1e-5. On top of the BERT encoder, we use a fully-connected layer of 1024 units whose output is projected to the final binary outputs.

For all three classification models, we keep updating the model until the ACC score on the dev set stops improving for 5000 steps.

1.2 Hyper-parameters for attacking methods

We report the attacking performance for three state-of-the-art attacking methods, TYC, HOTFLIP and TEXTFOOLER, and different variants of our grey-box attack method. For all attacking methods, we tune their respective hyper-parameters so as to achieving ACC scores on the target models that corresponds to the attacking thresholds at T1 (80-90%), T2 (70-80%), and T3 (60-70%). The three attacking thresholds are chose based on the idea that all or most of the compared attacking methods should be able to achieve and generate adversarial examples that are reasonably readable.

For the attacking method TYC, we tune the overshoot $\epsilon \in \{200, 2000, 20000\}$, the number of attacking iterations $i \in \{5, 10, 20\}$ [1], and the up limit of the number of changed words $lim \in \{7, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$. We first tune the first two hyper-parameters ϵ and i; and then tune lim by fixing $\epsilon = 2000$ and $i = 5$. After

---

[*] This work was completed during the employment of the authors in IBM Research Australia.

[1] The original implementation did not set a up limit for the attacking iteration, i.e. keep attacking until the output of the target model changed. However, we found that it could lead to unpractically long attacking time. Therefore, we set a limit on the attacking iterations.

manual tuning, the hyper-parameters of TYC at different attacking thresholds are shown in Table S1.

For the attacking method HotFlip, we tune only one hyper-parameter, e.g. the up limit of the number of changed words $lim \in \{1, 2, 3, 4, 5, 6, 7\}$. We manually tuned lim and its values at different attacking thresholds are shown in Table S1.

Similarly, for the attacking method TextFooler, we tune only one hyper-parameter, e.g. the up limit of the number of changed words $lim \in \{1, 2, 3, 4, 5, 6, 7\}$. This method has several other hyper-parameters, such as similar score threshold, similar score window, number of synonyms for each word, and important score threshold[2]. We use the default values for these hyper-parameters in most cases, and only try to relax the limitations when the attacking performance is not improved even all words (for example, all 50 words for the `yelp50` dataset) are allowed to be changed. In Table 2 of the main paper, when attacking the C-CNN for threshold T2 and T3, we also relaxed the other four hyper-parameters. However, no attacking improvement was observed. Hyper-parameters of TextFooler at different attacking thresholds are shown in Table S1.

Finally, for our attacking method, we tune hyper-parameters in two stages. During the pre-training of the auto-encoder, we try to build an auto-encoder that can reconstruct the input as faithfully as possible. Therefore we use the BLEU score on the development set as the tuning criteria. During the training of the attacking models, we initialise the auto-encoder with the weights obtained from the pre-training stage, and continue to train it according to the objective functions defined in Section 3.2. In this second stage, we are trying to find a balance between the attacking performance and reconstruction performance; and we use the ACC score of the target model on development set (which indicate the attacking performance) as the tuning criteria. We tune the attacking model in a total of 200,000 mini-batches with a batch size of 16, during which time we save the model when the ACC score of the target model on the development set drop into the ranges corresponding to T1, T2, and T3.

During the pre-training stage, we tune a number of hyper-parameters including the learning rate $lr \in \{1e-2, 1e-3, 1e-4\}$, the batch size $bs \in \{32, 64, 128\}$, the number of hidden layers $l \in \{1, 2, 3\}$, and the number of hidden units $u \in \{64, 128, 256\}$. After manual tuning, we finally use the auto-encoder with 2 hidden layers (512 units per layer) for both encoder and decoder and an attention layer with the same number of hidden units that is trained with a learning rate of 0.001 and a batch size of 32.

		C-LSTM	C-CNN
Methods	Thresholds	lim	lim
Tsai	T1	15	15
	T2	25	30
	T3	40	–
HotFlip	T1	3	1
	T2	4	5
	T3	6	7
TextFooler	T1	1	50
	T2	2	–
	T3	3	–

Table S1: The up limit of the number of changed words for TYC, HotFlip and TextFooler for achieving different attacking thresholds.

During the training of the attacking models, we tune a number of hyper-parameters including the learning rate $lr \in \{1e-3, 1e-4, 1e-5, 1e-6\}$, the two weighting parameters mitigating different objective functions (refer to Section 3.2) e.g. $\lambda_1 \in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ and $\lambda_2 \in \{0.2, 0.5, 0.7, 1, 0\}$, the Gumbel-softmax temperature $\tau \in \{0.1, 1.0, 10.0\}$. We found that the Gumbel-softmax temperature $\tau = 0.1$ provides the best performance, and therefore tune the other hyper-parameters while fixing τ at 0.1. For AE+LS+CF, we fixed the $\lambda_2 = 1.0$ [3] and tuned an additional hyper-parameter $\beta \in \{0.1, 0.5, 0.95\}$ that is introduced by label smoothing. We found that $\beta = 0.95$ provides the best balancing between BLEU and ACC score, and therefore fixed it at 0.95 while tuning the weighting parameter λ_1. Finally, for AE+LS+CF+CPY, we tune an additional hyper-parameter, the up limit of the number of changed words $lim \in \{3, 5, 7, 9, 11, 15, 19\}$, together with the weighting parameter λ_1. Detailed hyper-parameters of different variations of our models are shown in Table S2.

1.3 Hyper-parameters for defending methods

The defending models adversarially trained from HotFlip and TextFooler are based on the same set of hyper-parameters in adversarial attack at at-

---

[2] Please refer to the original paper of TextFooler and its Github implementation for definition of these hyper-parameters

[3] We found that the effect of changing λ_2 is not significant.

		C-LSTM				C-CNN			
Models	Thresholds	lr	λ_1	λ_2	lim	lr	λ_1	λ_2	lim
AE+BAL	T1	1e-4	0.2	0.7	–	1e-4	0.2	1.0	–
	T2	1e-4	0.2	0.7	–	1e-4	0.2	1.0	–
	T3	1e-4	0.15	1.0	–	1e-4	0.2	1.0	–
AE+LS+CF	T1	1e-5	0.8	1.0	–	1e-5	0.7	1.0	–
	T2	1e-5	0.8	1.0	–	1e-5	0.7	1.0	–
	T3	1e-5	0.8	1.0	–	1e-5	0.7	1.0	–
AE+LS+CF+CPY	T1	1e-5	0.8	1.0	9	1e-5	0.7	1.0	9
	T2	1e-5	0.8	1.0	9	1e-5	0.7	1.0	11
	T3	1e-5	0.8	1.0	11	1e-5	0.7	1.0	19

Table S2: The hyper-parameters of different variants of our method for achieving different attacking thresholds.

tacking threshold T2. Please refer to Table S1 for detailed hyper-parameter settings.

For AE+LS+CF-based defending model, besides the hyper-parameters required to be tuned in the attacking model, we tune an additional hyper-parameter: the alternative step $at\_step \in \{2, 5\}$, indicating for how many defending steps we perform one attacking step (please refer to Section 3.1 for definition of *attacking steps* and *defending steps*). The hyper-parameters for the defending models reported in this paper are set as follows: $lr = 1e-5$, $\lambda_1 = 0.4$, $\lambda_2 = 1.0$, and $at\_steps = 2$. Note that involving the defending component increased the number of trainable parameters to 76,655,822. For number of parameters for attacking models, please refer to Table S4.

2 Conditional generation

The main framework we introduced in the paper assumes no access to the ground-truth of the input example, i.e. it generates adversarial example purely based on the input example. However, most existing attacking methods assumes access to the ground-truth label of the input, such as TEXTFOOLER, so that the attack is only performed against examples that are correctly classified by the target model.

In this section, we explore the conditional generation in grey-box attack, which assumes access to the ground-truth of input examples at attacking time. Specifically, two seperate models are trained for positive-to-negative (PTN) and negative-to-positive (NTP) attack, respectively. The rationale is that, in conditional generation, given the ground truth of the input example, our attacking algorithm can choose between models for PTN and NTP accordingly, thereby performing more accurate attack.

As an additional experiment, we also explore the GAN-based auxiliary loss that were introduced in (Ren et al., 2020) for label preservation, which we denote as +GAN. Specifically, we involve an additional discriminative module $\mathcal{D}$ which consists of a discriminative model for each target class $t = \{1, ..., |C|\}$ and $|C|$ denotes the number of classes. The training phase is therefore divided into generative (attacking) steps, which are optimised to generate adversarial examples to fool the target model $\mathcal{C}$; and discriminative steps, which are optimised to discriminate between the original examples (labeled as '1') and generated examples (labeled as '0') of a specific class t. The discriminative losses for different classes are finally added together as the loss for discriminative steps, e.g. $L_{disc} = \sum_{t=1}^{|C|}(L^t_{disc})$, where $L^t_{disc} = L_{CE_{\mathcal{D}^t}}(logits_{\mathcal{D}^t([x,x^*])}, [1, 0])$. While for generative steps, an additional loss L_{aux} is added to L so as to fool the discriminators, where $L_{aux} = \sum_{t=1}^{|C|} L_{CE^t_{\mathcal{D}}}(logits_{\mathcal{D}^t([x,x^*])}, [1, 1])$. Intuitively, the GAN-based auxiliary losses are added to ensure that the generated adversarial examples, whose ground-truth label is t, are drawn from the same distribution of the input examples of class t.

We performed the same experiment with that reported in Table 1 of the main paper, in which we compared the performance of different variants of our grey-box attack framework on the development partition of `yelp50`. Table S3 includes the full results for both unified and conditional generation models for each of the variants, and Table S4 shows the number of trainable parameters for attacking variants.

Generally speaking, conditional generation did not help in terms of reconstruction, indicated by the

		ACC			BLEU			SENT		
		ALL	POS	NEG	SUC	POS	NEG	AGR	UKN	DAGR
(A) Unified	AE	66.0	99.8	28.4	55.3	71.7	58.7	–	–	–
	AE+BAL	75.6	72.3	78.8	65.9	73.9	70.9	0.12	0.80	0.08
	AE+LS	74.3	77.8	70.4	80.3	84.6	86.3	0.46	0.44	0.10
	AE+LS+GAN	74.6	77.1	71.9	75.9	80.1	81.3	0.42	0.54	0.04
	AE+LS+CF	76.6	66.5	86.7	79.9	82.5	85.0	0.64	0.28	0.08
	AE+LS+CF+GAN	79.6	69.4	89.9	74.5	79.4	81.9	0.62	0.34	0.04
	AE+LS+CF+CPY	77.4	70.9	83.8	**85.7**	90.6	90.2	**0.68**	0.30	0.02
(B) Conditional	AE$_{ptn}$+LS	–	74.4	–	81.9	87.6	–	0.54	0.36	0.10
	AE$_{ptn}$+LS+GAN	–	75.4	–	81.9	89.7	–	0.22	0.62	0.16
	AE$_{ptn}$+LS+CF	–	74.2	–	80.6	85.6	–	0.60	0.32	0.08
	AE$_{ptn}$+LS+CF+GAN	–	75.4	–	80.6	81.6	–	**0.80**	0.18	0.02
	AE$_{ptn}$+LS+CF+GAN+CPY	–	77.8	–	**85.9**	89.7	–	0.78	0.2	0.02
	AE$_{ntp}$+LS	–	–	74.4	82.7	–	88.6	0.60	0.20	0.20
	AE$_{ntp}$+LS+GAN	–	–	74.6	83.1	–	88.2	0.74	0.18	0.08
	AE$_{ntp}$+LS+CF	–	–	75.4	77.8	–	85.1	0.54	0.22	0.10
	AE$_{ntp}$+LS+CF+GAN	–	–	79.4	78.8	–	84.4	0.66	0.28	0.06
	AE$_{ntp}$+LS+CF+GAN+CPY	–	–	77.8	**83.5**	–	89.2	**0.74**	0.14	0.12

Table S3: Performance of adversarial examples generated by different auto-encoder variants on the `yelp50` development set.

Models	Number of Parameters
AE	61,791,232
AE+BAL	61,791,232
AE+LS	61,791,232
AE+LS+GAN	66,227,406
AE+LS+CF	72,219,648
AE+LS+CF+GAN	76,655,822
AE+LS+CF+CPY	72,219,648

Table S4: Number of parameters for different auto-encoder variants.

similar BLEU scores reported in the table; but it improved the sentiment agreement score from 0.68 to 0.80 and 0.74, respectively, for PTN and NTP attack. However, conditional generation requires access to the ground-truth of the input example and a 1-time query against the target model in attacking time. Therefore, we only reported the performance of unified generation in the main paper.

In terms of the GAN-based auxiliary loss, our results demonstrated that it brought benefits in terms of label preservation in conditional generation. However, it did not help with label preservation in unified generation, and decreased the BLEU score by 5 points. Therefore, we did not report it in the main paper.

References

Yankun Ren, Jianbin Lin, Siliang Tang, Jun Zhou, Shuang Yang, Yuan Qi, and Xiang Ren. 2020. Generating natural language adversarial examples on a large scale with generative models. *arXiv preprint arXiv:2003.10388*.