Attack on NLP

Todo List

1. B. Liang, H. Li, M. Su, P. Bian, X. Li, and W. Shi, “Deep text classification can be fooled,” arXiv preprint arXiv:1704.08006, 2017.
2. S. Samanta and S. Mehta, “Towards crafting text adversarial samples,” arXiv preprint arXiv:1707.02812, 2017.
3. Y. Belinkov and Y. Bisk, “Synthetic and natural noise both break neural machine translation,” arXiv preprint arXiv:1711.02173, 2017.
4. J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi, “Black-box generation of adversarial text sequences to evade deep learning classifiers,” arXiv preprint arXiv:1801.04354, 2018.
5. H. Hosseini, S. Kannan, B. Zhang, and R. Poovendran, “Deceiving google’s perspective api built for detecting toxic comments,” arXiv preprint arXiv:1702.08138, 2017.
6. Z. Gong, W. Wang, B. Li, D. Song, and W.-S. Ku, “Adversarial texts with gradient methods,” arXiv preprint arXiv:1801.07175, 2018.
7. Y. Zhang and B. Wallace, “A sensitivity analysis of (and practitioners guide to) convolutional neural networks for sentence classification,” in IJCNLP, vol. 1, 2017, pp. 253–263.
8. I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in ICLR, 2015, pp. 1–11.

TEXTBUGGER: Generating Adversarial Text Against Real-world Applications

Contribution

1. 提出了一种白盒/黑盒上的无目标的针对 NLP 分类任务的对抗攻击算法；
2. 作者采用的思想很简单：一是词的重要程度；二是 OOV 问题；三是相近词替换；
3. 文章的编写十分简洁明了，一眼看下去便能知其大概；

Notes

1. 一种同时支持**白盒/黑盒**的 **untargeted** 的**针对 NLP 分类任务**的对抗攻击算法；
2. 作者分析了已有的对抗攻击的一些缺点，也是这篇文章要解决的几个点：**计算复杂；白盒限制；人工干预；缺乏检验**；先看生成的对抗样本是什么样子的：

> ***Task:*** Sentiment Analysis. ***Classifier:*** CNN. ***Original label:*** 99.8% Negative. ***Adversarial label:*** 81.0% Positive.
>
> ***Text:*** I love these ~~awful~~ **awf ul** 80's summer camp movies. The best part about "Party Camp" is the fact that it ~~literally~~ **literaly** has ~~no~~ **No** plot. The ~~cliches~~ **clichs** here are limitless: the nerds vs. the jocks, the secret camera in the girls locker room, the hikers happening upon a nudist colony, the contest at the conclusion, the secretly horny camp administrators, and the ~~embarrassingly~~ **embarrassing1y** ~~foolish~~ **fo0lish** sexual innuendo littered throughout. This movie will make you laugh, but never intentionally. I repeat, never.
>
> ---
>
> ***Task:*** Sentiment Analysis. ***Classifier:*** Amazon AWS. ***Original label:*** 100% Negative. ***Adversarial label:*** 89% Positive.
>
> ***Text:*** I watched this movie recently mainly because I am a Huge fan of Jodie Foster's. I saw this movie was made right between her 2 Oscar award winning performances, so my expectations were fairly high. ~~Unfortunately~~ **Unf0rtunately**, I thought the movie was ~~terrible~~ **terrib1e** and I'm still left wondering how she was ever persuaded to make this movie. The script is really ~~weak~~ **wea k**.
>
> ---
>
> ***Task:*** Toxic Content Detection. ***Classifier:*** LSTM. ***Original label:*** 96.7% Toxic. ***Adversarial label:*** 83.5% Non-toxic.
>
> ***Text:*** hello how are you? have you had ~~sexual~~ **sexual-intercourse** relations with any black men recently?
>
> ---
>
> ***Task:*** Toxic Content Detection. ***Classifier:*** Perspective. ***Original label:*** 92% Toxic. ***Adversarial label:*** 78% Non-toxic.
>
> ***Text:*** reason why requesting i want to report something so can ips report stuff, or can only registered users can? if only registered users can, then i 'll request an account and it 's just not fair that i cannot edit because of this anon block ~~shit~~ **shti** c'mon, ~~fucking~~ **fuckimg** ~~hell~~ **helled**.

3. **白盒攻击**：

Algorithm 1 TEXTBUGGER under white-box settings

Input: legitimate document $\boldsymbol{x}$ and its ground truth label y, classifier $\mathcal{F}(\cdot)$, threshould ϵ
Output: adversarial document $\boldsymbol{x_{adv}}$

1: Inititialize: $\boldsymbol{x'} \leftarrow \boldsymbol{x}$
2: **for** word x_i in $\boldsymbol{x}$ **do**
3: Compute C_{x_i} according to Eq.2;
4: **end for**
5: $W_{ordered} \leftarrow Sort(x_1, x_2, \cdots, x_m)$ according to C_{x_i};
6: **for** x_i in $W_{ordered}$ **do**
7: $bug = SelectBug(x_i, \boldsymbol{x'}, y, \mathcal{F}(\cdot))$;
8: $\boldsymbol{x'} \leftarrow$ replace x_i with bug in $\boldsymbol{x'}$
9: **if** $S(\boldsymbol{x}, \boldsymbol{x'}) \leq \epsilon$ **then**
10: Return None.
11: **else if** $\mathcal{F}_l(\boldsymbol{x'}) \neq y$ **then**
12: Solution found. Return $\boldsymbol{x'}$.
13: **end if**
14: **end for**
15: **return** None

(1) 计算每个词的重要性 C_{x_i}（对应代码 2-4 行），即为分类结果求偏导，计算公式如下（应该是个向量？需要求均值或者求和？）：

$$C_{x_i} = J_{\mathcal{F}(i,y)} = \frac{\partial \mathcal{F}_y(\boldsymbol{x})}{\partial x_i}$$

(2) 根据重要性 C_{x_i} 对词从高到低进行排序（对应代码 5 行）

(3) 👍 选择 **字符级别的修改(character-level)** 和 **词级别的修改(word-level)**（思考：还有什么级别的攻击？词组级别的攻击，句子级别的攻击？）。**字符集别的修改依赖的思想是 OOV，词级别的修改依赖的是 Embedding 空间的语义相似性。**这里有一个有趣的现象，word2vec 这种 Embedding 方式会将词义完全相反的两个词 (**Better / Worst**) 分配在相近的空间中。这种修改方法在一定程度上依赖于人脑的推理能力，部分词的拼写发生了错误或者被删除了人同样能够识别整句句子的含义。五种修改方法：

- 插入空格，欺骗英文的单词分割；
- 删除一个字符，除了首尾字符；
- 交换两个字符，除了首位字符；

- 相似替换，如用 1 代替 i、用 m 替换 n；
- 用相近 (Embedding Top-k) 的单词替换目标单词；

算法如下，我们选择一种能**让目标分类概率值下降最多**的修改方法输出：

Algorithm 2 Bug Selection algorithm

```
function SelectBug(w, x, y, F(·))
    bugs = BugGenerator(w);
    for b_k in bugs do
        candidate(k) = replace w with b_k in x;
        score(k) = F_y(x) − F_y(candidate(k));
    end for
    bug_best = arg max_{b_k} score(k);
    return bug_best;
end function
```

> 思考一下：
>
> - 中文环境下的攻击场景会是怎样的?
> - 梯度信息能否再更多地利用一下，比如说：在选择相近词地时候，结合 Embedding + Gradient?

4. **黑盒攻击**：

Algorithm 3 TextBugger under black-box settings

Input: legitimate document $\boldsymbol{x}$ and its ground truth label y, classifier $\mathcal{F}(\cdot)$, threshould ϵ
Output: adversarial document $\boldsymbol{x_{adv}}$

```
1: Inititialize: x' ← x
2: for s_i in document x do
3:     C_sentence(i) = F_y(s_i);
4: end for
5: S_ordered ← Sort(sentences) according to C_sentence(i);
6: Delete sentences in S_ordered if F_l(s_i) ≠ y;
7: for s_i in S_ordered do
8:     for w_j in s_i do
9:         Compute C_{w_j} according to Eq.3;
10:    end for
11:    W_ordered ← Sort(words) according to C_{w_j};
12:    for w_j in W_ordered do
13:        bug = SelectBug(w_j, x', y, F(·));
14:        x' ← replace w_j with bug in x'
15:        if S(x, x') ≤ ε then
16:            Return None.
17:        else if F_l(x') ≠ y then
18:            Solution found. Return x'.
19:        end if
20:    end for
21: end for
22: return None
```

(1) 挑选重要的句子：划分段落中的句子，过滤掉目标标签不同的句子，并根据目标标签的概率对句子进行排序（对应代码 2-6 行）；（这边需要知道目标标签的概率，有些时候我们很可能是不知道的）

(2) 挑选重要的词：即消除一个词以后对目标标签概率的影响值 （对应代码 8-11 行）；

$$C_{w_j} = \mathcal{F}_y(w_1, w_2, \cdots, w_m) - \mathcal{F}_y(w_1, \cdots, w_{j-1}, w_{j+1}, \cdots, w_m)$$

(3) 和白盒攻击一样，修改上面的词；

5. Evaluation - Sentiment Analysis

(1) 数据集

- IMDB
- Rotten Tomatoes Movie Reviews (MR)

(2) 白盒攻击的模型:

- LR
- CNN
- LSTM

(3) 黑盒攻击的平台:

Google Cloud NLP	**IBM Waston Natural Language Understanding (IBM Watson)**
Microsoft Azure Text Analytics (Microsoft Azure)	**Amazon AWS Comprehend (Amazon AWS)**
Facebook fastText	**ParallelDots**
TheySay Sentiment	**Aylien Sentiment**
TextProcessing	**Mashape Sentiment**

(4) 对比工作:

- Random: 随机挑选 10% 的词进行修改 (白盒)
- **FGSM+Nearest Neighbor Search (NNS)** (白盒)
- **DeepFool+NNS** (白盒)
- **DeepWordBug** (黑盒)

(5) 评价指标: （👎 个人观点：这里罗列了 4 个指标，我觉得其中第二、三个可以不用计算，原因有两点：一是 Edit Distance和 Semantic Similarity这两个指标本身是比较直观的，可以用来分析语义的修改量和单词的修改多少；二是其他两个指标也只是用来罗列，作者没有对他们做出一些 interesting 的分析，也没有和其他人的工作进行比较，甚至还占用了大量的篇幅，放着没什么太大意义）

- Edit Distance
- Jaccard Similarity Coefficient: 集合的相似性

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|}=\frac{|A\cap B|}{|A|+|B|-|A\cap B|}$$

- Euclidean Distance: 使用词向量的距离进行度量

$$d(\boldsymbol{p},\boldsymbol{q})=\sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\cdots+(p_n-q_n)^2}$$

- Semantic Similarity: 度量句子语义的相似性，使用 sentence embedding，相关的模型使用的是 Universal Sentence Encoder

$$S(\boldsymbol{p},\boldsymbol{q})=\frac{\boldsymbol{p}\cdot\boldsymbol{q}}{||\boldsymbol{p}||\cdot||\boldsymbol{q}||}=\frac{\sum_{i=1}^{n}p_i\times q_i}{\sqrt{\sum_{i=1}^{n}(p_i)^2}\times\sqrt{\sum_{i=1}^{n}(q_i)^2}}$$

(6) 实现上的两个小细节:

- 对于不在词汇表中的词，直接随机化一个 embedding；
- 设置了 semantic similarity 的阈值为 0.8；

(7) 白盒攻击结果：

Model	Dataset	Accuracy	Random		FGSM+NNS [12]		DeepFool+NNS [12]		TEXTBUGGER	
			Success Rate	Perturbed Word	Success Rate	Perturbed Word	Success Rate	Perturbed Word	Success Rate	Perturbed Word
LR	MR	73.7%	2.1%	10%	32.4%	4.3%	35.2%	4.9%	**92.7%**	6.1%
	IMDB	82.1%	2.7%	10%	41.1%	8.7%	30.0%	5.8%	**95.2%**	4.9%
CNN	MR	78.1%	1.5%	10%	25.7%	7.5%	28.5%	5.4%	**85.1%**	9.8%
	IMDB	89.4%	1.3%	10%	36.2%	10.6%	23.9%	2.7%	**90.5%**	4.2%
LSTM	MR	80.1%	1.8%	10%	25.0%	6.6%	24.4%	11.3%	**80.2%**	10.2%
	IMDB	90.7%	0.8%	10%	31.5%	9.0%	26.3%	3.6%	**86.7%**	6.9%

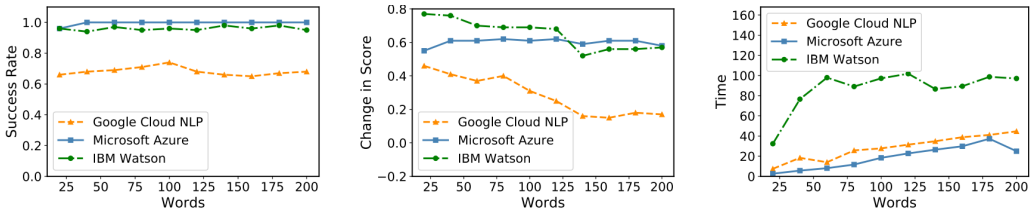
(8) 黑盒攻击在 IMDB 上的效果：

Targeted Model	Original Accuracy	DeepWordBug [11]			TEXTBUGGER		
		Success Rate	Time (s)	Perturbed Word	Success Rate	Time (s)	Perturbed Word
Google Cloud NLP	85.3%	43.6%	266.69	10%	**70.1%**	33.47	1.9%
IBM Waston	89.6%	34.5%	690.59	10%	**97.1%**	99.28	8.6%
Microsoft Azure	89.6%	56.3%	182.08	10%	**100.0%**	23.01	5.7%
Amazon AWS	75.3%	68.1%	43.98	10%	**100.0%**	4.61	1.2%
Facebook fastText	86.7%	67.0%	0.14	10%	**85.4%**	0.03	5.0%
ParallelDots	63.5%	79.6%	812.82	10%	**92.0%**	129.02	2.2%
TheySay	86.0%	9.5%	888.95	10%	**94.3%**	134.03	4.1%
Aylien Sentiment	70.0%	63.8%	674.21	10%	**90.0%**	44.96	1.4%
TextProcessing	81.7%	57.3%	303.04	10%	**97.2%**	59.42	8.9%
Mashape Sentiment	88.0%	31.1%	585.72	10%	**65.7%**	117.13	6.1%

(9) 黑盒攻击在 MR 上的效果:

Targeted Model	Original Accuracy	DeepWordBug [11]			TEXTBUGGER		
		Success Rate	Time (s)	Perturbed Word	Success Rate	Time (s)	Perturbed Word
Google Cloud NLP	76.7%	67.3%	34.64	10%	**86.9%**	13.85	3.8%
IBM Waston	84.0%	70.8%	150.45	10%	**98.8%**	43.59	4.6%
Microsoft Azure	67.5%	71.3%	43.98	10%	**96.8%**	12.46	4.2%
Amazon AWS	73.9%	69.1%	39.62	10%	**95.7%**	3.25	4.8%
Facebook fastText	89.5%	37.0%	0.02	10%	**65.5%**	0.01	3.9%
ParallelDots	54.5%	76.6%	150.89	10%	**91.7%**	70.56	4.2%
TheySay	72.3%	56.3%	69.61	10%	**90.2%**	30.12	3.1%
Aylien Sentiment	65.3%	65.2	83.63	10%	**94.1%**	13.71	3.5%
TextProcessing	77.6%	38.1%	59.44	10%	**87.0%**	12.36	5.7%
Mashape Sentiment	72.0%	73.6%	113.54	10%	**94.8%**	18.24	5.1%

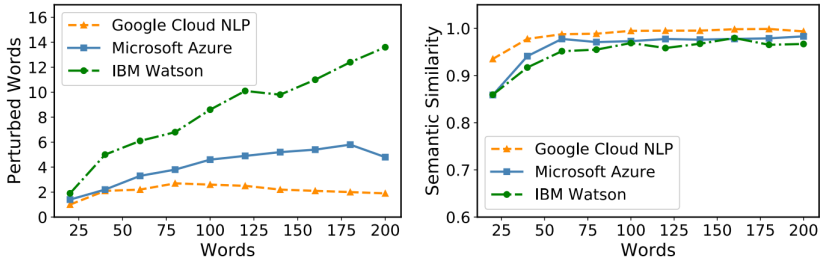
(10) 👍 句子长度对攻击的影响:



(a) Success Rate (b) Score (c) Time

Fig. 4. The impact of document length (i.e. number of words in a document) on attack's performance against three online platforms: Google Cloud NLP, IBM Watson and Microsoft Azure. The sub-figures are: (a) the success rate and document length, (b) the change of negative class's confidence value. For instance, the original text is classified as negative with 90% confidence, while the adversarial text is classified as positive with 80% confidence (20% negative), the score changes 0.9-0.2=0.7. (c) the document length and the average time of generating an adversarial text.

可以看到，句子长度并不会影响攻击的成功率，但会在一定程度上让其置信度有所下降（这并不会给攻击者带来太大的损失），以及让生成样本的时间增加。👎 **很可惜，这里我没有找到作者介绍 query 的次数，在黑盒攻击的情况下，query 次数是十分关键的一个指标，生成快并不能代表着 query 次数减少了。**

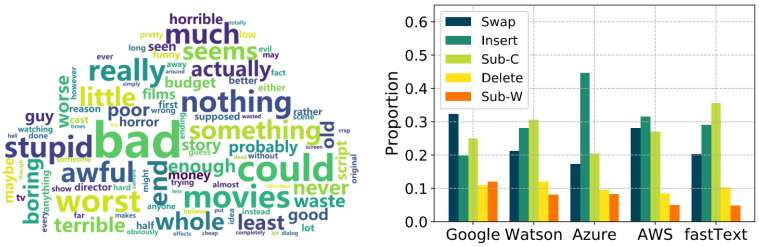


(a) Number of Perturbed Words (b) Semantic Similarity

Fig. 10. The impact of document length on the utility of generated adversarial texts in three online platforms: Google Cloud NLP, IBM Watson and Microsoft Azure. The subfigures are: (a) the number of perturbed words and document length, (b) the document length and the semantic similarity between generated adversarial texts and original texts.

可以看到，句子长度增加时，很正常的，每次修改词的量也需要增加，而句子的语义的变化则变得越来越小；

(11) 修改:



(a) Word Cloud (b) Bug Distribution

Fig. 11. (a) The word cloud is generated from IMDB dataset against the CNN model. (b) The bug distribution of the adversarial texts is generated from IMDB dataset against the online platforms.

比较有意思的一个点是，**作者列出了那些被判断为 negative 语句中的关键词。**另外，不同平台上算法选择的修改策略可能会有所不同。

6. Evaluation - Toxic Content Detection

(1) 数据集： Kaggle Toxic Comment Classification Competition dataset；👎 **这里有一点不同的是，这个数据集中本身是有 6 分类的，但是作者将其分成了 2 大类（Toxic and Not Toxic），这虽然看起来没什么不好的地方，但是体现出了作者的这种攻击其实是一种 untargeted 攻击。**

(2) 白盒模型：

- LR
- CNN
- LSTM

(3) 黑盒模型：

Google Perspective	**IBM Natural Language Classifier**
Facebook fastText	**ParallelDots AI**
Aylien Offensive Detector	

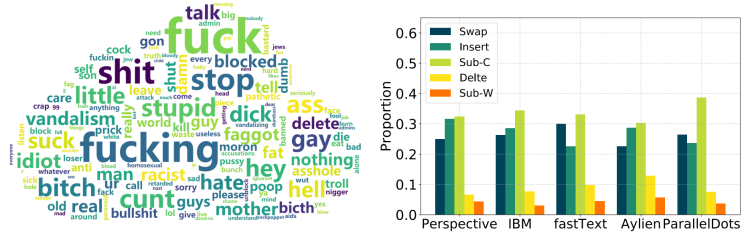
(4) 白盒攻击结果：

Targeted Model	Original Accuracy	Random		FGSM+NNS [12]		DeepFool+NNS [12]		TEXTBUGGER	
		Success Rate	Perturbed Word	Success Rate	Perturbed Word	Success Rate	Perturbed Word	Success Rate	Perturbed Word
LR	88.5%	1.4%	10%	33.9%	5.4%	29.7%	7.3%	**92.3%**	10.3%
CNN	93.5%	0.5%	10%	26.3%	6.2%	27.0%	9.9%	**82.5%**	10.8%
LSTM	90.7%	0.9%	10%	28.6%	8.8%	30.3%	10.3%	**94.8%**	9.5%

(5) 黑盒攻击结果：

Targeted Platform/Model	Original Accuracy	DeepWordBug [11]			TEXTBUGGER		
		Success Rate	Time (s)	Perturbed Word	Success Rate	Time (s)	Perturbed Word
Google Perspective	98.7%	33.5%	400.20	10%	**60.1%**	102.71	5.6%
IBM Classifier	85.3%	9.1%	75.36	10%	**61.8%**	21.53	7.0%
Facebook fastText	84.3%	31.8%	0.05	10%	**58.2%**	0.03	5.7%
ParallelDots	72.4%	79.3%	148.67	10%	**82.1%**	23.20	4.0%
Aylien Offensive Detector	74.5%	53.1%	229.35	10%	**68.4%**	37.06	32.0%

(6) 修改：



(a) Word Cloud (b) Bug Distribution

同样，作者列出了一些影响分类结果的关键词。

7. 👍 迁移能力：

Dataset	Model	White-box Models			Black-box APIs				
		LR	CNN	LSTM	IBM	Azure	Google	fastText	AWS
IMDB	LR	95.2%	20.3%	14.5%	14.5%	24.8%	15.1%	18.8%	19.0%
	CNN	28.9%	90.5%	21.2%	21.2%	31.4%	20.4%	25.3%	20.0%
	LSTM	28.8%	23.8%	86.6%	27.3%	26.7%	27.4%	23.1%	25.1%
MR	LR	92.7%	18.3%	28.7%	22.4%	39.5%	31.3%	19.8%	29.8%
	CNN	26.5%	82.1%	31.1%	25.3%	28.2%	21.0%	19.1%	20.5%
	LSTM	21.4%	24.6%	88.2%	21.9%	17.7%	22.5%	16.5%	18.7%

作者探讨了迁移攻击的有效性，可以看到，大致的迁移成功率约为 20% 左右。

8. 潜在的防御方法：
    - Spelling Check
    - Adversarial Training

> 我认为，文本分类问题是对抗攻击中是图像、语音、文本三大领域中最简单的一个问题，因为它并不会受到物理信道的影响，那么它就不用考虑物理鲁棒性这一大难题。另外，我认为在文本上应用字符、词级别的修改，本身是一种比较简单的方法（不像考虑词组搭配和句式变换等可能存在的攻击方法），甚至我们能够想象出这种攻击的一套规则，因此在防御的时候只需要考虑一些规则便可以（Spelling Check 和 Adversarial Training就像是经过了一次正则匹配一样）。

Links

- 论文链接：Li, Jinfeng, et al. "Textbugger: Generating adversarial text against real-world applications." *NDSS* (2019).
- 论文代码：CSE544T-Project-TextBugger
- ParallelDots 情感分析：https://www.paralleldots.com/
- GloVe 词向量：J. Pennington, R. Socher, and C. Manning, “Glove: Global vectors for word representation,” in EMNLP, 2014, pp. 1532–1543.
- Kaggle Toxic Comment Classification Competition dataset: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge