A Appendix on Results

A.1 Search Space Choices

Here, we provide more detail about our search space choices.

A.1.1 Transformations

We consider three different ways of generating substitute words:

1. Counter-fitted GLOVE word embedding (Mrksic et al., 2016): For a given word, we take its top N nearest neighbors in the embedding space as its synonyms. 7
2. HowNet (Dong et al., 2010): HowNet is a knowledge base of sememes in both Chinese and English.
3. WordNet (Miller, 1995): WordNet is a lexical database that contains knowledge about and relationships between English words, including synonyms.

A.1.2 Constraints

To preserve grammaticality, we require that the two words being swapped have the same part-of-speech (POS). This is determined by a part-of-speech tagger provided by Flair (Akbik et al., 2018), an open-source NLP library.

To preserve semantics, we consider three different constraints:

1. Minimum cosine similarity of word embeddings: For the word embedding transformation, we require the cosine similarity of the embeddings of the two words meet a minimum threshold.
2. Minimum BERTScore (Zhang* et al., 2020): We require that the F1 BERTScore between x and x’ meet some minimum threshold value.
3. Universal Sentence Encoder (Cer et al., 2018): We require that the angular similarity between the sentence embeddings of x and x’ meet some minimum threshold.

For word embedding similarity, BERTScore, and USE similarity, we need to set the minimum threshold value. We set all three values to be 0.9 based on the observation reported by Morris et al. (2020b) that high threshold values encourages strong semantic similarity. We do not apply word embedding similarity constraint for HowNet and WordNet transformations because it is not guaranteed that we can map the substitute words generated from the two sources to a word embedding space. We can also assume that the substitute words are semantically similar to the original words since they originate from a curated knowledge base.

Lastly, for all attacks carried out, we do not allow perturbing a word that has already been perturbed and we do not perturbed pre-defined stop words.

A.1.3 Datasets

We compare search algorithms on three datasets: the Movie Review and Yelp Polarity sentiment classification datasets and the SNLI entailment dataset. Figure 3 shows a histogram of the number of words in inputs from each dataset. We can see that inputs from Yelp are generally much longer than inputs from MR or SNLI.

A.2 Pseudocode for Search Algorithms

Before presenting the pseudocode of each search algorithm, we define a subroutine called perturb that takes text x and index i to produce set of perturbation x’ that satisfies the constraints. More specifically, perturb is defined as following:

\[
\text{perturb}(x, i) = \{ T(x, i) \mid C_j(T(x, i)) \forall j \in \{1, ..., m\} \}
\]

where \( T(x, i) \) represents the transformation method that swaps the \( i^{th} \) word \( x_i \) with its synonym to produce perturbed text \( x’ \). \( C_1, ..., C_m \) are constraints represented as Boolean functions. \( C_i(x) = True \) means that text \( x \) satisfies constraint \( C_i \).

Also, score(x) is the heuristic scoring function that was defined in the section 3.2.

Greedy search with word importance ranking requires subroutine for determining the importance of each word in text x. We leave the details of the importance functions to be found in individual papers that have proposed them, including Gao et al. (2018), Jin et al. (2019), Ren et al. (2019).

In genetic algorithm, each population member represents a distinct text produced via perturb and crossover operations. Genetic algorithm has a subroutine called sample that takes in population member p and randomly samples a word to transform with probabilities proportional to the number of synonyms a word has. Also, we modified the crossover subroutine proposed by Alzantot et al. (2018) to check if child produced by crossover...
Algorithm 1: Beam Search with beam width $b$

**Input**: Original text $x = (x_1, x_2, \ldots, x_n)$

**Output**: Adversarial text $x_{adv}$ if found

```plaintext
best ← \{x\}

while best == ∅ do

    $X_{cand}$ ← ∅

    for all $x_h \in$ best do

        for all $i \in \{1, \ldots, n\}$ do

            $X_{cand}$ ← $X_{cand}$ ∪ perturb($x_h, i$)

        end for

    end for

    if $X_{cand} \neq ∅$ then

        $x^*$ ← arg max$_{x' \in X_{cand}}$ score($x'$)

        if $x^*$ fools the model then

            return $x^*$ as $x_{adv}$

        else

            best ← \{top $b$ elements of $X_{cand}\}

        end if

    end if

end while
```

Algorithm 2: Greedy Search

**Input**: Original text $x = (x_1, x_2, \ldots, x_n)$

**Output**: Adversarial text $x_{adv}$ if found

```plaintext
$x^* \leftarrow x$

while $x_{adv}$ not found do

    $X_{cand}$ ← ∅

    for all $i \in \{1, \ldots, n\}$ do

        $X_{cand}$ ← $X_{cand}$ ∪ perturb($x^*, i$)

    end for

    if $X_{cand} \neq ∅$ then

        $x^*$ ← arg max$_{x' \in X_{cand}}$ score($x'$)

        if $x^*$ fools the model then

            return $x^*$ as $x_{adv}$

        end if

    else

        End search

    end if

end while
```

Algorithm 3: Greedy Search with Word Importance Ranking

**Input**: Original text $x = (x_1, x_2, \ldots, x_n)$

**Output**: Adversarial text $x_{adv}$ if found

```plaintext
$R \leftarrow$ ranking $r_1, \ldots, r_n$ of words $x_1, \ldots, x_n$ by their importance

$x^* \leftarrow x$

for $i = r_1, r_2, \ldots, r_n$ in $R$ do

    $X_{cand}$ ← perturb($x^*, i$)

end for

if $X_{cand} \neq ∅$ then

    $x^*$ ← arg max$_{x' \in X_{cand}}$ score($x'$)

    if $x^*$ fools the model then

        return $x^*$ as $x_{adv}$

    end if

else

    End search

end if
```

Figure 3: Histogram of words per dataset. Yelp inputs are generally much longer than inputs from MR or SNLI.
operation passes constraints. If the child fails any of the constraints, we retry the crossover for at max 20 times. If that also fails to produce a child that passes constraints, we randomly choose one its parents to be the child with equal probability.

Algorithm 4 Genetic Algorithm (with population size $K$ and generation $G$)

Input: Original text $x = (x_1, x_2, ..., x_n)$
Output: Adversarial text $x_{adv}$ if found

for $k = 1, \ldots, K$ do
    $i \leftarrow \text{sample}(x)$
    $X_{\text{cand}} \leftarrow \text{perturb}(x, i)$
    $P^0_k \leftarrow \arg \max_{x' \in X_{\text{cand}}} \text{score}(x')$
end for

for $g = 1, \ldots, G$ generations do
    for $k = 1, \ldots, K$ do
        $S^g_k \leftarrow \text{score}(P^g_k)$
    end for
    $x^* \leftarrow P^{g-1}_{\arg \max_i S^g_i}$
    if $x^*$ fools the model then
        return $x^*$ as $x_{adv}$
    else
        $P^g_1 \leftarrow x^*$
    end if
    $p = \text{Normalize}(S^g)$
    for $k = 2, \ldots, K$ do
        $\text{par}_1 \sim P^{g-1}$ with prob $p$
        $\text{par}_2 \sim P^{g-1}$ with prob $p$
        $\text{child} \leftarrow \text{crossover}(\text{par}_1, \text{par}_2)$
    end for
    $i \leftarrow \text{sample}($child$)$
    $X_{\text{cand}} \leftarrow \text{perturb}($child$, i)$
    $P^g_k \leftarrow \arg \max_{x' \in X_{\text{cand}}} \text{score}(x')$
end for

Lastly, we leave out the pseudocode for PSO due to its complexity. More detail can be found in Zang et al. (2020).

A.3 Analysis of Attacks against LSTM Models

Figures 4 shows how the number of words in the input affects runtime for each algorithm against LSTM models. Figure 5 shows the attack success rate of each search algorithm as the maximum number of queries permitted to perturb a single sample varies from 0 to 20,000 for Yelp dataset and 0 to 3000 for MR and SNLI.

A.4 Evaluation of Adversarial Examples

Table 4 shows the average percentage of words perturbed, average Universal Sentence Encoder similarity score, and average percent change in perplexity for all experiments.

B Future Work

Submodularity of transformer models. As mentioned in Section 5, our findings indicate that the NLP attack problem may be approximately submodular when dealing with transformer models. In the image space, attacks designed to take advantage of submodularity have achieved high query efficiency (Moon et al., 2019). With the exception of (Lei et al., 2019), attacks in NLP are yet to take advantage of this submodular property.

Transformations beyond word-level. Most proposed adversarial attacks in NLP focus on making substitutions at the word level or the character level. A few works have considered replacing phrases (Ribeiro et al., 2018) as well as paraphrasing full sentences (Lei et al., 2019; Iyyer et al., 2018). However, neither of these scenarios has been studied extensively. Future work in NLP adversarial examples would benefit from further exploration of phrase and sentence-level transformations.

Motivations for generating NLP adversarial examples. One purpose of generating adversarial examples for NLP systems is to improve the systems. Much work has focused on improvements in intrinsic evaluation metrics like achieving higher attack success rate via an improved search method. To advance the field, future researchers might focus more on using adversarial examples in NLP to build better NLP systems.
Figure 4: Number of queries vs. length of input text.
Figure 5: Attack success rate by query budget for each search algorithm and dataset.
<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset</th>
<th>Search Method</th>
<th>GLOVE Word Embedding</th>
<th>HowNet</th>
<th>WordNet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Avg P.W. %</td>
<td>Avg USE Sim</td>
<td>Avg P.W. %</td>
</tr>
<tr>
<td>Yelp</td>
<td>Greedy (b=1)</td>
<td>3.41</td>
<td>0.948</td>
<td>21.5</td>
<td>2.52</td>
</tr>
<tr>
<td></td>
<td>Beam Search (b=8)</td>
<td>3.26</td>
<td>0.949</td>
<td>20.7</td>
<td>2.45</td>
</tr>
<tr>
<td></td>
<td>Beam Search (b=8)</td>
<td>3.20</td>
<td>0.950</td>
<td>20.1</td>
<td>2.42</td>
</tr>
<tr>
<td></td>
<td>WIR (Beam)</td>
<td>6.48</td>
<td>0.930</td>
<td>43.5</td>
<td>4.73</td>
</tr>
<tr>
<td></td>
<td>WIR (DELL)</td>
<td>6.85</td>
<td>0.928</td>
<td>47.2</td>
<td>5.10</td>
</tr>
<tr>
<td></td>
<td>WIR (PM2)</td>
<td>4.36</td>
<td>0.942</td>
<td>27.3</td>
<td>3.11</td>
</tr>
<tr>
<td></td>
<td>WIR (Gradient)</td>
<td>6.16</td>
<td>0.933</td>
<td>37.8</td>
<td>5.58</td>
</tr>
<tr>
<td></td>
<td>WIR (GA0)</td>
<td>8.18</td>
<td>0.920</td>
<td>59.1</td>
<td>7.46</td>
</tr>
<tr>
<td></td>
<td>Genetic Algorithm</td>
<td>5.06</td>
<td>0.936</td>
<td>33.9</td>
<td>4.21</td>
</tr>
<tr>
<td></td>
<td>PSO</td>
<td>6.61</td>
<td>0.929</td>
<td>47.3</td>
<td>6.08</td>
</tr>
<tr>
<td>BERT</td>
<td>Greedy (b=1)</td>
<td>7.45</td>
<td>0.930</td>
<td>31.1</td>
<td>7.14</td>
</tr>
<tr>
<td></td>
<td>Beam Search (b=8)</td>
<td>7.22</td>
<td>0.901</td>
<td>31.4</td>
<td>6.10</td>
</tr>
<tr>
<td></td>
<td>Beam Search (b=8)</td>
<td>7.22</td>
<td>0.901</td>
<td>31.4</td>
<td>6.10</td>
</tr>
<tr>
<td></td>
<td>WIR (Beam)</td>
<td>9.42</td>
<td>0.884</td>
<td>42.3</td>
<td>7.77</td>
</tr>
<tr>
<td></td>
<td>WIR (DELL)</td>
<td>9.62</td>
<td>0.882</td>
<td>46.4</td>
<td>7.69</td>
</tr>
<tr>
<td></td>
<td>WIR (PM2)</td>
<td>7.36</td>
<td>0.898</td>
<td>33.8</td>
<td>6.22</td>
</tr>
<tr>
<td></td>
<td>WIR (Gradient)</td>
<td>8.61</td>
<td>0.892</td>
<td>38.1</td>
<td>8.25</td>
</tr>
<tr>
<td></td>
<td>WIR (GA0)</td>
<td>10.11</td>
<td>0.881</td>
<td>51.4</td>
<td>9.93</td>
</tr>
<tr>
<td></td>
<td>Genetic Algorithm</td>
<td>8.18</td>
<td>0.895</td>
<td>35.8</td>
<td>6.41</td>
</tr>
<tr>
<td></td>
<td>PSO</td>
<td>8.71</td>
<td>0.894</td>
<td>39.0</td>
<td>6.46</td>
</tr>
<tr>
<td>SNLI</td>
<td>Greedy (b=1)</td>
<td>5.59</td>
<td>0.915</td>
<td>37.8</td>
<td>5.02</td>
</tr>
<tr>
<td></td>
<td>Beam Search (b=8)</td>
<td>5.59</td>
<td>0.916</td>
<td>37.8</td>
<td>5.02</td>
</tr>
<tr>
<td></td>
<td>Beam Search (b=8)</td>
<td>5.59</td>
<td>0.916</td>
<td>37.8</td>
<td>5.02</td>
</tr>
<tr>
<td></td>
<td>WIR (Beam)</td>
<td>6.56</td>
<td>0.911</td>
<td>42.8</td>
<td>5.65</td>
</tr>
<tr>
<td></td>
<td>WIR (DELL)</td>
<td>6.77</td>
<td>0.91</td>
<td>44.0</td>
<td>5.81</td>
</tr>
<tr>
<td></td>
<td>WIR (PM2)</td>
<td>5.63</td>
<td>0.915</td>
<td>37.8</td>
<td>5.05</td>
</tr>
<tr>
<td></td>
<td>WIR (Gradient)</td>
<td>6.57</td>
<td>0.911</td>
<td>41.6</td>
<td>5.9</td>
</tr>
<tr>
<td></td>
<td>WIR (GA0)</td>
<td>7.06</td>
<td>0.909</td>
<td>47.7</td>
<td>6.19</td>
</tr>
<tr>
<td></td>
<td>Genetic Algorithm</td>
<td>5.71</td>
<td>0.915</td>
<td>38.1</td>
<td>5.14</td>
</tr>
<tr>
<td></td>
<td>PSO</td>
<td>5.76</td>
<td>0.915</td>
<td>38.6</td>
<td>5.14</td>
</tr>
<tr>
<td>LSTM</td>
<td>Greedy (b=1)</td>
<td>7.19</td>
<td>0.899</td>
<td>33.6</td>
<td>5.98</td>
</tr>
<tr>
<td></td>
<td>Beam Search (b=8)</td>
<td>7.19</td>
<td>0.899</td>
<td>33.7</td>
<td>5.96</td>
</tr>
<tr>
<td></td>
<td>Beam Search (b=8)</td>
<td>7.19</td>
<td>0.899</td>
<td>34.0</td>
<td>5.96</td>
</tr>
<tr>
<td></td>
<td>WIR (Beam)</td>
<td>8.99</td>
<td>0.889</td>
<td>41.7</td>
<td>7.22</td>
</tr>
<tr>
<td></td>
<td>WIR (DELL)</td>
<td>9.17</td>
<td>0.889</td>
<td>44.5</td>
<td>7.21</td>
</tr>
<tr>
<td></td>
<td>WIR (PM2)</td>
<td>7.45</td>
<td>0.898</td>
<td>33.7</td>
<td>6.01</td>
</tr>
<tr>
<td></td>
<td>WIR (Gradient)</td>
<td>8.73</td>
<td>0.892</td>
<td>41.4</td>
<td>7.33</td>
</tr>
<tr>
<td></td>
<td>WIR (GA0)</td>
<td>10.60</td>
<td>0.880</td>
<td>54.0</td>
<td>9.31</td>
</tr>
<tr>
<td></td>
<td>Genetic Algorithm</td>
<td>6.90</td>
<td>0.896</td>
<td>36.4</td>
<td>6.36</td>
</tr>
<tr>
<td></td>
<td>PSO</td>
<td>8.41</td>
<td>0.893</td>
<td>40.2</td>
<td>6.32</td>
</tr>
</tbody>
</table>

Table 4: Quality evaluation of the adversarial examples produced by each search algorithm. "Avg P.W. %" means average percentage of words perturbed, "Avg USE Sim" means average USE angular similarity, and "Δ% Perplexity" means percent change in perplexities.