TextBugger: Generating Adversarial Text Against Real-world Applications

Jinfeng Li  Shouling Ji  Tianyu Du  Bo Li  Ting Wang

NDSS 2019
Machine Learning For Natural Language Processing

Information Retrieval

Machine Translation

Sentiment Analysis

Positive: 89%
Negative: 11%

Information Extraction

Machine Learning For Multiple Tasks

Question Answering
Machine Learning As A Service For NLP
Recent works have revealed the vulnerabilities of DNNs in image and speech domain

- The DNNs for image classification are vulnerable to adversarial images. [Goodfellow et al., ICLR’15]
- Automatic speech recognition systems can be broken down by adversarial audios in physical world. [Yuan et al., USENIX’18]

Do the adversarial examples also exist in text domain?

Are the MLaaS for NLP also vulnerable to adversarial examples?
Adversarial Text

What is the adversarial text?
Carefully generated by adding small perturbations to the legitimate text.

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 Unfortunatley, I thought the movie was terrible terrible horrible and I’m still left wondering how she was ever persuaded to make this movie. The script is really weak weak.

What is the challenge for generating adversarial texts?

- The discrete property of text makes it hard to optimize.
- Small perturbations in text are usually clearly perceptible.
- Replacement of a single word may drastically alter the semantics of the sentence.
Related Works For Generating Adversarial Texts

Gradient-based Methods

- Modifying an input text repetitively until it is misclassified. [Papernot et al., MILCOM’ 16]
- Changing one token to another by a gradient-based optimization method. [Ebrahimi et al., NAACL’ 18]
- Perturbing the important words determined by embedding gradient with hand-crafted synonyms. [Samanta et al., arXiv’17]

Out-of-vocabulary Words

- Breaking machine learning systems down by random character manipulations. [Belinkov et al., ICLR’ 18]
- Attacking black-box models by applying random character perturbations. [Gao et al. SPW’ 18]
- Changing the toxicity score of the texts by adding spaces or dots between characters. [Hosseini et al., arXiv’ 17]
Replace with Semantically/Syntactically Similar Words

- Only replacing words with semantically similar ones. [Alzantot et al., arXiv’ 18]
- Replacing tokens by random words of the same POS tag with a probability proportional to the embedding similarity. [Ribeiro et al., ACL’ 18]

Other Methods

- Attacking reading comprehension systems by adding distracting sentences to the input document. [Jia et al., EMNLP’ 17]
- Generating adversarial sequence by Generative Adversarial Networks (GANs). [Zhao et al., ICLR’ 18]
These works are limited in practice due to at least one of the following reasons:

- Limited to short texts
- Significantly affect the original meaning
- Need hand-crafted synonyms and typos
- Requires manual intervention to polish the added sentences
- Not computationally efficient
Framework For TextBugger

Text Classification
- Online Platform
- Offline Model

Attack Model
- Black-box Attack Model
- White-box Attack Model

Text Classifi
cation

Confidence value

Gradient information

Adversarial Text

Text

Word Embedding

Noise

feed back

vectors
Threat Model

White-box

- Have complete knowledge about the targeted model

Black-box

- Do not know the model architecture, parameters or training data
- Only capable of querying the targeted model with output as the prediction or confidence scores
Step 1: Finding Important Words

White-box attack

➢ Find important words by gradient information.

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

\[
J_{\mathcal{F}(x)} = \frac{\partial \mathcal{F}(x)}{\partial x} = \left[ \frac{\partial \mathcal{F}_j(x)}{\partial x_i} \right]_{i \in 1..N, j \in 1..K}
\]

Denotes:
• \(x\) is the input text, \(x_i\) is the \(i^{th}\) word in \(x\).
• \(\mathcal{F}_j(x)\) is the confidence value of the \(j^{th}\) class.
• \(C_{x_i}\) is the importance of word \(x_i\).
• \(N\) is the total number of words in \(x\).
• \(K\) is the total number of classes.
Step 1: Finding Important Words

Black-box attack

- Find important sentences
  \[ C_{\text{sentence}}(i) = F_y(s_i) \]
  \[ S_{\text{ordered}} \leftarrow \text{Sort}(s) \text{ according to } C_{\text{sentence}}(i) \]
  Delete sentences in \( S_{\text{ordered}} \) if \( F_1(s_i) \neq y \)

- Find important words for each sentence in \( S_{\text{ordered}} \)
  \[ C_{w_j} = F_y(w_1, w_2, \ldots, w_m) - F_y(w_1, \ldots, w_{j-1}, w_{j+1}, \ldots, w_m) \]

Denotes:
- \( s_i \) is the \( i^{th} \) sentence in the input text \( x \).
- \( F_y(s_i) \) is \( s_i \)'s confidence value of the predicted class \( y \).
- \( S_{\text{ordered}} \) is the important sentences set.
- \( C_{\text{sentence}}(i) \) is the importance of word \( s_i \), \( C_{w_j} \) is the importance of the \( j^{th} \) word in \( s_i \).

Sentence: *It is so laddish and juvenile, only teenage boys could possibly find it funny.*
Step 2: Bugs Generation

**Character-level perturbation**: out-of-vocabulary phenomenon

- **Insert**: Insert a space into the word.
- **Delete**: Delete a random character of the word.
- **Swap**: Swap random two adjacent letters in the word.
- **Substitute-C (Sub-C)**: Replace characters with visually similar characters or adjacent characters in the keyboard.

**Word-level perturbation**: nearest neighbor searching in the embedding space

- **Substitute-W (Sub-W)**: Replace a word with its top $k$ nearest neighbors in a context-aware word vector space.

<table>
<thead>
<tr>
<th>Original</th>
<th>Insert</th>
<th>Delete</th>
<th>Swap</th>
<th>Sub-C</th>
<th>Sub-W</th>
</tr>
</thead>
<tbody>
<tr>
<td>foolish</td>
<td>f oolish</td>
<td>folish</td>
<td>foolsh</td>
<td>fo0lish</td>
<td>silly</td>
</tr>
<tr>
<td>awfully</td>
<td>awfull y</td>
<td>awfully</td>
<td>awfully</td>
<td>awfully</td>
<td>terribly</td>
</tr>
<tr>
<td>cliches</td>
<td>clich es</td>
<td>cliches</td>
<td>clihes</td>
<td>cliches</td>
<td>cliche</td>
</tr>
</tbody>
</table>
Step 3: Replacing Important Word By Generated Bug

Optimal bug selection

- choose the optimal bug according to the change of the confidence value

\[
candidate(k) = \text{replace } w \text{ with } b_k \text{ in } x
\]
\[
score(k) = F_y(x) - F_y(candidate(k))
\]

Important word replacement

- Replace the important word by the selected optimal bug

- Repeat until “convergence”
  - the semantic similarity is below the threshold
  - the new text is misclassified by the classifier
Attack Evaluation
Case Study

Sentiment Analysis

Toxic Content Detection
Attack Evaluation: Sentiment Analysis

Dataset

- **IMDB**: 50,000 positive and negative movie reviews
- **Rotten Tomatoes Movie Reviews (MR)**: 5,331 positive and 5,331 negative snippets

Targeted Model

- **White-box models**: LR, CNN, LSTM
- **Real-world Online Platforms**:
  - fastText
  - ParallelDots
  - They Say
  - AYLIEN
  - mashape

Baseline Algorithms

- **White-box**: Random, FGSM+NNS (Nearest Neighbor Search), DeepFool+NNS
- **Black-box**: DeepWordBug
Attack Evaluation: Sentiment Analysis

Evaluation Metrics

- **Edit Distance**

- **Jaccard Similarity Coefficient**
  \[
  J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}
  \]

- **Euclidean Distance**
  \[
  d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}
  \]

- **Semantic Similarity**
  \[
  S(p, q) = \frac{p \cdot q}{||p|| \cdot ||q||} = \frac{\sum_{i=1}^{n} p_i \times q_i}{\sqrt{\sum_{i=1}^{n} (p_i)^2} \times \sqrt{\sum_{i=1}^{n} (q_i)^2}}
  \]
Important Words Selected By TextBugger
Successful Attack Examples

**Task:** Sentiment Analysis.  **Classifier:** CNN.  **Original label:** 99.8% Negative.  **Adversarial label:** 81.0% Positive.

**Text:** I love these awful awful 80's summer camp movies. The best part about "Party Camp" is the fact that it literally literally has no No plot. The clichés clichés 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 embarrassingly foolish foolish 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 Unfortunately UnfOrtunately, I thought the movie was terrible terrible and I'm still left wondering how she was ever persuaded to make this movie. The script is really weak weak.
Attack Performance: Effectiveness And Efficiency

White-box Attack

Remarks

- Choosing important words to modify is necessary.
- Effective: TextBugger has high attack success rate on all models and performs better than baselines.
- Evasive: TextBugger perturbs few words to fool the models.

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

**Remarks**

- **Effective**: TextBugger has higher attack success rate against all online platforms than DeepWordBug.
- **Evasive**: TextBugger only perturbs fewer words than DeepWordBug.
- **Efficient**: TextBugger spends less time than DeepWordBug.
Sentiment Score Distribution

- TextBugger greatly changes the confidence value of the classification results.
- IBM Watson is more sensitive to the adversarial texts generated by TextBugger.
Utility Analysis: White-box Attack

(a) IMDB

Remarks

- The generated adversarial texts preserve good word-level and vector-level utility.
Utility Analysis: Black-box Attack

Remarks

- TextBugger generates higher quality adversarial texts than DeepWordBug.
The Impact Of Document Length

The Impact of Document Length on Attack Performance

Remarks
- Length has little impact on the success rate, but may decrease the change of negative class’s confidence value.
- The time required for generating one adversarial text increases slightly as the length grows.
The Impact Of Document Length

The Impact of Document Length on The Utility of Generated Adversarial Texts.

Remarks

- Longer document length leads to more perturbed words.
- The increasing perturbed words do not decrease the semantic similarity of the adversarial texts.
Bug Distribution

Remarks

- Azure and AWS are sensitive to the insert bug
- Watson and fastText are sensitive to Sub-C
- Delete and Sub-W are used less than others
Further Analysis

Transferability

User Study
Transferability

Remarks

- Transferability also exists in adversarial texts among models and online platforms.
- Transferability can be used to attack online platforms even they have call limits.
User Study

(a) The distribution of all mistakes in the samples.
(b) The proportion of found bugs accounting for each kind of bug added in the samples.

Remarks

- Adversarial texts generated by TextBugger are hard to distinguish.
- The insert bug is human-perceptible.
- Sub-W is the most robust bug.
Vulnerability Report
Summary

We proposed TextBugger, a framework for generating adversarial texts effectively and efficiently

- **Effective**: It outperforms state-of-the-art attacks in terms of attack success rate under both white-box and black-box settings.
- **Evasive**: It preserves the utility of benign text.
- **Efficient**: It generates adversarial text with computational complexity sub-linear to the text length.

We evaluated TextBugger on 15 real-world online applications

- **Dataset**: IMDB, MR and Kaggle.
- **Application**: Includes sentiment analysis and toxic content detection.

We conducted a user study on our generated adversarial texts

- **Utility-preserving**: TextBugger has little impact on human understanding.

We further discuss two potential defense strategies to defend against such attacks