A Strong Baseline for Query Efficient Attacks in a Black Box Setting

Rishabh Maheshwary*, **Saket Maheshwary*** and **Vikram Pudi**
Data Sciences and Analytics Center, Kohli Center on Intelligent Systems
International Institute of Information Technology, Hyderabad, India
{rishabh.maheshwary@research.iiit.ac.in, vikram@iiit.ac.in}

Abstract

Existing black box search methods have achieved high success rate in generating adversarial attacks against NLP models. However, such search methods are inefficient as they do not consider the amount of queries required to generate adversarial attacks. Also, prior attacks do not maintain a consistent search space while comparing different search methods. In this paper, we propose a query efficient attack strategy to generate plausible adversarial examples on text classification and entailment tasks. Our attack jointly leverages attention mechanism and locality sensitive hashing (LSH) to reduce the query count. We demonstrate the efficacy of our approach by comparing our attack with *four* baselines across *three* different search spaces. Further, we benchmark our results across the same search space used in prior attacks. In comparison to attacks proposed, on an average, we are able to reduce the query count by 75% across all datasets and target models. We also demonstrate that our attack achieves a higher success rate when compared to prior attacks in a limited query setting.

1 Introduction

In recent years, Deep Neural Networks (DNNs) have achieved high performance on a variety of tasks (Yang et al., 2016; Goodfellow et al., 2016; Kaul et al., 2017; Maheshwary et al., 2017; Maheshwary and Misra, 2018; Devlin et al., 2018). However, prior studies (Szegedy et al., 2013; Papernot et al., 2017) have shown evidence that DNNs are vulnerable to *adversarial examples* — inputs generated by slightly changing the original input. Such changes are imperceptible to humans but they deceive DNNs, thus raising serious concerns about their utility in real world applications. Existing NLP attack methods are broadly classified into *white box attacks* and *black box attacks*. White box attacks require access to the target model's parameters, loss function and gradients to craft an adversarial attack. Such attacks are computationally expensive and require knowledge about the internal details of the target model which are not available in most real world applications. Black box attacks crafts adversarial inputs using only the confidence scores or class probabilities predicted by the target model. Almost all the prior black box attacks consists of two major components (1) *search space* and (2) *search method*.

A *search space* is collectively defined by a set of transformations (usually synonyms) for each input word and a set of constraints (e.g., minimum semantic similarity, part-of-speech (POS) consistency). The synonym set for each input word is generated either from the nearest neighbor in the counter-fitted embedding space or from a lexical database such as HowNet (Dong and Dong, 2003) or WordNet (Miller, 1995). The search space is variable and can be altered by either changing the source used to generate synonyms or by relaxing any of the above defined constraints.

A *search method* is a searching algorithm used to find adversarial examples in the above defined search space. Given an input with $\mathcal{W}$ words and each word having $\mathcal{T}$ possible substitutes, the total number of perturbed text inputs is $(\mathcal{T}+1)^{\mathcal{W}}$. Given this exponential size, the search algorithm must be efficient and exhaustive enough to find optimal adversarial examples from the whole search space.

Black box attacks proposed in (Alzantot et al., 2018; Zang et al., 2020) employ combinatorial optimization procedure as a search method to find adversarial examples in the above defined search space. Such methods are extremely slow and require massive amount of queries to generate adversarial examples. Attacks proposed in (Ren et al., 2019; Jin et al., 2019) search for adversarial examples using word importance ranking which first rank the input words and than substitutes them with similar words. The word importance ranking scores

*Equal Contribution

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 8396–8409
November 7–11, 2021. ©2021 Association for Computational Linguistics

each word by observing the change in the confidence score of the target model after that word is removed from the input (or replaced with <UNK> token). Although, compared to the optimization based methods, the word importance ranking methods are faster, but it has some major drawbacks – (1) each word is ranked by removing it from the input (or replacing it with a <UNK> token) which therefore alter the semantics of the input during ranking, (2) it not clear whether the change in the confidence score of the target model is caused by the removal of the word or the modified input and (3) this ranking mechanism is inefficient on input of larger lengths.

In general, their exists a trade off between the attack success rate and the number of queries. A high query search method generates adversarial examples with high success rate and vice-versa. All such prior attacks are not efficient and do not take into consideration the number of queries made to the target model to generate an attack. Such attacks will fail in real world applications where there is a constraint on the number of queries that can be made to the target model.

To compare a new search method with previous methods, the new search method must be benchmarked on the same search space used in the previous search methods. However, a study conducted in (Yoo et al., 2020) have shown that prior attacks often modify the search space while evaluating their search method. This does not ensure a fair comparison between the search methods because it is hard to distinguish whether the increase in attack success rate is due to the improved search method or modified search space. For example, (Jin et al., 2019) compares their search method with (Alzantot et al., 2018) where the former uses Universal Sentence Encoder (USE) (Cer et al., 2018) and the latter use language model as a constraint. Also, all the past works evaluate their search methods only on a single search space. In this paper[1], we address the above discussed drawbacks through following contributions:

1. We introduce a *novel ranking mechanism* that takes significantly less number of queries by jointly leveraging word attention scores and LSH to rank the input words; without altering the semantics of the input.

2. We call for unifying the evaluation setting by benchmarking our search method on the same search space used in the respective baselines. Further, we evaluate the effectiveness of our method by comparing it with *four* baselines across *three* different search spaces.

3. On an average, our method is 50% faster as it takes 75% lesser queries than the prior attacks while compromising the attack success rate by less than 2%. Further, we demonstrate that our search method has a much higher success rate than compared to baselines in a limited query setting.

2 Related Work

2.1 White Box Attacks

This category requires access to the gradient information to generate adversarial attacks. Hotflip (Ebrahimi et al., 2017) flips characters in the input using the gradients of the one hot input vectors. Liang et al. used gradients to perform insertion, deletion and replacement at character level. Later (Li et al., 2018) used the gradient of the loss function with respect to each word to find important words and replaced those with similar words. Following this, (Wallace et al., 2019) added triggers at the start of the input to generate adversarial examples. Although these attacks have a high success rate, but they require knowledge about the model parameters and loss function which is not accessible in real word scenarios.

2.2 Black Box Attacks

Existing black box attacks can be classified into combinatorial optimization based attacks and greedy attacks. Attack proposed in (Alzantot et al., 2018) uses Genetic algorithm as a search method to search for optimal adversarial examples in the search space. On similar lines, (Zang et al., 2020) uses Particle Swarm Optimization procedure to search for adversarial examples. Recently, (Maheshwary et al., 2021) crafted adversarial attacks using Genetic algorithm in a hard label black box setting. Although, such methods have a high success rate but they are extremely slow and takes large amount of queries to search for optimal adversarial examples.

Greedy black box attacks generates adversarial attacks by first finding important words, which highly impacts the confidence score of the target model and than replacing those words with

[1]Code and adversarial examples available at: https://github.com/RishabhMaheshwary/query-attack

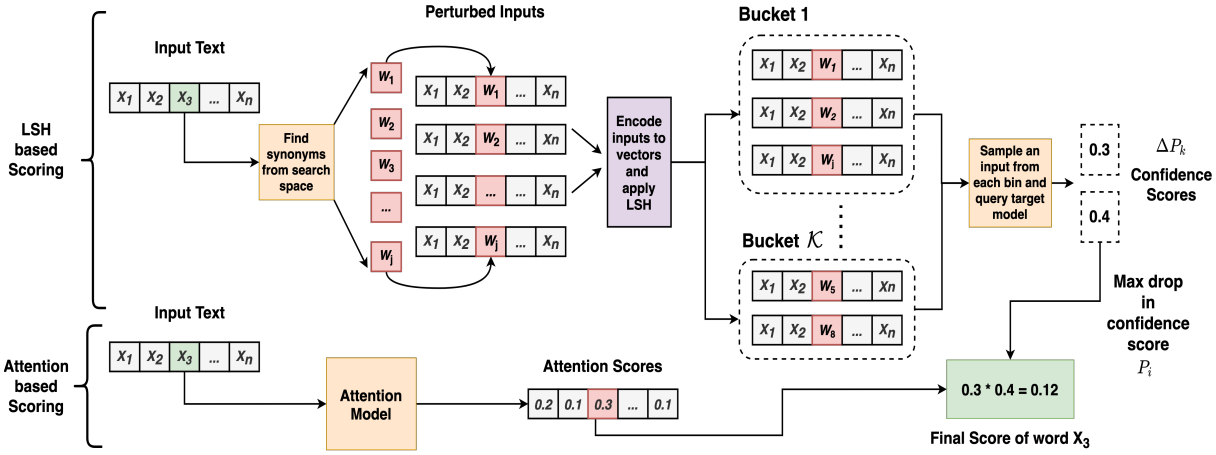


Figure 1: Scoring of each input word using attention mechanism and Locality Sensitive Hashing (LSH).

similar words. Search method proposed in (Ren et al., 2019) used word saliency and the target model confidence scores to rank words and replaced them with synonyms from WordNet (Miller, 1995). Although such a ranking mechanism is exhaustive, but it is not query efficient to rank the words. Inspired by this, (Jin et al., 2019) proposed TextFooler which ranks word only based upon the target model's confidence and replaces them with synonyms from the counter-fitted embedding space (Mrkšić et al., 2016). This ranking mechanism have a low attack success rate and is not exhaustive enough to search for adversarial examples with low perturbation rate.

Some prior works (Garg and Ramakrishnan, 2020; Li et al., 2020b; Maheshwary et al., 2020; Li et al., 2020a) have used masked language models to generate word replacements instead of using synonyms. However, all these methods follow a ranking mechanism similar to TextFooler. Moreover, as shown in (Yoo et al., 2020) most of the black box methods described above do not maintain a consistent search space while comparing their method with other search methods.

2.3 Locality Sensitive Hashing

LSH has been used in various NLP applications in the past. Ravichandran et al. used LSH for clustering nouns, (Van Durme and Lall, 2010) extended it for streaming data. Recently, (Kitaev et al., 2020) used LSH to reduce the computation time of self attention mechanism. There have been many variants of LSH, but in this paper we leverage the LSH method proposed in (Charikar, 2002).

3 Proposed Approach

Given a target model $\mathbf{F} : \mathcal{X} \rightarrow \mathcal{Y}$, that maps the input text sequence $\mathcal{X}$ to a set of class labels $\mathcal{Y}$. Our goal is to generate an adversarial text sequence $\mathcal{X}_{\mathcal{ADV}}$ that belongs to any class in $\mathcal{Y}$ except the original class of $\mathcal{X}$ i.e. $\mathbf{F}(\mathcal{X}) \neq \mathbf{F}(\mathcal{X}_{\mathcal{ADV}})$. The input $\mathcal{X}_{\mathcal{ADV}}$ must be generated by substituting the input words with their respective synonyms from a chosen search space.

Our search method consists of two steps (1) Word Ranking – ranks all words in the input text and (2) Word Substitution – substitutes input words with their synonyms in the order-of-rank (step 1).

3.1 Word Ranking

Recent studies (Niven and Kao, 2019) have shown evidence that certain words in the input and their replacement can highly influence the final prediction of DNNs. Therefore, we score each word based upon, (1) *how important it is for the final prediction* and (2) *how much its replacement with a similar word can alter the final prediction* of the target model. We use *attention mechanism* to select important words for classification and employ *LSH* to capture the impact of replacement of each word on the prediction of target model. Figure 1 demonstrates the working of the word ranking step.

3.1.1 Attention based scoring

Given an input $\mathcal{X}$, this step assigns high score to those influential words which impact the final outcome. The input sequence $\mathcal{X} = \{x_1, x_2..x_n\}$ is passed through a pre-trained attention model F_{attn} to get attention scores α_i for each word x_i. The scores are computed using Hierarchical Attention Networks (HAN) (Yang et al., 2016) and Decom-

pose Attention Model (DA) (Parikh et al., 2016) for text classification and entailment tasks respectively. Note, instead of querying the target model every time to score a word, this step scores all words together in a single pass (inferring the input sample by passing it through attention model), thus, significantly reducing the query count. Unlike prior methods, we do not rank each word by removing it from the input (or replacing it with a UNK token), preventing us from altering semantics of the input.

3.1.2 LSH based Scoring

This step assigns high scores to those words whose replacement with a synonym will highly influence the final outcome of the model. It scores each word based on the change in confidence score of the target model, when it is replaced by its substitute (or synonym) word. But computing the change in confidence score for each synonym for every input word, significantly large number of queries are required. Therefore, we employ LSH to solve this problem. LSH is a technique used for finding nearest neighbours in high dimensional spaces. It takes an input, a vector x and computes its hash $h(x)$ such that similar vectors gets the same hash with high probability and dissimilar ones do not. LSH differs from cryptographic hash methods as it aims to maximize the collisions of similar items. We leverage Random Projection Method (RPM) (Charikar, 2002) to compute the hash of each input text.

3.1.3 Random Projection Method (RPM)

Let us assume we have a collection of vectors in an m dimensional space R^m. Select a family of hash functions by randomly generating a spherically symmetrical random vector r of unit length from the m dimensional space. Then the hash function $h_r(u)$ is defined as:

$$h_r(u) = \begin{cases} 0 & r.u < 0 \\ 1 & r.u \geq 0 \end{cases} \tag{1}$$

Repeat the above process by generating d random unit vectors $\{r_0, r_1..r_d\}$ in R^m. The final hash $\bar{u}$ for each vector u is determined by concatenating the result obtained using on all d vectors.

$$\bar{u} = \{h_{r1}(u), h_{r2}(u)..h_{rd}(u)\}. \tag{2}$$

The hash of each vector u is represented by a sequence of bits and two vectors having same hash are mapped to same bucket in the hash table. Such a process is very efficient in finding nearest neighbor in high dimensional spaces as the hash code is generated using only the dot product between two matrices. Also, it is much easier to implement and simple to understand when compared to other nearest neighbour methods. We use the above process to score each word as follows:

1. First, an input word x_i is replaced with every synonym from its synonym set $S(x_i)$ resulting in perturbed sequence $\mathcal{X}_{ij} = \{x_1...w_j...x_n\}$, where i is the substituted index and j is the jth synonym from $S(x_i)$ i.e. $w_j \in S(x_i)$. Perturbed sequences not satisfying the search space constraints (Table 1) are filtered.

2. The remaining perturbed inputs are passed through a sentence encoder (USE) which returns a vector representation $\mathcal{V}_j$ for each perturbed input. Then we use LSH as described above to compute the hash of each vector. The perturbed inputs having the same hash are mapped to same bucket of the hash table.

$$\mathcal{V}_j = \mathbf{encode}(\mathcal{X}_{ij}) \quad \forall j;\ j \in [1, \mathcal{T}] \tag{3}$$
$$\mathcal{B} = \mathbf{LSH}(\mathcal{V}_j, \mathcal{X}_{ij}) \quad \forall j;\ j \in [1, \mathcal{T}] \tag{4}$$

   were $\mathcal{T}$ are the number of perturbed inputs obtained for each word and $\mathcal{B} = \{b_0, b_1...b_{\mathcal{K}}\}$ are the buckets obtained after LSH in a hash table, $\mathcal{K}$ being the number of buckets.

3. As each bucket contains similar perturbed inputs, a perturbed input is sampled randomly from each bucket and is passed to the target model $\mathbf{F}$. The maximum change in the confidence score of the target model among all these fed inputs is the score for that index.

$$\mathcal{V}_k^*, \mathcal{X}_k^* = sample(b_k)\ \forall k;\ k \in [0, \mathcal{K}] \tag{5}$$
$$\Delta P_k = \mathbf{F}(\mathcal{X}) - \mathbf{F}(\mathcal{X}_k^*)\ \forall k;\ k \in [0, \mathcal{K}] \tag{6}$$
$$P_i = \max(\Delta P_k) \qquad k \in [0, \mathcal{K}] \tag{7}$$

The steps 1 to 3 are repeated for all the indices in $\mathcal{X}$. LSH maps highly similar perturbed inputs to the same bucket, and as all such inputs are similar, they will impact the target model almost equally. Therefore, instead of querying the target model $\mathbf{F}$ for every input in the bucket, we sample an input randomly from each bucket and observe its impact on the target model $\mathbf{F}$. This will reduce the query count from being proportional to number of synonyms of each word to minimum number of buckets obtained after LSH.

3.1.4 False Negative error rate of LSH

Although LSH is efficient for finding nearest neighbour, there is still a small probability that similar perturbed inputs get mapped to different buckets. Therefore to reduce this probability, we conduct multiple rounds of hashing, $L = 15$, each round with a different family of hash functions and choose the round which has the most collisions i.e. round having minimum buckets. (Charikar, 2002) establishes an upper bound on the false negative error rate of LSH i.e two highly similar vectors are mapped to different buckets. The upper bound on the false negative error rate of LSH given by (for more details refer (Charikar, 2002))

$$(1 - P^d)^L \tag{8}$$

$$where\ P = 1 - \frac{\theta(u, v)}{\pi} \tag{9}$$

This shows that for given values of L and d, LSH maps similar vectors to the same bucket with very high probability. As LSH maps similar inputs to the same bucket with high probability, it cuts down the synonym search space drastically, thus reducing the number of queries required to attack. The dimension of the hash function d used in equation 2 is set to 5 and is same across all rounds.

3.1.5 Final Score Calculation

After obtaining the attention scores α_i and the scores from synonym words P_i for each index (calculated using LSH), we multiply the two to get the final score $score_i = \alpha_i * P_i$ for each word. All the words are sorted in descending order based upon the score $score_i$. The algorithm of the word ranking step is provided in the appendix A.

3.2 Word Substitution

We generate the final adversarial example for the input text by perturbing the words in the order retrieved by the word ranking step. For each word x_i in $\mathcal{W}$, we follow the following steps.

1. Each synonym w_j from the synonym set $S(x_i)$ is substituted for x_i which results in a perturbed text $X'_j = \{x_1..w_j..x_n\}$. The perturbed texts which do not satisfy the constraints imposed on the search space (Table 1) are filtered (Algorithm 1, lines $1 - 7$).

2. The remaining perturbed sequence(s) are fed to the target model $\mathbf{F}$ to get the class label y_{new} and its corresponding probability score P_{new}. The perturbed sequence(s) which alters the original class label y_{orig} is chosen as the final adversarial example $\mathcal{X}_{ADV}$. In case the original label does not change, the perturbed sequence which has the minimum P_{new} is chosen (Algorithm 1, lines $7 - 14$).

The steps $1 - 2$ are repeated on the chosen perturbed sequence for the next ranked word. Note, we only use LSH in the word ranking step, because in ranking we need to calculate scores for all the words in the input and so we need to iterate over all possible synonyms of every input word. However, in the word substitution step we replace only one word at a time and the substitution step stops when we get an adversarial example. As the number of substitutions are very less (see perturbation rate in Table 3) to generate adversarial example, the substitution step iterate over very less words when compared to ranking step.

Algorithm 1 Word Substitution

Input: Test sample $\mathcal{X}$, Ranked words $\mathcal{W}$
Output: Adversarial text $\mathcal{X}_{ADV}$

1: $\mathcal{X}_{ADV} \leftarrow \mathcal{X}$
2: y_{orig} , $P_{orig} \leftarrow \mathbf{F}(\mathcal{X}_{ADV})$
3: $P_{best} \leftarrow P_{orig}$
4: **for** $(score_i, x_i)$ *in* $\mathcal{W}$ **do**
5: $\quad S \leftarrow Synonyms(x_i)$
6: $\quad$ **for** w_j *in* S **do**
7: $\quad\quad X'_j \leftarrow$ *Replace* x_i *with* w_j
8: $\quad\quad y_{new}$, $P_{new} \leftarrow \mathbf{F}(X'_j)$
9: $\quad\quad$ **if** $y_{new} \neq y_{orig}$ **then**
10: $\quad\quad\quad \mathcal{X}_{ADV} \leftarrow X'_j$
11: $\quad\quad\quad$ **return** $\mathcal{X}_{ADV}$
12: $\quad\quad$ **if** $P_{new} < P_{best}$ **then**
13: $\quad\quad\quad P_{best} \leftarrow P_{new}$
14: $\quad\quad\quad \mathcal{X}_{ADV} \leftarrow X'_j$
15: **return** $\mathcal{X}_{ADV}$

4 Experiments

4.1 Datasets and Target Models

We use *IMDB* — A document level sentiment classification dataset for movie reviews (Maas et al., 2011) and *Yelp Reviews* — A restaurant review dataset (Zhang et al., 2015), for classification task. We use *MultiNLI* — A natural language inference dataset (Williams et al., 2017) for entailment task. We attacked *WordLSTM* (Hochreiter and Schmidhuber, 1997) and *BERT-base-uncased* (Devlin et al., 2018) for evaluating our attack strategy on text

classification and entailment tasks. For WordLSTM, we used a single layer bi-directional LSTM with 150 hidden units, a dropout of 0.3 and 200 dimensional GloVe (Pennington et al., 2014) vectors. Additional details are provided in appendix A.

4.2 Search Spaces and Baselines

We compare our search method with four baselines across three different search spaces. Also, while comparing our results with each baselines we use the same search space as used in that baseline paper. The details of search spaces are shown in Table 1.

PSO: (Zang et al., 2020) It uses particle swarm optimization algorithm as a search method and uses HowNet (Dong and Dong, 2006) as search space.

TextFooler: (Jin et al., 2019) It finds important words and replace them with synonyms from counter-fitted embeddings (Mrkšić et al., 2016).

PWWS: (Ren et al., 2019) It takes word saliency score into account to rank words and uses WordNet (Miller, 1995) for synonym substitution.

Genetic Attack: It crafts examples using a population based algorithm (Alzantot et al., 2018).

Baseline	Transformation	Constraint
Genetic Attack	Counter-fitted	Word similarity, LM score
TextFooler	embeddings	USE = 0.84, POS consistency
PSO	HowNet	USE = 0.84, POS consistency
PWWS	WordNet	POS consistency

Table 1: Baselines and their search spaces

4.3 Experimental Settings

In a black box setting, the attacker has no access to the training data of the target model. Therefore, we made sure to train the attention models on a different dataset. For attacking the target model trained on IMDB, we trained our attention model on the Yelp Reviews and vice-versa. For entailment, we trained the attention model on SNLI (Bowman et al., 2015) and attacked the target model trained on MNLI. Following (Jin et al., 2019), the target models are attacked on same 1000 samples, sampled from the test set of each dataset. The same set of samples are used across all baselines when evaluating on a single dataset. For entailment task we only perturb the premise and leave the hypothesis unchanged. We used spacy for POS tagging and filtered out stop words using NLTK. We used Universal Sentence Encoder (Cer et al., 2018) to encode the perturbed inputs while performing LSH. The hyperparameters d and L are tuned on the validation set (10% of each dataset). Additional details regarding hyperparameter tuning and attention models can be found in appendix.

4.4 Evaluation Metrics

We use (1) *attack success rate* – the ratio of the successful attacks to total number of attacks, (2) *query count* – the number of queries, (3) *perturbation rate* – the percentage of words substituted in an input and (4) *grammatical correctness* – the average grammatical error increase rate (calculated using Language-Tool[2]) to verify the quality of generated adversarial examples. For all the metrics except attack success rate, lower the value better the result. Also, for all metrics, we report the average score across all the generated adversarial examples on each dataset. Further, we also conducted human evaluation to assess the quality of generated adversarial examples.

4.5 Results

Table 2 and 4 shows the comparison of our proposed method with each baseline across all evaluation metrics. On an average across all baselines, datasets and target models we are able to reduce the query count by 75%. The PSO and Genetic attack takes atleast $50x$ and $20x$ more queries respectively than our attack strategy. Also, when compared to PWWS and TF we are able to reduce the query count by atleast 65% and 33% respectively while compromising the success rate by less than 2.0%. The perturbation rate and the grammatical correctness is also within 1% of the best baseline. In comparison to PSO and Genetic Attack, we are able to achieve even a lower perturbation rate and grammatical error rate with much lesser queries across some datasets and target models. Similarly, our attack outperforms TextFooler almost on all evaluation metrics. The runtime comparison and the anecdotes from generated adversarial text are provided in the appendix.

5 Ablation Study

We study the effectiveness of attention and LSH component in our method by doing a three way ablation. We observe the change in success rate, perturbation rate and queries when both or either one of the two ranking components are removed.

No LSH and attention: First, we remove both the attention and LSH scoring steps and rank the

[2]https://languagetool.org/

Model	Attack	IMDB		Yelp		MNLI	
		Qrs	Suc%	Qrs	Suc%	Qrs	Suc%
BERT	PSO	81350.6	**99.0**	73306.6	**93.2**	4678.5	**57.97**
	Ours	**737**	97.4	**554.2**	91.6	**97.2**	56.1
LSTM	PSO	52008.7	**99.5**	43671.7	**95.4**	2763.3	**67.8**
	Ours	**438.1**	**99.5**	**357.6**	94.75	**79.8**	66.4

(a) Comparison with PSO.

Model	Attack	IMDB		Yelp		MNLI	
		Qrs	Suc%	Qrs	Suc%	Qrs	Suc%
BERT	Gen	7944.8	66.3	6078.1	**85.0**	1546.8	**83.8**
	Ours	**378.6**	**71.1**	**273.7**	84.4	**43.4**	81.9
LSTM	Gen	3606.9	97.2	5003.4	**96.0**	894.5	**87.8**
	Ours	**224**	**98.5**	**140.7**	95.4	**39.9**	86.4

(b) Comparison with Genetic Attack.

Model	Attack	IMDB		Yelp		MNLI	
		Qrs	Suc%	Qrs	Suc%	Qrs	Suc%
BERT	PWWS	1583.9	**97.5**	1013.7	**93.8**	190	**96.8**
	Ours	**562.9**	96.4	**366.2**	92.6	**66.1**	95.1
LSTM	PWWS	1429.2	**100.0**	900.0	**99.1**	160.2	**98.8**
	Ours	**473.8**	**100.0**	**236.3**	**99.1**	**60.1**	98.1

(c) Comparison with PWWS.

Model	Attack	IMDB		Yelp		MNLI	
		Qrs	Suc%	Qrs	Suc%	Qrs	Suc%
BERT	TF	1130.4	**98.8**	809.9	**94.6**	113	85.9
	Ours	**750**	98.4	**545.5**	93.2	**100**	**86.2**
LSTM	TF	544	**100.0**	449.4	**100.0**	105	95.9
	Ours	**330**	**100.0**	**323.7**	**100.0**	**88**	**96.2**

(d) Comparison with TextFooler (TF).

Table 2: Result comparison. Succ% is the attack success rate and Qrs is the average query count. Note as each baseline uses a different search space, our method will yield different results when comparing with each baseline.

Model	Attack	IMDB		Yelp		MNLI	
		Pert%	I%	Pert%	I%	Pert%	I%
BERT	PSO	4.5	0.20	10.8	0.30	8.0	3.5
	Ours	**4.2**	**0.10**	**7.8**	**0.15**	**7.1**	**3.3**
LSTM	PSO	2.2	0.15	7.7	0.27	**6.7**	**1.27**
	Ours	**2.0**	**0.11**	**4.9**	**0.15**	6.8	1.3

(a) Comparison with PSO.

Model	Attack	IMDB		Yelp		MNLI	
		Pert%	I%	Pert%	I%	Pert%	I%
BERT	Gen	**6.5**	1.04	11.6	1.5	**8.7**	**1.9**
	Ours	6.7	**1.02**	**10.5**	**1.49**	9.2	2.1
LSTM	Gen	4.1	0.62	8.6	1.3	7.7	2.5
	Ours	**3.19**	**0.56**	**6.2**	**1.05**	**8.2**	**2.1**

(b) Comparison with Genetic Attack.

Model	Attack	IMDB		Yelp		MNLI	
		Pert%	I%	Pert%	I%	Pert%	I%
BERT	PWWS	**5.2**	**0.74**	**7.3**	**1.5**	**7.1**	1.71
	Ours	7.5	0.9	9.9	1.9	9.6	**1.48**
LSTM	PWWS	2.3	**0.3**	**4.8**	**1.29**	**6.6**	**1.5**
	Ours	**1.9**	0.4	5.5	**1.29**	7.8	2.1

(c) Comparison with PWWS.

Model	Attack	IMDB		Yelp		MNLI	
		Pert%	I%	Pert%	I%	Pert%	I%
BERT	TF	9.0	1.21	5.2	**1.1**	11.6	**1.23**
	Ours	**6.9**	**0.9**	**6.6**	1.2	**11.4**	1.41
LSTM	TF	**2.2**	2.3	5.7	2.06	**9.8**	1.7
	Ours	2.4	**1.5**	**5.3**	**1.5**	10.1	**1.4**

(d) Comparison with TextFooler (TF).

Table 3: Result comparison. Pert% is the perturbation and I% is the average grammatical error increase.

words in random order. Table 4 shows the results obtained on BERT across all three datasets. On an average the attack success rate drops by 7%, the perturbation rate increases drastically by 6%. This shows that although the query count reduces, substituting words in random order degrades the quality of generated adversarial examples and is not effective for attacking target models.

Attention and no LSH: We remove the LSH component of our ranking step and rank words based upon only the attention scores obtained from the attention model. Table 4 shows the results on BERT across all datasets. On an average the attack success rate drops by 2.5%, the perturbation rate increases by 3% and the query increases by 37%. Therefore, LSH reduces the queries significantly by eliminating near duplicates in search space.

LSH and no Attention: We remove the attention component and rank words using only LSH. Results in Table 4 shows that on an average, without attention scoring the attack success rate drops by 2%, the perturbation rate increases by 0.5% and the query increases by 20%. Therefore, attention is important as it not only reduces queries but also enables the ranking method to prioritize important words required in target model prediction.

With LSH and Attention: Finally in Table 4 we observe that, using both LSH and attention in our ranking our attack has a much better success rate, a lower perturbation rate in much lesser queries. This shows that both the components are necessary to do well across all evaluation metrics. We obtained similar results on LSTM when evaluating across different datasets and search spaces.

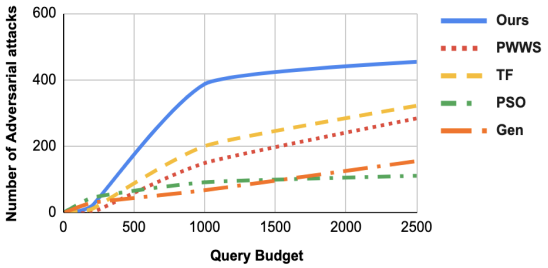
6 Quantitative Analysis

6.1 Limited Query Setting

In this setting, the attacker has a fixed query budget L, and can generate an attack in L queries or

(a) Adv. samples generated against BERT on IMDB

(b) Adv. samples generated against BERT on Yelp



(c) Adv. samples generated against LSTM on IMDB

(d) Adv. samples generated against LSTM on Yelp

Figure 2: Comparison of the number of adversarial samples generated by varying the query budget L.

Dataset	Random			Only Attention			Only LSH			Both LSH and Attention		
	Suc%	Pert%	Qrs	Suc%	Pert%	Qrs	Suc%	Pert%	Qrs	Suc%	Pert%	Qrs
IMDB	90.5	13.3	507.9	94.0	9.3	851.3	95.3	8.0	694.9	**96.4**	**7.5**	**562.9**
Yelp	87.3	15.0	305.9	91.0	11.0	550.0	90.2	10.2	475.2	**92.6**	**9.8**	**366.2**
MNLI	88.8	14.3	60.1	92.4	11.7	121.2	94.3	10.1	100.1	**95.1**	**9.6**	**66.1**

Table 4: Ablation Study of attention mechanism and LSH on PWWS search space.

less. To demonstrate the efficacy of our attack under this constraint, we vary the query budget L and observe the attack success rate on BERT and LSTM across IMDB and Yelp datasets. We vary the query budget from 0 to 2500 and observe how many adversarial examples can be generated successfully on a test set of 500 samples. We keep the search space same (used in PWWS) across all the search methods. The results in Figure 7 shows that with a query budget of 1000, our approach generates atleast 200 (44.4%) more adversarial samples against both BERT and LSTM on IMDB when compared to the best baseline. Similarly, on Yelp our method generates atleast 100 (25%) more adversarial samples on BERT and LSTM when compared to the best baseline. This analysis shows that our attack has a much higher success rate in a limited query setting, thus making it extremely useful for real world applications.

6.2 Input Length

To demonstrate that how our strategy scales with change in the input length (number of words in the input) compared to other baselines, we attacked BERT on Yelp. We selected inputs having number of words in the range of 10 to 250 and observed the number of queries taken by each attack method. Results in figure 3 shows that our attack takes the least number of queries across all input lengths. Further, our attack scales much better on longer inputs ($>$ 250 words) as it is $2x$ faster than PWWS and TextFooler, $13x$ faster than Genetic attack and $133x$ faster than PSO.

6.3 Transferability

An adversarial example is said to be *transferable*, if it is generated against one particular target model but is able to fool other target models as well. We implemented transferability on IMDB and MNLI datasets across two target models. The results are shown in Table 5. Our transferred examples dropped the accuracy of other target models on an average by 16%.

6.4 Adversarial Training

We randomly sample 10% samples from the training dataset of MNLI and IMDB and generate adversarial examples using our proposed strategy.

Transfer	Accurracy	IMDB	MNLI
BERT → LSTM	Original	90.9	85.0
	Transferred	**72.9**	**60.6**
LSTM → BERT	Original	88.0	70.1
	Transferred	**67.7**	**62.1**

Table 5: Transferability on IMDB and MNLI datasets

We augmented the training data with the generated adversarial examples and re-trained BERT on IMDB and MNLI tasks. We then again attacked BERT with our proposed strategy and observed the changes. The results in Figure 4 shows that as we add more adversarial examples to the training set, the model becomes more robust to attacks. The after attack accuracy and perturbation rate increased by 35% and 17% respectively and required higher queries to attack.

7 Qualitative Analysis

7.1 Human Evaluation

We verified the quality of generated adversarial samples via human based evaluation as well. We asked the evaluators to classify the adversarial examples and score them in terms of grammatical correctness (score out of 5) as well as its semantic similarity compared to the original text. We randomly sampled 25% of original instances and their corresponding adversarial examples generated on BERT for IMDB and MNLI datasets on PWWS search space. The actual class labels of adversarial examples were kept hidden and the human judges were asked to classify them. We also asked the human judges to evaluate each sample for its semantic similarity and assign a score of 0, 0.5 or 1 based on how well the adversarial examples were able to retain the meaning of their original counterparts. We also asked them to score each example in the range 1 to 5 for grammatical correctness. Each adversarial example was evaluated by 3 human evaluators

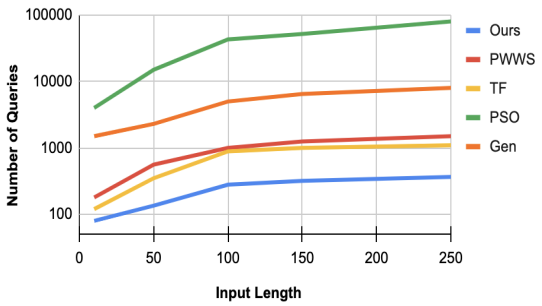


Figure 3: Queries taken vs number of words in input

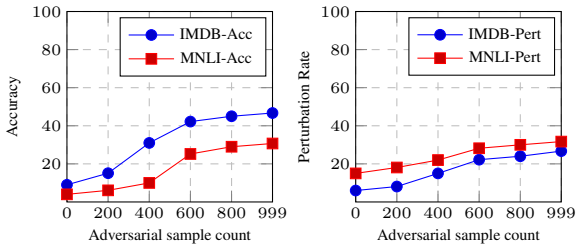


Figure 4: Increase in after attack accuracy and perturbation rate as more adversarial samples are augmented.

and the scores obtained were averaged out. The outcome is in Table 6.

Evaluation criteria	IMDB	MNLI
Classification result	94%	91%
Grammatical Correctness	4.32	4.12
Semantic Similarity	0.92	0.88

Table 6: Demonstrates scores given by evaluators

8 Conclusion

We proposed a query efficient attack that generates plausible adversarial examples on text classification and entailment tasks. Extensive experiments across *three* search spaces and *four* baselines shows that our attack generates high quality adversarial examples with significantly lesser queries. Further, we demonstrated that our attack has a much higher success rate in a limited query setting, thus making it extremely useful for real world applications.

9 Future Work

Our proposed attack provides a strong baseline for more query efficient black box attacks. The existing word level scoring methods can be extended to sentence level. Also, the attention scoring model used can be trained on different datasets to observe how the success rate and the query efficiency gets affected. Furthermore, existing attack methods can be evaluated against various defense methods to compare the effectiveness of different attacks.

10 Acknowledgement

We would like to thank all the reviewers for their critical insights and positive feedback. We would also like to thank Riyaz Ahmed Bhat for having valuable discussions which strengthen our paper and helped us in responding to reviewers.

References

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating natural language adversarial examples. *arXiv preprint arXiv:1804.07998*.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Zhendong Dong and Qiang Dong. 2003. Hownet-a hybrid language and knowledge resource. In *International Conference on Natural Language Processing and Knowledge Engineering, 2003. Proceedings. 2003*, pages 820–824. IEEE.

Zhendong Dong and Qiang Dong. 2006. *Hownet and the computation of meaning (with Cd-rom)*. World Scientific.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2017. Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751*.

Siddhant Garg and Goutham Ramakrishnan. 2020. Bae: Bert-based adversarial examples for text classification. *arXiv preprint arXiv:2004.01970*.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2019. Is bert really robust? natural language attack on text classification and entailment. *arXiv preprint arXiv:1907.11932*.

Ambika Kaul, Saket Maheshwary, and Vikram Pudi. 2017. Autolearn—automated feature generation and selection. In *2017 IEEE International Conference on data mining (ICDM)*, pages 217–226. IEEE.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

Dianqi Li, Yizhe Zhang, Hao Peng, Liqun Chen, Chris Brockett, Ming-Ting Sun, and Bill Dolan. 2020a. Contextualized perturbation for textual adversarial attack. *arXiv preprint arXiv:2009.07502*.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2018. Textbugger: Generating adversarial text against real-world applications. *arXiv preprint arXiv:1812.05271*.

Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020b. Bert-attack: Adversarial attack against bert using bert. *arXiv preprint arXiv:2004.09984*.

Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. 2017. Deep text classification can be fooled. *arXiv preprint arXiv:1704.08006*.

Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics.

Rishabh Maheshwary, Saket Maheshwary, and Vikram Pudi. 2020. A context aware approach for generating natural language attacks. *arXiv preprint arXiv:2012.13339*.

Rishabh Maheshwary, Saket Maheshwary, and Vikram Pudi. 2021. Generating natural language attacks in a hard label black box setting. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*.

Saket Maheshwary, Soumyajit Ganguly, and Vikram Pudi. 2017. Deep secure: A fast and simple neural network based approach for user authentication and identification via keystroke dynamics. In *IWAISe: First International Workshop on Artificial Intelligence in Security*, page 59.

Saket Maheshwary and Hemant Misra. 2018. Matching resumes to jobs via deep siamese network. In *Companion Proceedings of the The Web Conference 2018*, pages 87–88.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Nikola Mrkšić, Diarmuid O Séaghdha, Blaise Thomson, Milica Gašić, Lina Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016. Counter-fitting word vectors to linguistic constraints. *arXiv preprint arXiv:1603.00892*.

Timothy Niven and Hung-Yu Kao. 2019. Probing neural network comprehension of natural language arguments. *arXiv preprint arXiv:1907.07355*.

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519.

Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. 2005. Randomized algorithms and nlp: Using locality sensitive hash functions for high speed noun clustering. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 622–629.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1085–1097.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Benjamin Van Durme and Ashwin Lall. 2010. Online generation of locality sensitive hash signatures. In *Proceedings of the ACL 2010 conference short papers*, pages 231–235.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal adversarial triggers for attacking and analyzing nlp. *arXiv preprint arXiv:1908.07125*.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489.

Jin Yong Yoo, John X Morris, Eli Lifland, and Yanjun Qi. 2020. Searching for a search method: Benchmarking search algorithms for generating nlp adversarial examples. *arXiv preprint arXiv:2009.06368*.

Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. 2020. Word-level textual adversarial attacking as combinatorial optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6066–6080.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657.

A Appendix

Algorithm 2 Word Ranking

Input: Test sample $\mathcal{X}$

Output: $\mathcal{W}$ containing score of each word x_i

$F_{attn} \leftarrow HAN()\ or\ DA()$
$\alpha \leftarrow F_{attn}(\mathcal{X})$
for x_i **in** $\mathcal{X}$ **do**
  $S \leftarrow Synonyms(x_i)$
  for w_j **in** S **do**
    $\mathcal{X}_{ij} \leftarrow Replace\ x_i\ with\ w_j\ in\ \mathcal{X}$
    $\mathcal{V}_j \leftarrow encode(\mathcal{X}_{ij})$
    $\{b_1..b_{\mathcal{K}}\} = \mathbf{LSH}(\mathcal{V}_j, \mathcal{X}_{ij})$
  for $k = 1\ to\ \mathcal{K}$ **do**
    $\mathcal{V}_k^*, \mathcal{X}_k^* \leftarrow sample(b_k)$
    $\Delta P_k = \mathbf{F}(\mathcal{X}) - \mathbf{F}(\mathcal{X}_k^*)$
  $P_i = \max(\Delta P_k)$
  $score_i \leftarrow \alpha_i * P_i$
  $\mathcal{W}.insert((score_i, x_i))$
$Sort\ \mathcal{W}\ by\ score_i\ in\ descending\ order$

Dataset	Train	Test	Classes	Avg. Len
IMDB	12K	12K	2	215
Yelp	560K	18K	2	152
MultiNLI	12K	4K	3	10

Table 7: Statistics of all datasets

A.1 Models

Target Models: We attacked WordLSTM and BERT-base-uncased for evaluating our attack strategy on text classification and entailment tasks. For WordLSTM a single layer bi-directional LSTM with 150 hidden units, 200 dimensional GloVe vectors and a dropout of 0.3 were used.

Ranking Models: We used Hierarchical Attention Networks (HAN) and Decompose Attention Model (DA) for classification and entailment tasks respectively. For training HA, we used 200 dimensional word2vec embeddings and 50 dimensinal GRU cells. We used 100 dimensional word context vectors initialized randomly. We trained the model with a batch size of 64, learning rate of 0.0001, dropout of 0.3 and momentum of 0.9. For DA, a 2 layer LSTM with 200 neurons and 200 dimensional glove embeddings are used. A batch size of 32, learning rate of 0.025, dropout of 0.2 are used. All the hyper-parameters were tuned on the 20% validation set of each dataset.

Attack	Runtime
PSO	72 hrs
Genetic Attck	10 hrs
PWWS	3 hrs
TextFooler	2.5 hrs
Ours	1 hrs

Table 8: Runtime comparison while attacking BERT trained on Yelp dataset on a set of 500 samples across WordNet search space.

A.2 Hyperparameter Study

Figure 5 and 6 shows the variation of attack success rate and queries taken to attack the target model as d increases. With increase in d the number of collisions decreases and therefore the number of buckets obtained $\mathcal{K}$ increases. This increases the overall query count. Also, with increase in d the success rate first increases and then remains unchanged. Therefore, we use $d = 5$ because after that the success rate is almost the same but the query count increases drastically. Figure 7a and 7b shows the variation of attack success rate and queries taken to attack the target model as the rounds of hashing L increases. Conducting multiple rounds of hashing reduces the probability that that similar perturbed text inputs are mapped to different buckets. We choose $L = 15$ as after it the attack success rate and the queries remain almost unchanged. The values of d, L are kept same across all datasets and target models.

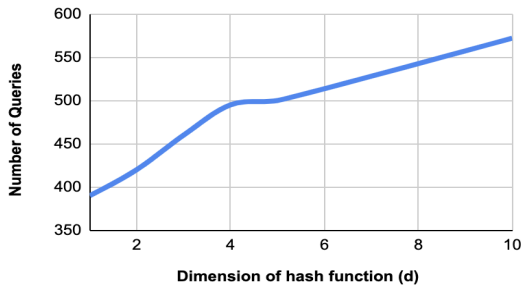


Figure 5: Queries vs the dimension d of hash function

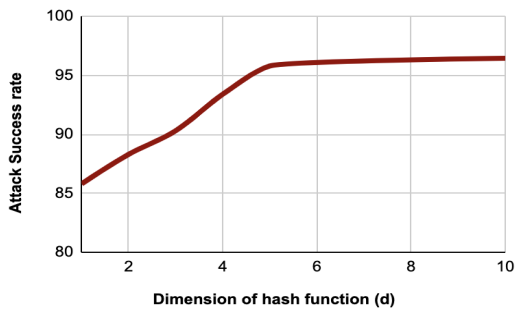
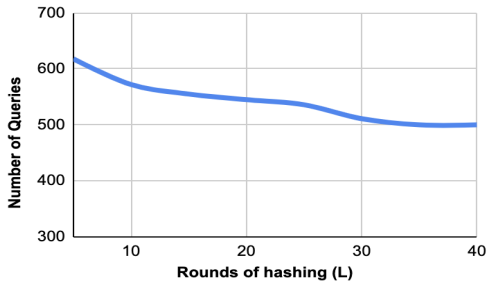
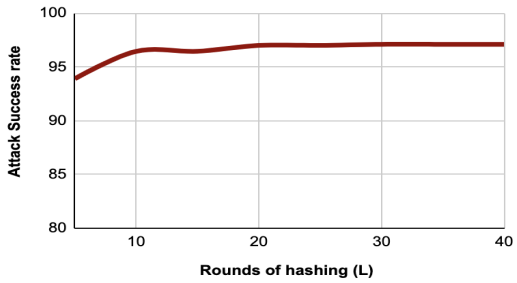


Figure 6: Success rate vs dimension d of hash function



(a) Queries vs rounds of hashing L



(b) Success rate vs rounds of hashing L

Figure 7: Comparison of success rate and queries required by changing the rounds of hashing L.

Examples	**Prediction**
The movie has an excellent screenplay (the situation is credible, the action has pace), **first-class** [**fantabulous**] direction and acting (especially the 3 leading actors but the others as well -including the mobster, who does not seem to be a professional actor). I **wish** [**want**] the movie, the director and the actors success.	**Positive** → **Negative**
Let me start by saying I don't recall laughing once during this comedy. From the opening scene, our protagonist Solo (Giovanni Ribisi) shows himself to be a self-absorbed, feeble, and neurotic loser completely unable to cope with the smallest responsibilities such as balancing a checkbook, keeping his word, or forming a coherent thought. I guess we're supposed to be drawn to his fragile vulnerability and cheer him on through the process of clawing his way out of a deep depression. I actually **wanted** [**treasured**] him to get his kneecaps busted at one point. The dog was not a character in the film. It was simply a prop to be used, neglected, scorned, abused, coveted and disposed of on a whim. So be warned.	**Negative** → **Postive**
Local-international **gathering** [**assembly**] **spot** [**stain**] since the 1940s. One of the coolest pubs on the planet. Make new friends from all over the world, with some of the **best** [**skilful**] regional and imported beer selections in town.	**Postive** → **Negative**
This film is strange, even for a silent movie. Essentially, it follows the adventures about a engineer in post-revolutionary Russia who daydreams about going to Mars. In this movie, it seems like the producers KNOW the Communists have truly screwed up the country, but also seems to want to make it look like they've accomplished something good. Then we get to the "Martian" scenes, where everyone on Mars wears goofy hats. They have a revolution after being inspired by the Earth Men, but are quickly betrayed by the Queen who sides with them. Except it's all a dream, or is it. (And given that the Russian Revolution eventually lead to the Stalin dictatorship, it makes you wonder if it was all allegory.) **Now** [**Nowdays**], I've seen GOOD Russian cinema. For instance, Eisenstein's Battleship Potemkin is a good movie. This is just, well, silly.	**Negative** → **Postive**
This movie is one of the **worst** [**tough**] **remake** [**remakes**] I have ever seen in my **life** [**aliveness**]! The acting is **laughable** [**funnny**] and Corman has not improved his piranhas any since 1978. 90% of the **special** [**exceptional**] **effects** [**impressions**] are **taken** [**lifted**] from Piranha (1978), Up From The Depths (1979) and Humanoids From The Deep (1979). It makes Piranha II: The Spawning look like it belongs on the American Film Institute List.	**Negative** → **Postive**
The story **ultimately** [**eventually**] takes hold and grips hard.	**Positive** → **Negative**
It's weird, wonderful, and not **neccessarily** [**definitely**] for kids.	**Negative** → **Positive**.

Table 9: Demonstrates adversarial examples generated after attacking BERT on classification task. The actual word is highlighted green and substituted word is in square brackets colored red. Prediction shows before and after labels marked green and red respectively.

Examples	Prediction
Premise: If we travel for 90 minutes, we could **arrive** **[reach]** arrive at larger ski resorts. **Hypothesis**: Larger ski resorts are 90 minutes away.	**Entailment** → **Neutral**
Premise: I should put it in this **way** **[manner]**. **Hypothesis**: I'm not explaining it.	**Entailment** → **Neutral**
Premise: **Basically** **[Crucially]**, to sell myself. **Hypothesis**: Selling myself is a very important thing.	**Contradict** → **Neutral**
Premise: **June** **[April]** 21, 1995, provides the specific requirements for assessing and reporting on controls. **Hypothesis**: There are specific requirements for assessment of legal services.	**Contradict** → **Entail**

Table 10: Demonstrates adversarial examples generated after attacking BERT on entailment task. The actual word is highlighted green and substituted word is in square brackets colored red. Prediction shows before and after labels marked green and red respectively.