Записки научных
семинаров ПОМИ
Том 530, 2023 г.

**T. Ter-Hovhannisyan, H. Aleksanyan, K. Avetisyan**

# ADVERSARIAL ATTACKS ON LANGUAGE MODELS: WORDPIECE FILTRATION AND CHATGPT SYNONYMS

Abstract. Adversarial attacks on text have gained significant attention in recent years due to their potential to undermine the reliability of NLP models. We present novel black-box character- and word-level adversarial example generation approaches applicable to BERT-based models. The character-level approach is based on the idea of adding natural typos into a word according to its WordPiece tokenization. As for word-level approaches, we present three techniques that make use of synonymous substitute words created by ChatGPT and post-corrected to be in the appropriate grammatical form for the given context. Additionally, we try to minimize the perturbation rate taking into account the damage that each perturbation does to the model. By combining character-level approaches, word-level approaches, and the perturbation rate minimization technique, we achieve a state of the art attack rate. Our best approach works 30-65% faster than the previously best method, Tampers, and has a comparable perturbation rate. At the same time, proposed perturbations retain the semantic similarity between the original and adversarial examples and achieve a relatively low value of Levenshtein distance.

## §1. Introduction

In recent years, adversarial attacks have gained growing attention as a technique for exploring the robustness and reliability of machine learning models. These attacks perform small perturbations to input data that can lead models to make incorrect predictions [8, 12]. The purpose of an adversarial attack is to generate data examples while keeping it similar to the original but containing perturbations imperceptible to the human that can mislead the model [3].

*Key words and phrases*: adversarial attacks, character-level attacks, word-level attacks, ChatGPT synonyms, WordPiece.

Creating adversarial examples for text data, which is discrete, is more challenging than for continuous data such as images. For images, imperceptible perturbations of pixels can lead to a wrong prediction by a model. However, for text data small perturbations are noticeable and a single word replacement can significantly change the meaning of a sentence. As a result, adversarial examples for text should meet the following criteria: (1) the perturbations must be undetectable by human perception, specifically the semantic sense, (2) they must be grammatically correct and sound natural, and (3) they must mislead the model.

In this work, we propose different novel black-box adversarial examples generation approaches that surpass existing ones in terms of attack rate. We propose both character-level and word-level perturbation approaches, and additionally their combination. The described approaches are applicable to BERT-based models.

The character-level approach is based on the idea of adding natural typos into a word according to its WordPiece tokenization. According to the tokenization, the approach tries to make typos in such a way that the perturbed word's embedding is far from its original embedding. This is done by minimizing the intersection of the original word tokens and the tokens of the word with a typo.

For word-level perturbations, we have tried 3 different approaches. All of these approaches are based on ChatGPT-generated synonyms and try to replace original words with synonyms in the correct grammatical form (e.g. case, gender, number). The use of ChatGPT is due to its ability to produce synonyms for words that are not present in existing synonym dictionaries.

Additionally, we combined the character-level and word-level approaches by applying character-level perturbations over already substituted words.

The experiments were conducted on 3 benchmark datasets for the English language (*IMDB*[1], *YELP*[2] [20], *MR*[3] [15]) and a translation classification dataset (*NEG-1*[4] [22]) for the Russian language. According to the results, we have achieved a state of the art attack rate on these benchmarks.

[1]https://huggingface.co/datasets/imdb
[2]https://huggingface.co/datasets/yelp_polarity
[3]https://huggingface.co/datasets/rotten_tomatoes
[4]http://nlp.isa.ru/ru-en-text-align-corp/Negative-1

## §2. Related work

The first text adversarial attacks were performed against recurrent networks [1,13,16]. Due to the growth of the popularity of Transformer-based models, the attacks were also executed on BERT-base models that were more sustainable to the attacks than RNNs [7,11,14].

The attacking strategy in the text domain is divided into two steps: important word detection and perturbation. The important word is a word whose modification in the text can mislead the model. The first types of attacks used a gradient-based approach of deciding vulnerable places in the text. Such attacks are called *white-box*, which means that the adversary has access to target model parameters and architecture [4, 5, 8, 18]. This means that the attacker can study the inner workings of the model and use this knowledge to craft adversarial examples that can fool the model into making incorrect predictions.

Due to the complexity of calculation in the white-box setting and the inaccessibility of some models, the *black-box* approach was proposed [13,16]. These methods can be more valuable in real-life situations where the attacker has limited access to the targeted model. In this setting, the attacker has no access to the model weights but the output. The soft black-box methods can access the probabilities on the last layer of a model [7, 21]. The black-box attacks imitate the white-box attacks by approximations made from the output of the model on different examples.

For the perturbations, the approaches presented in the literature can be grouped into *word-level* and *character-level*. The word-level attacks perform replacement, insertion or deletion of whole words. The early methods of word-level attacks relied on rule-based techniques [1,16,19]. These methods use linguistic constraints like part-of-speech or named entity tagging, which are complicated to compute and cannot guarantee semantic consistency and fluency. For word replacements, some works employed synonym sets [16, 21], others like [10] used semantic vectors with special restrictions on stop-words and antonyms. Alternative methods that use BERT-based language models (LM) to generate word replacements were developed [7,14]. These methods have shown to increase the attack success rate and maintain semantic similarity.

Character-level attacks such as [2, 5, 6] use the idea of manipulating characters in words. In [18], adversarial examples were crafted with misspellings based on the keyboard layout. They can occur inadvertently due to human nature and sometimes cannot be noticed [9]. It was studied

that misspelling of important words, mostly the character replacement, can cause BERT to misclassify the examples.

Having the perturbation candidates for the important word, the next step is to decide which are better for the attack. Greedy algorithms in attacks [11, 16] maximize changes in model prediction using beam-search to select the best candidates for the replacement. The combinatorial optimization-based attacks like genetic algorithms presented in [1, 19] involve the use of fitness functions to control semantic quality. However, they are time-consuming due to the large search space.

## §3. Char-level attacks

In this section, we present a novel approach of character-level attacks on WordPiece [17] tokenization-based models. We restricted the word modification process so that the modifications look like natural typos.

First, utilizing the attacked model we define the sequence of the words-to-be-changed by calculating the importance of the words. The importance calculation algorithm was similar to the ones described in many previous works [14, 16]. Consider a sentence $\mathrm{S} = \{w_1, w_2, \ldots, w_n\}$ where $w_i$ represents the words in the example. After removing the word $w_i$, the resulting example is denoted by $\mathrm{S}\backslash w_i = \mathrm{S}\backslash\{w_i\} = \{w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_n\}$. A model to be attacked, $M_y(-)$, is used to represent the prediction score for label $y$. The importance score $I_{w_i}$ is calculated as the difference between the prediction score before and after removing the word $w_i$ from the sentence, which is formally defined as follows:

$$I_{w_i} = M_y(S) - M_y(S\backslash_{w_i}) \tag{1}$$

Therefore, importance scores are obtained for every word in the sentence. They are then used to select the most influential words in the context of the attacked model.

**3.1. Character-level modification actions.** After obtaining a sorted sequence of words to be modified, we utilized 4 modification actions that will look natural, presented in [18], with additional restrictions of not using homoglyphs and digits:

(1) Deletion — deleting a random character from the word; this imitates the process of missing a character while typing fast. (e.g. “moon” → “mon”);

(2) Insertion — inserting next to a random letter of the word a letter that is: a) located next to that letter according to the keyboard layout, b) the same letter (e.g., "moon" → "mooon");
(3) Mistype — changing a random letter of the word to a letter that is located next to that letter according to the keyboard layout (e.g. "moon" → "moom");
(4) Swap — swapping two random adjacent characters of the word (e.g., "moon" → "mono").

For the *insertion* and *mistype* modification actions we imitate the possibility of typographical errors, such as pressing the wrong key or pressing the same key twice. Additionally, the insertion at the end of the word included not only the nearest located letters of the last characters, but also the characters that are located near the *"Space"*. To make the actions look more natural, instead of using the whole alphabet, only nearby letters on the keyboard were utilized. Each word was changed only with one modification action so that we would have one typo per changed word.

**3.2. WordPiece Tokenization-Based Adversarial Examples.** Looping over the important words according to the sorted sequence, the goal was to maximize the attack chance on every step without utilizing a greedy approach. To do so, we propose the idea of utilizing the peculiarities of the WordPiece tokenization commonly used in BERT-based models. So, on each step, utilizing the modification actions described above, we get all of the possible candidates. We applied WordPiece tokenization on all of these candidates and the original word itself. Afterwards, we utilized the following 3 candidate filtration approaches.

(1) **Min Token Intersection (MTI)** — the intersection between tokens of the original word and each candidate is calculated. Only the candidates that have a minimal intersection with the tokens of the original word are left. E.g. for the word "melodrama" → ['mel', '##od', '##rama'] we leave "melodarma" → ['mel', '##oda', '##rma'], instead of "melodramas" → ['mel', '##od', '##rama', '##s'].
(2) **Max Token Count Distance (MTCD)** — the token count distance between the original word and the candidates is calculated. Only the words with the maximum value of the distance are left. E.g. in this case for the word "melodrama" → ['mel', '##od', '##rama'] we leave "nelodrama" → ['ne', '##lo',

'##dra', '##ma'], instead of "melodarma" → ['mel', '##oda', '##rma'].

(3) **Min Token Intersection + Max Token Count Distance (MTI + MTCD)** — the first two approaches are combined together so that the candidates with minimal intersection and maximum token count distance are left. E.g. for the word "melodrama" → ['mel', '##od', '##rama'] from the candidates "melodramas" → ['mel', '##od', '##rama', '##s'] and "melodrma" → ['mel', '##od', '##rma'] both remain after the intersection, but after calculating max distance only "melodramas" remains.

The underlying assumption of this approach is that a larger difference between subword sets of the original word and its replacement candidate will result in an even more semantically distant sentence.

After the candidates are filtered, we choose the candidate, from the remaining ones, that has the most effect on the model and change the original word with it. Then, the algorithm proceeds to the next iteration (word).

## §4. Word-level attacks

To make semantically more relevant perturbations, many existing approaches are based on utilizing various dictionaries of synonyms (WordNet, BabelNet, HowNet, etc.) or word embedding similarity. The problem with word embedding similarity is that the candidates could appear semantically not close enough to the original word. The problem with dictionaries is that there could be some important words that will not be present in the dictionaries. To circumvent these problems, the perturbations were processed by applying word synonyms from ChatGPT. Utilizing ChatGPT opens opportunities to receive synonyms for the words not included in the existing dictionaries (sometimes absurd or grammatically incorrect words). E.g. for the word "oooooh", that most probably is not present in any existing synonym dictionaries, ChatGPT produced the following synonyms: "ohh", "ahhh", "ooh", "woo". It is also able to produce synonyms for named entities such as human names, e.g., for the word "Marianne" synonyms like "Mary", "Maria", "Marie" were provided.

**4.1. Prompt Generation.** To utilize ChatGPT efficiently, the model was requested to generate synonyms for multiple words at once. The prompt (illustrated in Fig. 1) was a simple command followed by a list of

```
Generate multiple synonyms for the following words(set the
synonyms on the same line with the corresponding word):
  1. he
  2. threw(verb)
  3. into
  4. the
  5. wastebasket(noun)
  6. letter(noun)
```

Figure 1. ChatGPT prompt for generating multiple synonyms for given English words.

words, for which synonyms were required. To construct an effective prompt for getting the correct synonyms of a polysemous word from ChatGPT, we additionally determined the part of speech tag of the word using Python's Stanza library and appended the word with that tag in brackets. This technique was applied only for words in noun, verb, adverb, adjective, interjection, and proper noun part-of-speech categories. ChatGPT's response included multiple synonyms for each of the given words.

**4.2. Word Replacement Strategies.** While replacing the words, we want the perturbations to be as natural as possible keeping their morphological features (e.g., case, gender etc.). To achieve that goal, we conducted a comparison of different word replacement strategies.

4.2.1. *Morphological Analysis and Inflection (MAI)..* In this strategy, we try to imitate the use of standard dictionaries of synonyms. First, we obtain the lemma of the synonym when perturbing the text. Then, we inflect it using morphological analyzers and replace the original word with the inflected synonym.

So, within the scope of this strategy, the given text examples were initially lemmatized. The ChatGPT prompt was generated utilizing these lemmatized words. Then, as a baseline solution for keeping the morphological features while replacing the words, we utilized morphological analyzers. For each original word, their morphological features were extracted. The extraction was processed via spaCy for English words and pymorphy2 for Russian words. These features were used to inflect the synonyms obtained for the lemmatized original words and therefore were also in lemmatized form. The inflection was processed via spaCy's lemminflect extension for

English words and pymorphy2 for Russian words. The inflected synonyms were then used as the ones to conduct the perturbation.

4.2.2. *BERT-based mask prediction (BERT MP)..* Another experimental approach for keeping the morphological features is based on masked token prediction. Getting the synonyms from ChatGPT in their lemmatized form we try to predict their endings. The prediction is processed in the following steps.

(1) The original word is replaced with its synonym (which is a lemmatized word).
(2) The synonym is processed with the WordPiece tokenization.
(3) Based on the output of the tokenization the word ending is mask-predicted in two different ways.
    (a) If the synonym is tokenized into more than one token its last token is masked and predicted.
    (b) If the synonym is tokenized as one token, the ending prediction is processed iteratively. On the first iteration, a *"[MASK]"* token is added at the end of the word and predicted. Starting from the second iteration the last letters are removed one by one and the *"[MASK]"* token is attached at the end of the truncated word. The overall process of prediction process is stopped at the fifth iteration. Having the probabilities of each prediction on each iteration the ending with the highest probability is taken.

This process is shown in Fig. 2.

4.2.3. *ChatGPT Morphologically Inflected Form-Saved Synonyms (MAI FSS).* By modifying the prompt given to ChatGPT, we can make it generate synonyms for the words in unlemmatized forms while preserving the morphological features for the synonyms that it provides. The prompt for Russian examples is shown in Fig. 3. We utilized this feature to directly obtain the synonyms with proper endings. Nevertheless, for some words there appeared wrong predictions where the synonyms were provided in their lemmatized form. Such synonyms were additionally inflected the same way we did in the ***MAI*** word replacement approach.

Once we receive the list of synonyms utilizing one of the described methods, we supplement that list. For each of the synonyms, we generate their typoed versions the same way we have done it for character-level

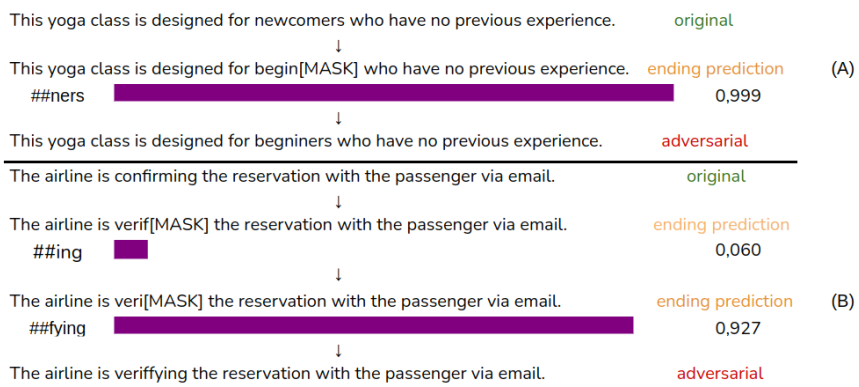


Figure 2. BERT-based mask prediction: (A) the synonym is tokenized into more than one token, (B) the synonym is tokenized into one token.

```
Provide synonyms for the following words in Russian, keeping
the same grammatical form (e.g. case, gender, number). Set the
synonyms on the same line with the corresponding word. For
example, if the word is 'города', the answer could be 'города
- местности, поселения, мегаполиса'.
  1. солдаты(noun)
  2. грелись(verb)
  3. около(adverb)
  4. костра(noun)
```

Figure 3. ChatGPT prompt for generating multiple synonyms for given Russian words.

approaches. The typoed synonyms are added to the list of untouched ones making up the final candidates list.

On each step, we choose the candidate that has the most effect on the model and replace the original word with it.

## §5. Perturbation minimization

Initially, as has been done in many other approaches, we one by one modified important words until a successful attack. If more than 40% of

the example words were modified we considered the attack unsuccessful, otherwise the attack was considered successful.

For successful attacks, after the perturbation step we tried to minimize the perturbation rate. To do so, we calculated the damage that each perturbation gives on each step. Starting from the word with the least damage we change them back to the original words and try to attack again. If the attack is unsuccessful we leave the modification of the word and iterate to the next important word. If the attack is still successful after bringing a modified word back to its original we leave the original word and proceed to the next important word.

The pseudocode of this process is given in Algorithm 1.

## §6. Attacked datasets

We study the effectiveness of our adversarial attacks on two binary classification tasks: sentiment analysis, and translation classification.

As for the sentiment analysis we utilized 3 datasets that are examined in various works: *IMDB*, *MR*, *YELP*.

As a translation classification dataset, we utilized *NEG-1*, where the task is to determine whether the first sentence in Russian is a translation of the second one in English. This dataset is used for evaluating the accuracy of detailed analysis step in cross-lingual plagiarism detection task. As our main field of study is cross-lingual plagiarism detection, we were interested in testing the effect of adversarial attacks on the BERT-based detailed analysis translation classification model. Taking into account the specifics of the plagiarism detection task, it is more important to make the system predict plagiarised sentences as not plagiarised than vice versa while generating adversarial attacks. So, for this dataset, we utilized only the positive-labeled sentence pairs of it.

## §7. Results

Within this work, all the experiments, including the comparison between existing approaches and presented ones, were conducted on 1000 examples per described dataset. For each method, we computed the following metrics.

(1) Attack Rate (AR) — the ratio of the number of attacked examples and the number of correct predicted examples.

**Algorithm 1** Perturbation Minimization

**Input:** *originalSentence*, *attackedSentence*, *replacements*
**Output:** *attackedSentence* with minimized perturbation
*originalSentence:* original sentence from dataset
*attackedSentence:* changed sentence that is misleading the model
*replacements:* information about all replacements (word and its replacement)

```
1: procedure MinimizePerturbation
2:     damage = [ ]
3:     tempSentence = originalSentence
4:     for each word in replacements do
5:         score = predict(tempSentence)
6:         tempSentence = replace(tempSentence, word)
7:         damage ← |predict(tempSentence) - score|
8:     Sort replacements by damage
9:     newReplacements = [ ]
10:    while replacements ≠ newReplacements do
11:        newReplacements = replacements
12:        replacements = Reverse(replacements)
13:        for each word in replacements do
14:            tempSentence = restoreOriginalWord(attackedSentence, word)
15:            if isAttacked(tempSentence) then
16:                attackedSentence = tempSentence
17:                remove word from replacements
18:        return attackedSentence
```

(2) Perturbation Rate (PR) — the percentage of modified words in the adversarial example.
(3) Levenstein distance (Lev.) — Levenstein distance by characters between original and adversarial example.
(4) Semantic similarity — semantic similarity between original and adversarial example (we used USE model for English and MUSE model for Russian).

We separately present the results of character-level, word-level, and combined approaches.

Table 1. The average values (attack rate (AR), perturbation rate (PR), semantic similarity (USE), Levenstein distance (Lev.), and 4 types of character-level word modifications) of our character-level attack methods and published method across MR, IMDB, and YELP datasets.

| | AR | PR | Lev. | Insertion | Deletion | Mistype | Swap |
|---|---|---|---|---|---|---|---|
| MTI | **88.24%** | **6.55%** | **5.53** | 47.25% | 13.86% | 31.80% | 7.10% |
| MTCD | 66.44% | 8.26% | 9.08 | 83.51% | 3.02% | 9.88% | 3.60% |
| MTI + MTCD | 66.36% | 8.14% | 8.99 | 81.09% | 3.86% | 10.56% | 4.67% |
| DeepWordBug | 71.63% | 12.76% | 8.41 | - | - | - | - |

7.0.1. *Character-level approaches.* We compared the proposed approaches with the state of the art character-level DeepWordBug method. Experimental results are shown in Table 1 and denote the average metrics on three English datasets. According to the results, the "*Minimum Token Intersection*" achieves the highest results significantly surpassing the DeepWordBug method in terms of all metrics. Our approach also achieves higher results than many other word-level approaches, having significantly lower values of the Levenstein distance and natural types of typos. Additionally, we present the statistics of the percent of each typos generation action that was utilized while generating the adversarial examples.

The semantic similarity was not calculated for character-level approaches as we assume that the typos are natural and the adversarial and original samples are semantically similar.

7.0.2. *Word-level approaches.* For word-level approaches, the average results over the 3 English datasets are shown in Table 2, alongside the percentage of which types of candidates were utilized during the attack process.

According to the results, these methods achieve a higher attack rate and lower perturbation rate than many other existing approaches (Table 4). The perturbation is mainly processed utilizing synonyms with typos.

7.0.3. *Combined approaches.* To achieve the highest results we combined both character- and word-level approaches, obtaining a list of candidates that contains original words with typos, synonyms, and synonyms with typos. The results of these methods are shown in Table 3, separately for each dataset. Additionally, we present the average percentage of which types of candidates were utilized during the attack process. The results

Table 2. The average values (AR, PR, USE, and Levenstein distance, the percentage of words modified using synonyms (%syns), and typos on synonyms (%syns*)) of our word-Level attack methods across MR, IMDB, and YELP datasets.

| | AR | PR | USE | Lev. | %syns | %syns* |
|---|---|---|---|---|---|---|
| BERT MP | 92.87% | 6.11% | **0.95** | 28.37 | 11.89% | 88.11% |
| MAI | **93.95%** | **5.98%** | **0.95** | **26.33** | 8.92% | 91.08% |
| MAI FSS | 92.35% | 6.51% | 0.94 | 34.82 | 9.87% | 90.13% |

Table 3. Evaluation metrics for combined attacks on all datasets, including AR, PR, USE, and Levenstein distance between the original and adversarial samples, and the percentage of words modified using synonyms (%syns), typos (%typos), and typos on synonyms (%syns*).

| | | AR | PR | USE | Lev. | %syns | %syns* | %typos |
|---|---|---|---|---|---|---|---|---|
| MR | Comb. BERT MP | **93.10%** | 9.99% | 0.89 | 7.18 | 7.33% | 44.98% | 47.69% |
| | Comb. MAI | 92.76% | **9.93%** | **0.90** | **7.03** | 4.27% | 49.61% | 46.12% |
| | Comb. MAI FSS | **93.10%** | 10.15% | 0.89 | 7.65 | 5.28% | 45.31% | 49.41% |
| IMDB | Comb. BERT MP | **98.80%** | 2.77% | **0.98** | 24.38 | 8.75% | 52.80% | 38.45% |
| | Comb. MAI | **98.80%** | **2.60%** | **0.98** | **23.90** | 6.43% | 59.86% | 33.71% |
| | Comb. MAI FSS | 97.90% | 2.70% | **0.98** | 24.43 | 5.75% | 53.56% | 40.70% |
| YELP | Comb. BERT MP | 96.92% | 5.15% | **0.96** | 24.38 | 6.53% | 56.40% | 37.07% |
| | Comb. MAI | **97.02%** | **4.96%** | **0.96** | **24.06** | 6.73% | 61.73% | 31.54% |
| | Comb. MAI FSS | 96.61% | 5.25% | **0.96** | 25.07 | 4.88% | 54.57% | 40.56% |
| NEG-1 | Comb. BERT MP | 93.54% | **13.01%** | **0.93** | **12.93** | 5.92% | 52.53% | 41.55% |
| | Comb. MAI | **93.89%** | 13.07% | **0.93** | 13.30 | 5.12% | 55.23% | 39.64% |
| | Comb. MAI FSS | 93.74% | **13.01%** | 0.92 | 14.57 | 5.60% | 52.80% | 41.60% |

show that mainly the typos-affected synonyms are used. The explanation is that typos-affected synonyms are most perturbed relative to the original words. At the same time, we achieve a high usage percentage of typos-affected original words, which significantly decreases the Levenstein distance making the adversarial example more imperceptible.

In terms of the Russian dataset, the results show that the method is multilingually applicable achieving nearly 94% attack rate.

The comparison results of the two best combined approaches (*BERT MP, MAI*) with other existing ones are shown in Table 4. The comparison was processed over the averaged results on 3 considered datasets.

Table 4. The average[variance] of values (AR, PR, USE, and Levenstein distance) of our combined attack methods and the published methods across MR, IMDB, and YELP datasets.

| | BERT MP | MAI | PWWS | TextFooler | Tampers | Bert-attack | TextBugger |
|---|---|---|---|---|---|---|---|
| AR | **96.27%** [5.63%] | 96.19% [6.42%] | 91.31% [53.06%] | 94.56% [15.98%] | 95.35% [9.70%] | 83.06% [12.35%] | 78.62% [130.23%] |
| PR | 5.97% [9.02%] | 5.83% [9.33%] | 8.79% [18.15%] | 12.65% [24.24%] | **4.43%** [6.01%] | 10.56% [11.59%] | 22.44% [28.49%] |
| USE | **0.95** | **0.95** | 0.92 | 0.90 | 0.93 | 0.88 | 0.93 |
| Lev. | **18.65** | 18.33 | 33.93 | 52.71 | 35.25 | 118.29 | 35.48 |

Table 5. Runtime comparison per sample between our fastest method MAI and Tampers.

| | MR | IMDB | YELP |
|---|---|---|---|
| Tampers | 9,5 sec./sample | 98,5 sec./sample | 79,6 sec./sample |
| Comb. MAI | **6,8 sec./sample** | **41,0 sec./sample** | **27,7 sec./sample** |

According to the results, the proposed approaches achieve state-of-the-art results in terms of attack rate simultaneously surpassing the best methods in terms of semantic similarity, and the Levenstein distance.

We separately computed the time consumption (Table 5) of our fastest approach with the *Tampers* that achieves a high attack rate with the least perturbation rate. The time consumption was computed on one RTX 3090. Our approach showed from 30 to 65 per cent faster results.

## §8. Conclusion

In this work, we have presented novel character- and word-level approaches that are based on WordPiece tokenization and ChatGPT-generated synonyms respectively. The proposed character-level approach achieves significantly higher results than the other state of the art character-level approach. Our character-level method is also comparable to other word-level approaches but with much lower Levenstein distance and natural types of typos. Combining both approaches, we have developed an adversarial example generation method that achieves state of the art results in terms of attack rate reaching over 96%, and additionally surpassing the best methods in terms of semantic similarity and the Levenstein distance.

## References


1. M. Alzantot, Y. Sharma, A. Elgohary, B.-J. Ho, M. Srivastava, and K.-W. Chang, *Generating Natural Language Adversarial Examples*, arxiv (2018).
2. Y. Belinkov and Y. Bisk, *Synthetic and Natural Noise Both Break Neural Machine Translation*, arxiv (2018).
3. A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, *Adversarial Attacks and Defences: A Survey*, arxiv (2018).
4. J. Ebrahimi, D. Lowd, and D. Dou, *On Adversarial Examples for Character-Level Neural Machine Translation*, arxiv (2018).
5. J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, *HotFlip: White-Box Adversarial Examples for Text Classification*, arxiv (2018).
6. J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi, *Black-box Generation of Adversarial Text Sequences to Evade Deep Learning Classifiers*, arxiv (2018).
7. S. Garg and G. Ramakrishnan, *BAE: BERT-based Adversarial Examples for Text Classification*, Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, 2020, pp. 6174–6181.
8. I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and Harnessing Adversarial Examples*, arxiv (2015).
9. J. Grainger and C. Whitney, *Does the huamn mnid raed wrods as a wlohe?*, 58–9.
10. R. Jia, A. Raghunathan, K. Göksel, and P. Liang, *Certified Robustness to Adversarial Word Substitutions*, arxiv (2019).
11. D. Jin, Z. Jin, J. T. Zhou, and P. Szolovits, *Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment*, arxiv (2020).
12. A. Kurakin, I. Goodfellow, and S. Bengio, *Adversarial examples in the physical world*, arxiv (2017).
13. J. Li, S. Ji, T. Du, B. Li, and T. Wang, *TextBugger: Generating Adversarial Text Against Real-world Applications*, Proceedings 2019 Network and Distributed System Security Symposium, 2019.
14. L. Li, R. Ma, Q. Guo, X. Xue, and X. Qiu, *BERT-ATTACK: Adversarial Attack Against BERT Using BERT*, arxiv (2020).
15. B. Pang and L. Lee, *Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales*, , ACL '05, Association for Computational Linguistics, 2005, pp. 115–124.
16. S. Ren, Y. Deng, K. He, and W. Che, *Generating Natural Language Adversarial Examples through Probability Weighted Word Saliency*, Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2019, pp. 1085–1097.
17. X. Song, A. Salcianu, Y. Song, D. Dopson, and D. Zhou, *Fast WordPiece Tokenization*, arxiv (2021).
18. L. Sun, K. Hashimoto, W. Yin, A. Asai, J. Li, P. Yu, and C. Xiong, *Adv-BERT: BERT is not robust on misspellings! Generating nature adversarial samples on BERT*, arxiv (2020).

19. Y. Zang, F. Qi, C. Yang, Z. Liu, M. Zhang, Q. Liu, and M. Sun, *Word-level Textual Adversarial Attacking as Combinatorial Optimization*, Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2020, pp. 6066–6080.
20. X. Zhang, J. Zhao, and Y. LeCun, *Character-level Convolutional Networks for Text Classification*, arxiv (2016).
21. X. Zhao, L. Zhang, D. Xu, and S. Yuan, *Generating Textual Adversaries with Minimal Perturbation*, arxiv (2022).
22. V. Zubarev and V. Sochenkov, *Cross-Language text alignment for plagiarism detection based on contextual and context-free models*, 2019.

Russian-Armenian University,
ISP RAS, Yerevan, Armenia
*E-mail*: tterhovhannisyan@ispras.ru

*E-mail*: a.aleksanyan@ispras.ru

*E-mail*: karavet@ispras.ru

Поступило 6 сентября 2023 г.