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.

[7] We choose counter-fitted embeddings because they encode synonym/antonym representations better than vanilla GLoVe embeddings (Mrksic et al., 2016).

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:

$$perturb(x, i) = \{T(x, i) \mid C_j(T(x, i)) \; \forall j \in \{1, ..., m\}\}$$

where $T(\mathbf{x}, i)$ represents the transformation method that swaps the i^{th} word x_i with its synonym to produce perturbed text$\mathbf{x}'$. $C_1, \ldots, C_m$ 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 $crossover$

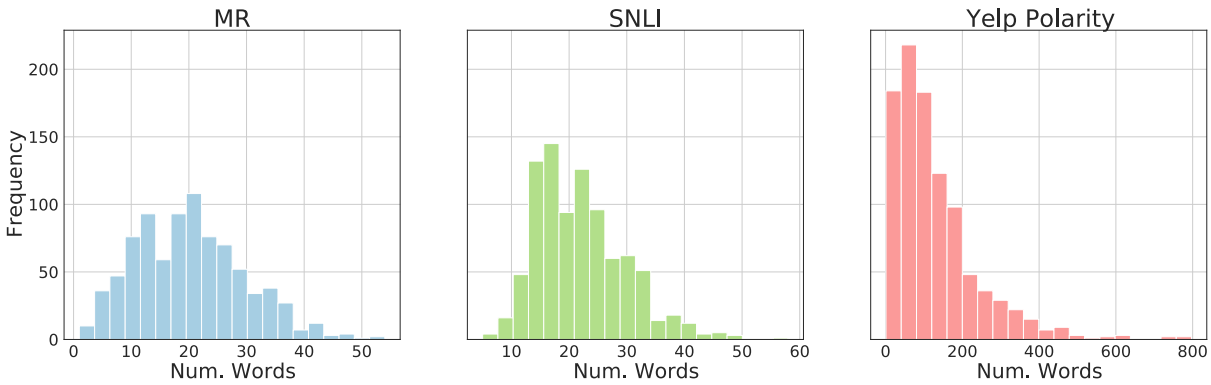

Figure 3: Histogram of words per dataset. Yelp inputs are generally much longer than inputs from MR or SNLI.

Algorithm 1 Beam Search with beam width b

Input: Original text $x = (x_1, x_2, ...x_n)$
Output: Adversarial text x_{adv} if found
  $best \leftarrow \{x\}$
  while $best == \emptyset$ **do**
    $X_{cand} \leftarrow \emptyset$
    for all $x_b \in best$ **do**
      for all $i \in \{1, \ldots, n\}$ **do**
        $X_{cand} \leftarrow X_{cand} \cup perturb(x_b, i)$
      end for
    end for
    if $X_{cand} \neq \emptyset$ **then**
      $x^* \leftarrow \arg\max_{x' \in X_{cand}} score(x')$
      if x^* fools the model **then**
        return x^* as x_{adv}
      else
        $best \leftarrow \{$top b elements of $X_{cand}\}$
        ▷ elements are ranked by their $score$
      end if
    else
      End search
    end if
  end while

Algorithm 2 Greedy Search

Input: Original text $x = (x_1, x_2, ...x_n)$
Output: Adversarial text x_{adv} if found
  $x^* \leftarrow x$
  while x_{adv} not found **do**
    $X_{cand} \leftarrow \emptyset$
    for all $i \in \{1, \ldots, n\}$ **do**
      $X_{cand} \leftarrow X_{cand} \cup perturb(x^*, i)$
    end for
    if $X_{cand} \neq \emptyset$ **then**
      $x^* \leftarrow \arg\max_{x' \in X_{cand}} score(x')$
      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 $x = (x_1, x_2, ...x_n)$
Output: Adversarial text x_{adv} if found
  $R \leftarrow$ 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**
    $X_{cand} \leftarrow perturb(x^*, i)$
    if $X_{cand} \neq \emptyset$ **then**
      $x^* \leftarrow \arg\max_{x' \in X_{cand}} score(x')$
      if x^* fools the model **then**
        return x^* as x_{adv}
      end if
    else
      End search
    end if
  end for

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**
    $i \leftarrow sample(x)$
    $X_{cand} \leftarrow perturb(x, i)$
    $P_k^0 \leftarrow \arg\max_{x' \in X_{cand}} score(x')$
  end for
  for $g = 1, \ldots, G$ generations **do**
    for $k = 1, \ldots, K$ **do**
      $S_k^{g-1} \leftarrow score(P_k^{g-1})$
    end for
    $x^* \leftarrow P^{g-1}_{\arg\max_j S_j^{g-1}}$
    if x^* fools the model **then**
      return x^* as x_{adv}
    else
      $P_1^g \leftarrow x^*$
      $p = Normalize(S^{g-1})$
      for $k = 2, \ldots, K$ **do**
        $par_1 \sim P^{g-1}$ with prob p
        $par_2 \sim P^{g-1}$ with prob p
        $child \leftarrow crossover(par_1, par_2)$
        $i \leftarrow sample(child)$
        $X_{cand} \leftarrow perturb(child, i)$
        $P_k^g \leftarrow \arg\max_{x' \in X_{cand}} score(x')$
      end for
    end if
  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.

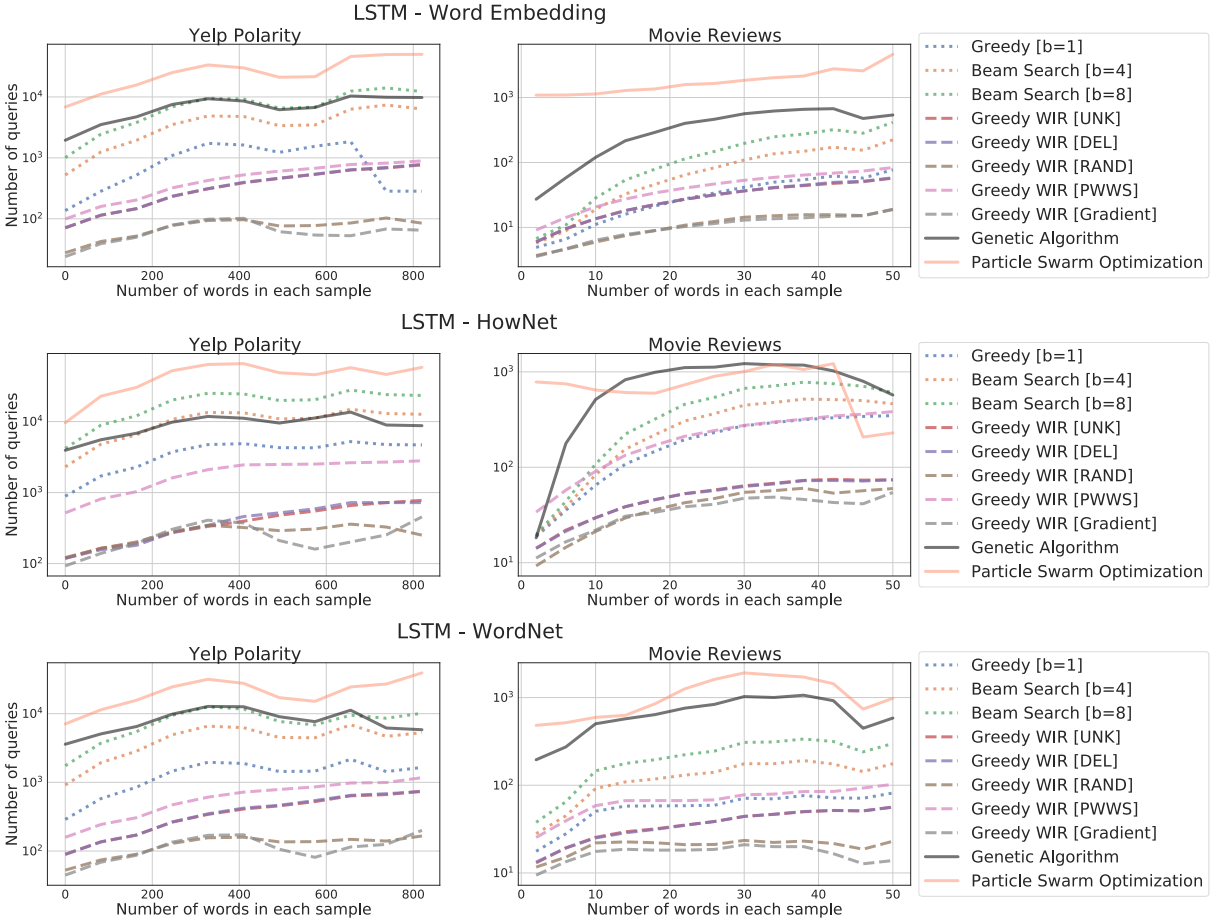

Figure 4: Number of queries vs. length of input text.

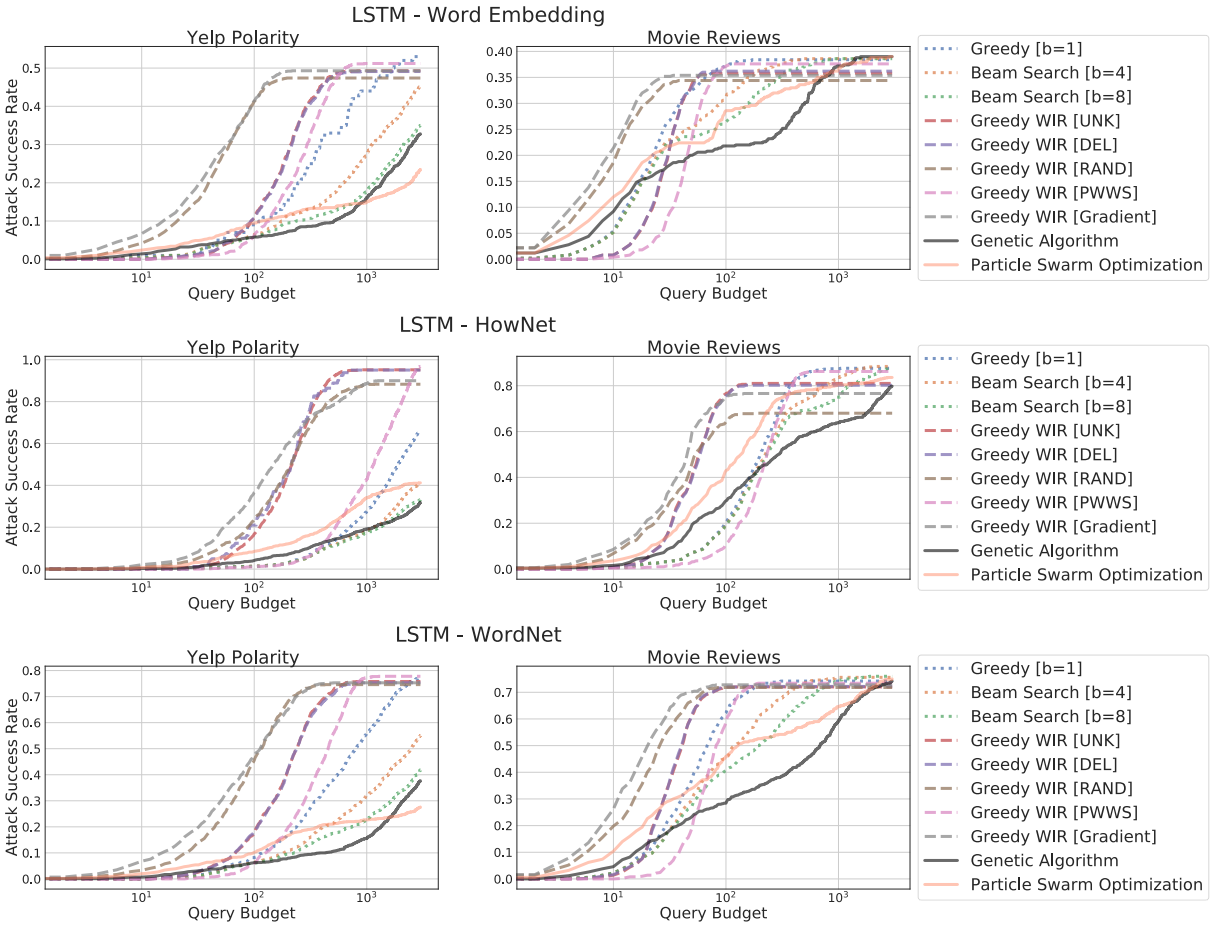

Figure 5: Attack success rate by query budget for each search algorithm and dataset.

Model	Dataset	Search Method	GLOVE Word Embedding			HowNet			WordNet		
			Avg P.W. %	Avg USE Sim	Δ% Perplexity	Avg P.W. %	Avg USE Sim	Δ% Perplexity	Avg P.W. %	Avg USE Sim	Δ% Perplexity
BERT	Yelp	Greedy (b=1)	3.41	0.948	21.5	2.52	0.945	22.8	4.76	0.943	49.9
		Beam Search (b=4)	3.26	0.949	20.7	2.45	0.946	22.0	4.49	0.944	46.7
		Beam Search (b=8)	**3.20**	**0.950**	**20.1**	**2.42**	**0.947**	**21.4**	**4.46**	**0.945**	**46.4**
		WIR (UNK)	6.48	0.930	43.5	4.73	0.922	42.3	9.02	0.924	92.1
		WIR (DEL)	6.85	0.928	47.2	5.10	0.919	46.4	9.38	0.923	98.8
		WIR (PWWS)	4.36	0.942	27.3	3.11	0.94	28.1	6.10	0.937	66.1
		WIR (Gradient)	6.16	0.933	37.8	5.58	0.913	44.5	9.10	0.925	86.4
		WIR (RAND)	8.18	0.920	59.1	7.46	0.898	74.6	11.16	0.914	124.8
		Genetic Algorithm	5.06	0.936	33.9	4.21	0.928	42.7	6.70	0.932	77.3
		PSO	6.61	0.929	47.3	6.08	0.913	62.3	9.67	0.922	111.0
	MR	Greedy (b=1)	7.25	0.900	31.8	6.14	0.887	36.5	10.26	0.864	102.8
		Beam Search (b=4)	**7.22**	**0.901**	**31.4**	**6.10**	**0.887**	**36.1**	10.10	**0.866**	**97.9**
		Beam Search (b=8)	**7.22**	**0.901**	**31.4**	**6.10**	**0.887**	**36.1**	**10.05**	**0.866**	101.6
		WIR (UNK)	9.42	0.884	42.3	7.77	0.866	48.0	14.14	0.845	141.2
		WIR (DEL)	9.62	0.882	46.4	7.69	0.865	46.1	14.60	0.840	146.4
		WIR (PWWS)	7.36	0.898	33.8	6.22	0.884	37.6	10.80	0.865	111.1
		WIR (Gradient)	8.61	0.892	38.1	8.25	0.862	40.8	14.58	0.844	123.2
		WIR (RAND)	10.1	0.881	51.4	9.93	0.846	69.5	17.28	0.827	149.4
		Genetic Algorithm	8.18	0.895	35.8	6.41	0.885	37.8	12.30	0.854	124.5
		PSO	8.71	0.894	39.0	6.46	0.884	38.7	16.08	0.839	187.8
	SNLI	Greedy (b=1)	5.59	0.915	37.8	5.02	0.889	31.7	6.53	0.903	55.9
		Beam Search (b=4)	**5.59**	**0.916**	**37.8**	**5.02**	**0.889**	**31.6**	**6.50**	**0.903**	**55.7**
		Beam Search (b=8)	5.59	0.916	37.8	5.02	0.889	31.6	6.50	0.903	55.9
		WIR (UNK)	6.56	0.911	42.8	5.65	0.887	33.4	8.03	0.899	65.5
		WIR (DEL)	6.77	0.91	44.0	5.81	0.887	34.2	8.22	0.898	67.6
		WIR (PWWS)	5.63	0.915	37.8	5.05	0.89	30.5	6.59	0.906	54.5
		WIR (Gradient)	6.57	0.911	41.6	5.9	0.881	37.7	8.06	0.899	65.1
		WIR (RAND)	7.06	0.909	47.7	6.19	0.884	42.9	8.65	0.895	74.6
		Genetic Algorithm	5.71	0.915	38.5	5.14	0.888	32.7	6.73	0.902	58.3
		PSO	5.76	0.915	38.6	5.14	0.888	32.5	6.94	0.902	58.5
LSTM	Yelp	Greedy (b=1)	4.04	0.943	28.9	2.47	0.948	23.9	4.58	0.946	52.1
		Beam Search (b=4)	4.01	0.942	28.9	2.47	0.949	23.7	4.53	0.946	51.9
		Beam Search (b=8)	**4.01**	**0.943**	**28.7**	**2.44**	**0.949**	**23.0**	**4.51**	**0.946**	**51.3**
		WIR (UNK)	5.83	0.933	42.4	3.51	0.935	34.4	7.22	0.935	75.6
		WIR (DEL)	5.86	0.932	41.1	3.61	0.936	33.3	7.22	0.935	75.4
		WIR (PWWS)	4.57	0.940	32.6	2.58	0.947	23.9	5.14	0.944	57.0
		WIR (Gradient)	7.05	0.926	52.2	5.25	0.916	50.7	8.42	0.929	87.9
		WIR (RAND)	7.28	0.925	53.6	6.33	0.906	69.5	9.40	0.925	102.4
		Genetic Algorithm	5.94	0.933	42.8	3.73	0.930	41.9	6.37	0.936	80.9
		PSO	6.70	0.929	47.3	5.03	0.924	58.7	7.98	0.93	95.2
	MR	Greedy (b=1)	**7.19**	**0.899**	**33.5**	**5.96**	0.884	37.2	10.21	0.871	100.6
		Beam Search (b=4)	7.19	0.899	33.7	5.96	0.884	37.6	10.03	0.871	98.7
		Beam Search (b=8)	7.19	0.899	34.0	5.96	0.884	37.6	**10.00**	**0.871**	97.4
		WIR (UNK)	8.99	0.889	41.7	7.22	0.874	42.9	12.99	0.856	104.5
		WIR (DEL)	9.17	0.889	44.5	7.21	0.874	42.2	13.03	0.856	107.5
		WIR (PWWS)	7.45	0.898	33.7	6.01	0.884	**37.1**	10.50	0.871	**87.9**
		WIR (Gradient)	8.73	0.892	41.4	7.33	0.870	40.8	13.12	0.859	104.1
		WIR (RAND)	10.60	0.880	54.0	9.31	0.853	57.7	16.05	0.842	148.2
		Genetic Algorithm	8.02	0.896	36.4	6.36	0.881	39.5	11.98	0.860	120.2
		PSO	8.41	0.893	40.2	6.32	0.882	40.8	13.90	0.854	130.0

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.