Summer Project Report: PGD Attack Modification on Text Classifier

Xuanzhou Chen
Department of Electrical & Computer Engineering
New York University
Brooklyn, NY 11201
xc2425@nyu.edu

Abstract

In this summer project report, a modified PGD attack is proposed based on the original paper by Sadrizadeh et al. [2022]. Using the K-means and our proposed "bucket" searching algorithm, we designed a novel PGD attack, which is significantly accelerated with less time complexity compared to the original work. In our experiments, The modified PGD attack algorithm is deployed on the original datasets (YELP reviews) to attack the language model, GPT-2. The performances of both the modified and original PGD attack are compared with respect to their total attack time, text semantic similarity and attack successful rate. Our results show that the modified PGD attack time is drastically decreased to about 10% of the baseline, whereas the semantic similarity and attack successful rate are slightly reduced nearly by 3%. Colab Link: https://colab.research.google.com/drive/1mkb5pSuydSddhRCsNmgyhDCNd0jJZ9WC#scrollTo=YFgyV_xIVl_L

1 Introduction

In recent years, Transformer as a powerful machine learning language model is becoming increasingly prevalent in the applications of natural language processing (NLP). In the mean time, however, these transformer-based language models are vulnerable with respect to both inevitable perturbations and intentionally malicious attacks in the sense that they can be misled to give wrong predictions. To tackle this model deficiency, there is growing concern by NLP and security researchers to enhance the adversarial robustness of the language models.

Designing a text attack algorithm is the first crucial step. One accomplished text attack method is the Projected Gradient Descent (PGD) attack, where the attacker has access to the model gradients. Due to the discrete property of the text data, researchers cannot directly apply the gradient descent and optimization during the neural network training of language models. The PGD attack method solves a challenging point of generating adversarial examples in textual applications by using projection. Sadrizadeh et al. [2022] proposed a PGD attack to fool transformer-based text classifiers. The authors first used a block sparsity constraint to generate small perturbations on original text inputs for the optimization problem setting, then applied gradient descent algorithm to find the minimizer in a expended continuous space, and eventually projected the minimizer to the optimal adversarial example in the discrete space.

Based on Sadrizadeh et al. [2022], we improved the text attack algorithm by reducing time complexity for searching the optimal adversarial example in the projection step. The main contribution is summarized as follows:
1. Designed a k-means combined "bucket" searching method for the optimal adversarial example in the projection step of the original PGD attack algorithm.
2. Reduced the time complexity of the original PGD attack by adopting our designed "bucket"

Preprint. Under review.

searching.
3. Conducted experiments for both original and modified attack on the YELP Review dataset to attack GPT-2 pretrained model in text classification.

The rest of report is organized as follows. In Section 2, the original attack algorithm is demonstrated and time complexity is analyzed. In section 3, our searching algorithm and a new projection step are described. New algorithm is evaluated on the YELP Review dataset against pretrained GPT-2 and compared to the baseline in Section 4. Finally, some conclusions and furture work are presented in Section 5.

2 Problem Formulation

In this section, we will first describe the original PGD attack algorithm and then analyze the time complexity.

2.1 Original PGD Attack

2.1.1 Optimization Formulation

For the input embedding, suppose there are n input tokens $x_1, ..., x_n$ for one sentence in the corpus, the absolute positional embedding is used to convert these tokens to a sequence of n continuous vectors, $\mathbf{e_x} = [emb(x_1), emb(x_2), ..., emb(x_n)]$.

First of all, we need to define a sequence of n perturbation vectors $\mathbf{r}_x = [\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_n]$. Then we merge the perturbation constraint with the adversarial loss to compute the total loss function to formulate the optimization problem. The loss function is thus composed of two parts, the adversarial loss $\mathcal{L}_{Adv}$ and the perturbation loss $\mathcal{L}_{BSparse}$. $\mathcal{L}_{Adv}$ is defined as the negative of the cross entropy: $\mathcal{L}_{Adv} = -\mathcal{L}_f(\mathbf{e_{x'}}, y)$, where $\mathcal{L}_f$ is the loss function of the model when the input is the adversarial example $\mathbf{e_{x'}}$ and the ground-truth class is y. The adversarial loss $\mathcal{L}_{Adv}$ is to fool the model with an untargeted attack. The perturbation loss $\mathcal{L}_{BSparse}$ is designed for imposing the sparsity limits to the perturbation, where it is defined as $\mathcal{L}_{BSparse} = \sum_{i=1}^{n} ||\mathbf{r}_i||_2$. Here, the $l1$ summation of $l2$ norm is applied. Therefore, the objective to the optimization for seeking an adversarial input which has the sparse perturbation can be stated as follows:

$$\hat{\mathbf{e}_{\mathbf{x}'}} = \text{argmin}_{\mathbf{e}_{\mathbf{x}'} \in \mathcal{E}_\mathcal{V}} \mathcal{L}_{Adv} + \alpha \mathcal{L}_{BSparse}$$

where $\mathcal{E}_\mathcal{V}$ is the discrete subspace of every token of the vocabulary set $\mathcal{V}$ in the embedding space. In addition, α is the hyper-parameter representing the relative importance of the perturbation loss.

2.1.2 Attack Algorithm

To solve the above optimization problem, Sadrizadeh et al. [2022] first expand the discrete embedding space $\mathcal{E}_\mathcal{V}$ into a continuous embedding space $\mathcal{E}$. For each iteration, the first step is to use the gradient descent algorithm to find the minimizer $\mathbf{e_g}$ in the space $\mathcal{E}$, the second step is to project the minimizer to the adversarial example $\mathbf{e_p}$ in $\mathcal{E}_\mathcal{V}$. Iteration terminates when either the adversarial example is misclassified by the victim model or the iteration times exceed the maximum number of iterations. The detailed attack algorithm is shown in Figure 1 & 2.

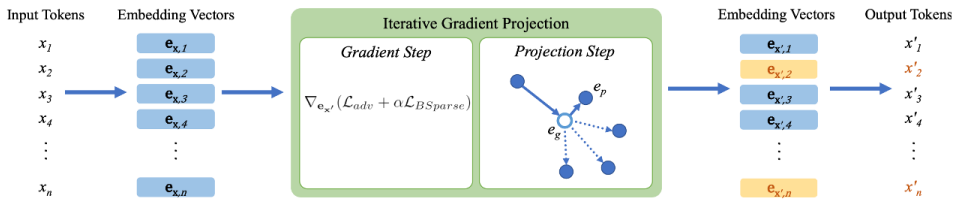

Fig. 1. Block diagram of the proposed method.

Figure 1: Original PGD Attack

Algorithm 1 Block-Sparse Adversarial Attack

Input:
$f(.)$: Target classifier model, $\mathcal{V}$: Vocabulary set
$\mathbf{x}$: Tokenized input sentence, lr: Learning rate
A: Set of decreasing values for Hyper-parameter α to control the importance of the block-sparsity term
K: Maximum number of iterations
Output:
$\mathbf{x}'$: Generated adversarial example
procedure
initialization:
buffer $\leftarrow$ empty, $y \leftarrow f(\mathbf{x})$, $k \leftarrow 0$
$\forall i \in \{1, ...n\}$ $\mathbf{e_{g},}_i \leftarrow \text{emb}(x_i)$
for α in A **do**
while $f(\mathbf{e_p}) = y$ and $k \leq K$ **do**
$k \leftarrow k + 1$
Step 1: Gradient descent in the continuous embedding space:
$\mathbf{e_g} \leftarrow \mathbf{e_g} - lr.\nabla_{\mathbf{e_{x'}}}(\mathcal{L}_{adv} + \alpha\mathcal{L}_{BSparse})$
Step 2: Projection to the discrete subspace $\mathcal{E}_\mathcal{V}$ and update if the sentence is new:
for $i \in \{1, ..., n\}$ **do**
$\mathbf{e_{p},}_i \leftarrow \underset{\mathbf{e} \in \mathcal{E}_\mathcal{V}}{\text{argmin}} \frac{\mathbf{e}^\top \mathbf{e_{g},}_i}{\|\mathbf{e}\|_2 . \|\mathbf{e_{g},}_i\|_2}$
end for
if $\mathbf{e_p}$ not in **buffer then**
add $\mathbf{e_p}$ to **buffer**
$\mathbf{e_g} \leftarrow \mathbf{e_p}$
end if
end while
if $f(\mathbf{e_p}) \neq y$ **then**
break (adversarial example is found)
end if
end for
return $\mathbf{e_{x'}} \leftarrow \mathbf{e_p}$
end procedure

Reduce Time Complexity

hyper-parameter that determines the relative importance of the block-sparsity term.

Figure 2: Searching Algorithm Modifying Projection Step (Step 2)

2.2 Time Complexity Analysis

Suppose there are n tokens in one input sentence and we denote the vocabulary set as $\mathcal{V}$.Then the for loop in the projection step (Step 2 in Figure 2) of original PGD attack has the time complexity of $\Theta(|\mathcal{V}|n)$. If we have M sentences in the corpus, and the sentence i has n_i tokens, then the time complexity $T(n) = \sum_{i=1}^{M} |\mathcal{V}|n_i$.

3 Proposed Method

3.1 Preprocessing

3.1.1 Preprocessing Without K-means

Inspired by the idea of hashing, we design the "bucket" preprocessing method to speed-up the searching in the Step 2 of the PGD attack in the original paper. The "bucket" preprocessing is described as follows:

- First, we build K "buckets" (i.e. vectors with shape of $1 \times dim$).
- Second, we calculate the cosine similarity between each embedding vector and each "bucket" vector and assign each embedding vector to the bucket with the highest cosine similarity.
- Third, we compute the average vectors of all embedding vectors for each "bucket".

Note that after "bucket" preprocessing, we may have less than K buckets with some embedding vectors in them.

3.1.2 Preprocessing with K-means

In order to accelerate the searching process (in Step 2) to a greater extent, we apply the K-means algorithm to preprocess the embedding vectors. The main idea is concluded as follows. Here, we proposed this K-means preprocessing method as Iterative K-means preprocessing (IKMP).

- First, we build K "buckets" (i.e. the clustroids).
- Second, we map each embedding vector into a particular bucket by calculating the cosine similarity.
- Third, we compute the average vectors of all embedding vectors for each "bucket", so the embedding vectors will be remapped to a new "bucket" by recomputing the similarity.
- Fourth, repeat the third step until those average vectors converge for all "buckets".

Note that the number of "buckets" in each iteration decreases and will converge to a certain number $M \leq k$

3.2 Performing Attack

When performing the PGD attack, given the minimizer we find by gradient descent algorithm from Step 1 for each embedding vector i, the minimizer will be firstly matched to the closest "bucket" whose average vector has the highest cosine similarity with the minimizer. Then the minimizer will be projected to the closest embedding vector which has the highest cosine similarity with it. Finally, the closest embedding vector will be plugged in the adversarial example to replace the corresponding original token embedding vector.

3.3 Time Complexity Analysis

It is trivial to see that the total time complxity is equal to the addition of the K-means preprocessing time complexity and attacking time complexity. We have the total time complexity $T = O((K + 1) \times |\mathcal{V}| \times \texttt{maxIter} + \sum_i^{\texttt{maxIter}} N_i \times \texttt{dim}) + (N + \frac{|\mathcal{V}|}{N})$, where N_i is the number of "buckets" in i-th iteration and N is the final number of "buckets".

3.3.1 Preprocessing Time Complexity

Suppose we use $K >= 1$ as the number of "buckets" (i.e., centroids), $|\mathcal{V}|$ as the vocabulary size, `dim` as the number of embedding dimensions, and maxIter as the maximum number of iteration. N_i to denote the length of the buckets in i-th iteration. Time complexity is computed as follows: $T \leq K \times |\mathcal{V}| \times \texttt{maxIter} + |\mathcal{V}| \times \texttt{maxIter} + \sum_i^{\texttt{maxIter}} N_i \times \texttt{dim}$. Hence, we have total time complexity of the iterative K-means preprocessing with $O((K + 1) \times |\mathcal{V}| \times \texttt{maxIter} + \sum_i^{\texttt{maxIter}} N_i \times \texttt{dim})$.

From the upper bound shown above, the time complexity of iterative K-means preprocessing is then only determined by the number of initial "buckets"/centroids K, the max iteration number `maxIter`, and the number of "buckets" in each iteration. It also depends the vocabulary size and the number of embedding dimensions, but these two factors are already predefined by the language model.

Algorithm 1 Iterative K-means Preprocessing (IKMP)

Input: maxIter - max iteration number; $|\mathcal{V}|$ - total number of vector embeddings $\vec{e}_i \in \mathcal{E}_{\mathcal{V}}$, $i = 1, 2, ..., |V|$; K - initial number of clustroids

Output: centroids $\leftarrow [\vec{v}_1, ..., \vec{v}_K]$; buckets $\leftarrow$ dict{'idx': [],...}

$iter \leftarrow 0$
centroids $\leftarrow [\vec{v}_1, ..., \vec{v}_K]$ ▷ Initially randomized K centroids
while $iter <$ maxIter **do**
    buckets $\leftarrow$ dict{} ▷ Empty dictionary
    for i in $1, ..., |\mathcal{V}|$ **do**
        for j in $1, ..., K$ **do**
            idx $\leftarrow argmin_j \frac{\vec{e}_i^T \cdot \vec{v}_j}{||\vec{e}_i||_2 \cdot ||\vec{v}_j||_2}$ ▷ Assign closest 'bucket' for $\vec{e}_i$
            if idx **not in** bucket.keys **then**
                buckets[idx] = list[] ▷ Add a new key 'idx' in the dict
            end if
            buckets[idx].append($\vec{e}_i$) ▷ Add $\vec{e}_i$ into 'bucket'
        end for
    end for
    centroids $\leftarrow$ list[] ▷ Empty previous centroids
    for key in `buckets.keys` **do**
        $v \leftarrow$ AvgVector(buckets[key]) ▷ Update centroids
        centroids.append(v)
    end for
    $iter = iter + 1$
    if all AvgVectors converge **then**
        `break`
    end if
end while

Note*: In the implementation of verifying the average vector convergence, we use the summation of $l2$ norm to calculate the difference between each average vectors in the previous iteration and the current iteration.

3.3.2 Attacking Time Complexity

Suppose after preprocessing, we have N final "buckets". Based on the PGD attack algorithm, the cost time is calculated as follows if we assume the embedding vectors are approximately uniformly distributed in each "buckets": $T = N + \frac{|\mathcal{V}|}{N} \geq 2\sqrt{|\mathcal{V}|}$.

4 Experimental Results

We denote the initial number as K, the number of final "buckets" as N, the attack time (s) per sample as T, the average number of vectors in the buckets as #AvgVec/bkt. AvgSim refers to the average cosine similarity. SucAvgSim refers to the average cosine similarity in the successful attacks. TER refers to the token error rate. SAR refers to the successful attack rate. The hyperparameter $\alpha = [10, 8, 5, 2]$. The experiment is conducted on 200 sentences from the YELP Review corpus.

Table 1: 'Bucket' Preprocessed PGD Attack Performance on YELP-Review Dataset

K	N	T	AvgSim	SucAvgSim	#AvgVec/bkt	FailedAttk	TER	SAR
/	/	15.742	0.601	0.904	1	1	0.098	0.995
2^{10}	868	0.145	0.423	0.943	57.90	77	0.078	0.615
2^{11}	1520	0.705	0.427	0.937	33.06	76	0.086	0.62
2^{12}	2551	0.669	0.432	0.902	19.70	68	0.115	0.66
2^{13}	3895	0.108	0.446	0.863	12.90	56	0.175	0.72
2^{14}	5878	0.093	0.441	0.902	8.55	67	0.117	0.665
2^{15}	8392	0.690	0.471	0.823	5.99	37	0.223	0.815
2^{16}	11387	0.177	0.473	0.817	4.414	32	0.219	0.84
2^{17}	14552	0.490	0.491	0.809	3.454	22	0.229	0.89
2^{18}	18021	0.374	0.506	0.820	2.789	19	0.213	0.905
2^{19}	21584	0.260	0.516	0.829	2.328	18	0.210	0.91
2^{20}	25493	1.135	0.522	0.826	1.971	14	0.189	0.93

1 Preliminary Results of Modified Attack Performance Without K-means on **YELP Review** Dataset.
2 The First Line Represents the PGD attack performance in the Original Paper

Table 2: IKMP Modified PGD Attack Performance on YELP-Review Dataset

K	N	T	AvgSim	SucAvgSim	#AvgVec/bkt	FailedAttk	TER	SAR
/	/	15.742	0.601	0.904	1	1	0.098	0.995
2^{10}	879	0.070	0.487	0.823	57.18	29	0.182	0.855
2^{11}	1494	0.110	0.496	0.835	33.64	27	0.184	0.865
2^{12}	2492	0.100	0.495	0.837	20.17	30	0.166	0.85
2^{13}	4082	0.227	0.506	0.834	12.31	24	0.162	0.88
2^{14}	5953	0.094	0.515	0.842	8.44	22	0.158	0.89
2^{15}	8380	0.536	0.535	0.859	6.00	17	0.147	0.915
2^{16}	11130	0.750	0.538	0.848	4.52	13	0.154	0.935
2^{17}	14397	0.102	0.534	0.840	3.49	13	0.164	0.935
2^{18}	18189	0.354	0.540	0.849	2.763	12	0.154	0.94
2^{19}	21765	0.562	0.546	0.841	2.31	7	0.155	0.965
2^{20}	25753	1.109	0.555	0.857	1.95	7	0.140	0.965

1 Preliminary Results of Modified Attack Performance With K-means on **YELP Review** Dataset.
2 The First Line Represents the PGD attack performance in the Original Paper

4.1 Preprocessing With/Without K-means Experiment

Figure 2: Preliminary Results

5 Conclusion

In this project, we faster the PGD attack from the original paper through our proposed "bucket" preprocessing before adversarial example(AE) searching in the projection step. Moreover, K-means algorithm is applied to further curtailed the attack time. Experiments of our new attack on YELP Review compared to the original work show that the modified PGD attack time is shortened to about 10% of the baseline with a slight drop in attack successful rate ($\sim$ 3%), meanwhile the semantic similarity is well preserved.

5.1 Future Work

There are two research directions in the future. One direction is to transfer the modified attack to other NLP task such as machine translation. Since machine translation is a white-box attack, we can perform the modified PGD attack in different positions inside the Transformer model structure. The other future direction is to design another preprocecssing method by applying dimension reduction methods other than K-means and give mathematical proof for the guarantees of the upper bound of time complexity.

References

Sahar Sadrizadeh, Ljiljana Dolamic, and Pascal Frossard. Block-sparse adversarial attack to fool transformer-based text classifiers, 2022. URL `https://arxiv.org/abs/2203.05948`.