Black-box Generation of Adversarial Text Sequences to Evade Deep Learning Classifiers

By: Ji Gao, Jack Lanchantin, Mary Lou Soffa, Yanjun Qi

Presented by: Jennifer Fang [Week 02]

Department of Computer Science: University of Virginia

@ https://qdata.github.io/deep2Read/
Black-box Generation of Adversarial Text Sequences to Evade Deep Learning Classifiers

**Goal:** Create a new algorithm for black box testing to generate small text perturbations to cause deep-learning classifiers to misclassify a text input.

The new algorithm created is called DeepWordBug.
Black Box vs. White Box Testing

- **Black box testing**: testing as if you are a hacker i.e. no knowledge of the inside workings, don’t know details of learned models or feature representations of inputs
  - Can only manipulate input samples by testing and observing a classification model’s outputs
  - Usually it’s easy to query a model
  - But there’s no access to the inner structure of the models, which makes black box more applicable than white box

- **White box testing**: testing with full knowledge of the application

- Both black and white box testing cannot modify the model
Key Terms

- **Hyperparameter**: a parameter whose value is set before the experiment
  - Instead of deriving its value through training, this parameter has a set value
- **Adversarial samples**: inputs intentionally designed to cause the model to make a mistake
- **Transferability**: an important property where samples that are generated for one model can also be used to fool another DNN model
Goal of DeepWordBug

**Proven that:** Adding small modifications to text inputs can fool deep learning classifiers

**Question to answer:** Are deep learning classifiers robust?

Results have implications in text-based spam detection.

Two types of modifications to text input

\[ x' = x + \Delta x, \quad \|\Delta x\|_p < \epsilon, \quad x' \in X \]

\[ F(x) \neq F(x') \text{ or } F(x') = t \]

Targeted          Untargeted
DeepWordBug Example

Positive review

Original sample:
This film has a special place in my heart

Adversarial sample:
This film has a special place in my heart

Negative review
Differences of text vs. pictures

1. Text input $x$ is symbolic. Perturbation on $x$ is hard to define.
2. No metric has been defined to measure text difference. $L_p$-norms makes sense on continuous pixel values, but they don’t make sense on texts since they are discrete.
Basis of DeepWordBug

1. Determine the important tokens to change.
   - Use scoring functions to evaluate
2. Change those tokens
   - Create “imperceivable” changes which can evade a target deep learning classifier
Scoring Functions

1. **Replace-1 Score**
   - Replace one $x_i$ with $x_i'$
   - $R1S(x_i) = F(x_1, x_2, \ldots, x_{i-1}, x_i, \ldots, x_n) - F(x_1, x_2, \ldots, x_{i-1}, x_i', \ldots, x_n)$

2. **Temporal Head Score**
   - Difference between the model’s prediction score as it reads up to the $i^{th}$ token and as it reads up to the $(i-1)^{th}$ token
   - $THS(x_i) = F(x_1, x_2, \ldots, x_{i-1}, x_i) - F(x_i, x_2, \ldots, x_{i-1})$
Scoring Functions

3. Temporal Tail Score

- The complement of the THS
- Compares the difference between two trailing parts of a sentence, the one containing a certain token versus the one that does not.
- $\text{TTS}(x_i) = F(x_i, x_{i+1}, x_{i+2}, \ldots, x_n) - F(x_{i+1}, x_{i+2}, \ldots, x_n)$

4. Combination Score

- THS and TTS model from opposing sides, so the Combination Score combines the two
- $\text{CS}(x_i) = \text{THS}(x_i) + \lambda(\text{TTS}(x_i))$
- $\lambda$ is a hyperparameter
Text Transformations

1. **Swap**: Swap two adjacent letters in the word.
2. **Substitution**: Substitute a letter in the word with a random letter.
3. **Deletion**: Delete a random letter from the word.
4. **Insertion**: Insert a random letter in the word.

<table>
<thead>
<tr>
<th>Original</th>
<th>Swap</th>
<th>Substitution</th>
<th>Deletion</th>
<th>Insertion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team</td>
<td>Taem</td>
<td>Texm</td>
<td>Tem</td>
<td>Tezam</td>
</tr>
<tr>
<td>Artist</td>
<td>Artsit</td>
<td>Arxist</td>
<td>Artst</td>
<td>Articst</td>
</tr>
<tr>
<td>Computer</td>
<td>Comptuer</td>
<td>Computnr</td>
<td>Compter</td>
<td>Computer</td>
</tr>
</tbody>
</table>

*Table 1: Different transformer functions and their results.*
DeepWordBug Algorithm

Algorithm 1 DeepWordBug Algorithm

Input: Input sequence $x = x_1 x_2 \ldots x_n$, RNN classifier $F(\cdot)$, Scoring Function $S(\cdot)$, Transforming function $T(\cdot)$, maximum allowed perturbation on edit distance $\epsilon$.

1: for $i = 1..n$ do
2:     $scores[i] = S(x_i; x)$
3: end for
4: Sort $scores$ into an ordered index list: $L_1 .. L_n$ by descending score
5: $x' = x$
6: cost = 0, $j = 1$
7: while cost $< \epsilon$ do
8:     cost = cost + Transform($x'_{L_j}$)
9:     $j++$
10: end while
11: Return $x'$

Apply Scoring Function

Transform Text

Return $x'$
Experiment Setup

1. **Datasets**: 7 large scale datasets, including Enron Spam Dataset
2. **Target models**: 2 well trained models
   - **Word-LSTM**: a Bi-directional LSTM, which contains an LSTM in both directions (reading from first word to last and from last word to first) [used 4 different transformers]
   - **Char-CNN**: uses one-hot encoded characters as inputs to a 9-layer convolutional network [only used substitution transformer]
Comparison methods

1. Random (baseline): randomly selects tokens as targets
2. Gradient (baseline): uses full knowledge of the model to find most important tokens
3. DeepWordBug: use previously described white-box scoring functions to find most important tokens: Replace 1 Scoring, Temporal Head Score, Temporal Tail Score, Combined Score
Additional Parameter

$\epsilon$ = maximum allowed perturbation; maximum allowed edit distance (in characters)
### Word-LSTM Model

<table>
<thead>
<tr>
<th>Baselines</th>
<th>Original</th>
<th>Random</th>
<th>Gradient</th>
<th>Replace-1</th>
<th>Temporal Head</th>
<th>Temporal Tail</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG's News</td>
<td>90.5</td>
<td>89.3</td>
<td>1.33%</td>
<td>48.5</td>
<td>10.13%</td>
<td>36.1</td>
<td>60.0%</td>
</tr>
<tr>
<td>Amazon Review Full</td>
<td>62.0</td>
<td>61.1</td>
<td>1.48%</td>
<td>55.7</td>
<td>10.13%</td>
<td>18.6</td>
<td>70.05%</td>
</tr>
<tr>
<td>Amazon Review Polarity</td>
<td>95.5</td>
<td>93.9</td>
<td>1.59%</td>
<td>86.9</td>
<td>8.93%</td>
<td>40.7</td>
<td>57.36%</td>
</tr>
<tr>
<td>DBPedia</td>
<td>98.7</td>
<td>95.2</td>
<td>3.54%</td>
<td>74.4</td>
<td>24.61%</td>
<td>28.8</td>
<td>70.82%</td>
</tr>
<tr>
<td>Yahoo! Answers</td>
<td>73.4</td>
<td>65.7</td>
<td>10.54%</td>
<td>50.0</td>
<td>31.83%</td>
<td>27.9</td>
<td>61.93%</td>
</tr>
<tr>
<td>Yelp Review Full</td>
<td>64.7</td>
<td>60.9</td>
<td>5.86%</td>
<td>53.2</td>
<td>17.76%</td>
<td>23.4</td>
<td>63.83%</td>
</tr>
<tr>
<td>Yelp Review Polarity</td>
<td>95.9</td>
<td>95.4</td>
<td>0.55%</td>
<td>88.4</td>
<td>7.85%</td>
<td>37.8</td>
<td>60.63%</td>
</tr>
<tr>
<td>Enron Spam Email</td>
<td>96.4</td>
<td>87.8</td>
<td>29.69%</td>
<td>76.7</td>
<td>29.47%</td>
<td>39.1</td>
<td>59.48%</td>
</tr>
<tr>
<td>Mean</td>
<td>6.82%</td>
<td>16.46%</td>
<td>63.02%</td>
<td>44.40%</td>
<td>68.05%</td>
<td>64.38%</td>
<td>61.28%</td>
</tr>
<tr>
<td>Median</td>
<td>2.57%</td>
<td>13.95%</td>
<td>4.94%</td>
<td>9.52%</td>
<td>6.77%</td>
<td>9.56%</td>
<td>9.56%</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>9.81%</td>
<td>8.71%</td>
<td>4.94%</td>
<td>9.52%</td>
<td>6.77%</td>
<td>9.56%</td>
<td>9.56%</td>
</tr>
</tbody>
</table>

### Char-CNN Model

<table>
<thead>
<tr>
<th>Baselines</th>
<th>Original</th>
<th>Random</th>
<th>Gradient</th>
<th>Replace-1</th>
<th>Temporal Head</th>
<th>Temporal Tail</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG's News</td>
<td>90.0</td>
<td>82.4</td>
<td>8.36%</td>
<td>62.3</td>
<td>30.74%</td>
<td>30.8</td>
<td>65.80%</td>
</tr>
<tr>
<td>Amazon Review Full</td>
<td>61.1</td>
<td>51.0</td>
<td>16.53%</td>
<td>47.0</td>
<td>23.04%</td>
<td>25.6</td>
<td>58.17%</td>
</tr>
<tr>
<td>Amazon Review Polarity</td>
<td>95.2</td>
<td>93.4</td>
<td>1.91%</td>
<td>84.3</td>
<td>11.41%</td>
<td>46.4</td>
<td>51.27%</td>
</tr>
<tr>
<td>DBPedia</td>
<td>98.4</td>
<td>95.8</td>
<td>2.58%</td>
<td>92.9</td>
<td>5.60%</td>
<td>74.9</td>
<td>23.91%</td>
</tr>
<tr>
<td>Yahoo! Answers</td>
<td>71.0</td>
<td>52.2</td>
<td>26.45%</td>
<td>43.5</td>
<td>38.76%</td>
<td>30.0</td>
<td>57.72%</td>
</tr>
<tr>
<td>Yelp Review Full</td>
<td>63.5</td>
<td>52.6</td>
<td>17.05%</td>
<td>45.7</td>
<td>28.06%</td>
<td>27.6</td>
<td>56.56%</td>
</tr>
<tr>
<td>Yelp Review Polarity</td>
<td>95.3</td>
<td>91.2</td>
<td>4.31%</td>
<td>84.8</td>
<td>11.03%</td>
<td>42.8</td>
<td>55.05%</td>
</tr>
<tr>
<td>Enron Spam Email</td>
<td>95.6</td>
<td>85.5</td>
<td>10.56%</td>
<td>69.0</td>
<td>27.84%</td>
<td>76.4</td>
<td>20.13%</td>
</tr>
<tr>
<td>Mean</td>
<td>10.97%</td>
<td>22.06%</td>
<td>48.58%</td>
<td>11.05%</td>
<td>31.79%</td>
<td>30.35%</td>
<td>10.10%</td>
</tr>
<tr>
<td>Median</td>
<td>9.46%</td>
<td>25.44%</td>
<td>55.80%</td>
<td>10.10%</td>
<td>30.19%</td>
<td>29.11%</td>
<td>12.93%</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>8.54%</td>
<td>11.53%</td>
<td>16.91%</td>
<td>7.10%</td>
<td>14.56%</td>
<td>12.93%</td>
<td>12.93%</td>
</tr>
</tbody>
</table>

Table 5: Effectiveness of WordBug on 8 Datasets using the Word-LSTM and Char-CNN model. Acc is the accuracy of the method and Decrease is the percent decrease of the accuracy by using the specified attacking method over the original accuracy. Word-LSTM uses Substitution transformer. All results are under maximum edit distance difference 30 (ε = 30).
Decrease in Performance

![Bar chart showing relative performance decrease for various methods under DeepWordBug]

![Bar chart showing relative performance decrease for various methods under DeepWordBug]
Results

1. **Accuracy**: reduced 68% performance of the Word-LSTM model and 48% performance of the Char-CNN model

2. **Influence of the Scoring Function**: very important
   a. DeepWordBug’s scoring is better than the gradient
   b. Without scoring (random case), adversarial performance is low

3. **Transferable?** Yes, even for models with different word embedding

4. **Influence of Transformation Function**: wasn’t much difference within the functions; having a good scoring function is more important

5. **Influence of Dictionary size**: low; works for all dictionary sizes

6. **Probability of classifications**: 94.6% of classifications were classified with > 0.9 confidence for Word-LSTM model on the Enron Spam Dataset (# classes = 2), $\epsilon = 30$
Transferability and Confidence

Figure 12: How strong the machine learning model will believe the wrong answer lead by the adversarial sample, the x-axis are the confidence range and the y-axis are the probability distribution. The result is generated using Word-LSTM model on the Enron Spam Dataset (Number of classes = 2), with edit distance maximum $\epsilon = 30$. 
Applications

Adversarial training: with training on DeepWordBug, adversarial accuracy improves from 12% to 62%

Autocorrection: reduces the performance of adversarial samples; can combat this with stronger transformation functions such as substitution-2 and deletion-2
Why DeepWordBug Works

- When changes are made to a word, the word becomes unknown, which map to the unknown embedding vector
- Small changes can thus make a big impact
- Adversarial samples are probably decipherable to humans, but not to models
- Area for ML to catch up with humans
Advantages:

1. **Black-box**: DeepWordBug generates adversarial samples in a pure black-box manner.

2. **Performance**: DeepWordBug results in a 68% decrease on average from the original classification accuracy for a word-level LSTM model and 48% decrease on average for a character-level CNN model.
   - Results are transferable and are not reliant on dictionary size or transformation technique used

3. **Applications**: Adversarial training is successful; by using DeepWordBug generated samples, model accuracy on generated adversarial samples increases from 12% to 62%