24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

Generating Natural Language Adversarial Examples on a Large Scale with Generative Models

Yankun Ren[1], **Jianbin Lin**[1], **Siliang Tang**[*,2], **Jun Zhou**[1], **Shuang Yang**[1], **Yuan Qi**[1] and **Xiang Ren**[3]

Abstract. Today text classification models have been widely used. However, these classifiers are found to be easily fooled by adversarial examples. Fortunately, standard attacking methods generate adversarial texts in a pair-wise way, that is, an adversarial text can only be created from a real-world text by replacing a few words. In many applications, these texts are limited in numbers, therefore their corresponding adversarial examples are often not diverse enough and sometimes hard to read, thus can be easily detected by humans and cannot create chaos at a large scale. In this paper, we propose an end to end solution to efficiently generate adversarial texts from scratch using generative models, which are not restricted to perturbing the given texts. We call it unrestricted adversarial text generation. Specifically, we train a conditional variational autoencoder (VAE) with an additional adversarial loss to guide the generation of adversarial examples. Moreover, to improve the validity of adversarial texts, we utilize discrimators and the training framework of generative adversarial networks (GANs) to make adversarial texts consistent with real data. Experimental results on sentiment analysis demonstrate the scalability and efficiency of our method. It can attack text classification models with a higher success rate than existing methods, and provide acceptable quality for humans in the meantime.

1 Introduction

Today machine learning classifiers have been widely used to provide key services such as information filtering, sentiment analysis. However, recently researchers have found that these ML classifiers, even deep learning classifiers are vulnerable to adversarial attacks. They demonstrate that image classifier [10] and now even text classifier [26] can be fooled easily by adversarial examples that are deliberately crafted by attacking algorithms. Their algorithms generate adversarial examples in a pair-wise way. That is, given one input $x \in \mathcal{X}$, they aim to generate one corresponding adversarial example $x' \in \mathcal{X}$ by adding small imperceptible perturbations to x. The adversarial examples must maintain the semantics of the original inputs, that is, x' must be still classified as the same class as x by humans. On the other hand, adversarial training is shown to be a useful defense method to resist adversarial examples [31, 10]. Trained on a mixture of adversarial and clean examples, classifiers can be resistant to adversarial examples.

In the area of natural language processing (NLP), existing methods are pair-wise, thus heavily depend on input data x. If attackers want to generate adversarial texts which should be classified as a chosen class with pair-wise methods, they must first collect texts labeled as the chosen class, then transform these labeled texts to the corresponding adversarial examples by replacing a few words. As the amount of labeled data is always small, the number of generated adversarial examples is limited. These adversarial examples are often not diverse enough and sometimes hard to read, thus can be easily detected by humans. Moreover, in practice, if attackers aim to attack a public opinion monitoring system, they must collect a large number of high-quality labeled samples to generate a vast amount of adversarial examples, otherwise, they can hardly create an impact on the targeted system. Therefore, pair-wise methods only demonstrate the feasibility of the attack but cannot create chaos on a large scale.

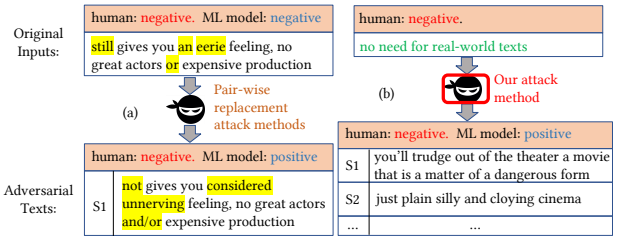


Figure 1. An illustration of adversarial text generation. (a) Given one negative text which is also classified as negative by a ML model, traditional methods replace a few words (yellow background) in the original text to get one paired adversarial text, which is still negative for humans, but the model prediction changes to positive. (b) Our unrestricted method does not need input texts. We only assign a ground-truth class - negative, then our method can generate large-scale adversarial texts. which are negative for humans, but classified as positive by the ML model.

In this paper, we propose an unrestricted end to end solution to efficiently generate adversarial texts, where adversarial examples can be generated from scratch without real-world texts and are still meaningful for humans. We argue that adversarial examples do not need to be generated by perturbing existing inputs. For example, we can generate a movie review that does not stem from any examples in the dataset at hand. If the movie review is thought to be a positive review by humans but classified as a negative review by the targeted model, the movie review is also an adversarial example. Adversarial examples generated in this way can break the limit of input number, thus we can get large scale adversarial examples. On the other hand, the proposed method can also be used to create more adversarial examples for defense. Trained with more adversarial examples often means more robustness for these key services.

The proposed method leverages a conditional variational autoencoder (VAE) to be the generator which can generate texts of a desired

[1] Ant Financial Services Group, Emails: {yankun.ryk, jianbin.ljb, jun.zhoujun, shuang.yang, yuan.qi}@anfin.com
[2] Zhejiang University, Email: siliang@zju.edu.cn
[3] University of Southern California, Email: xiangren@usc.edu
* Corresponding author.

class. To guide the generator to generate texts that mislead the targeted model, we access the targeted model in a white-box setting and use an adversarial loss to make the targeted model make a wrong prediction. In order to make the generated texts consistent with human cognition, we use discrimators and the training framework of generative adversarial networks (GANs) to make generated texts similar as real data of the desired class. After the whole model is trained, we can sample from the latent space of VAE and generate infinite adversarial examples without accessing the targeted model. The model can also transforms a given input to an adversarial one.

We evaluate the performance of our attack method on a sentiment analysis task. Experiments show the scalability of generation. The adversarial examples generated from scratch achieve a high attack success rate and have acceptable quality. As the model can generate texts only with feed-forwards in parallel, the generation speed is quite fast compared with other methods. Additional ablation studies verify the effectiveness of discrimators, and data augmentation experiments demonstrate that our method can generate large-scale adversarial examples with higher quality than other methods. When existing data at hand is limited, our method is superior over the pair-wise generation.

In summary, the major contributions of this paper are as follows:

- Unlike the existing literature in text attacks, we aim to construct adversarial examples not by transforming given texts. Instead, we train a model to generate text adversarial examples from scratch. In this way, adversarial examples are not restricted to existing inputs at hand but can be generated from scratch on a large-scale.
- We propose a novel method based on the vanilla conditional VAE. To generate adversarial examples, we incorporate an adversarial loss to guide the vanilla VAE's generation process.
- We adopt one discrimator for each class of data. When training, we train the discrimators and the conditional VAE in a min-max game like GANs, which can make generated texts more consistent with real data of the desired class.
- We conduct attack experiments on a sentiment analysis task. Experimental results show that our method is scalable and achieves a higher attack success rate at a higher speed than recent baselines. The quality of generated texts is also acceptable. Further ablation studies and data augmentation experiments verify our intuitions and demonstrate the superiority of scalable text adversarial example generation.

2 Related Work

There has been extensive studies on adversarial machine leaning, especially on deep neural models [31, 10, 16, 28, 1]. Much work focuses on image classification tasks [31, 10, 5, 11, 33]. [31] solves the attack problem as an optimization problem with a box-constrained L-BFGS. [10] proposes the fast gradient sign method (FGSM), which perturbs images with noise computed as the gradients of the inputs.

In NLP, perturbing texts is more difficult than images, because words in sentences are discrete, on which we can not directly perform gradient-based attacks like continuous image space. Most methods adapt the pair-wise methods of image attacks to text attacks. They perturb texts by replacing a few words in texts. [24, 9, 6] calculate gradients with respect to the word vectors and perturb word embedding vectors with gradients. They find the word vector nearest to the perturbed vector. In this way, the perturbed vector can be map to a discrete word to replace the original one. These methods are gradient-based replacement methods.

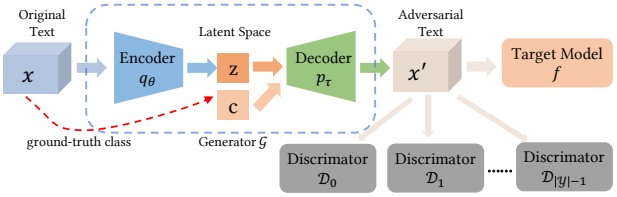


Figure 2. The architecture of the whole model. In the training phase, $\mathcal{G}$ generates an adversarial text x' to reconstruct the original text x, and feed x' to $\mathcal{D}$ and f to make f predict differently on x and x'. After trained, the model can generate large-scale adversarial texts based on sampled latent space vector z and a chosen class c without original texts x.

Other attacks on texts can be summarized as gradient-free replacement methods. They replace words in texts with typos or synonyms. [16] proposes to edit words with tricks like insertion, deletion and replacement. They choose appropriate words to replace by calculating the word frequency and the highest gradient magnitude. [15] proposes five automatic word replacement methods, and use magnitude of gradients of the word embedding vectors to choose the most important words to replace. [26] is based on synonyms substitution strategy. Authors introduce a new word replacement order determined by both the word saliency and the classification probability. However, these replacement methods still generate adversarial texts in a pair-wise way, which restrict the adversarial texts to the variants of given real-world texts. Besides, the substitute words sometimes change text meanings. Thus existing adversarial text generation methods only demonstrate the feasibility of the attack but cannot create chaos on a large scale.

In order to tackle the above problems, we propose an unrestricted end to end solution to generate diverse adversarial texts on a large scale with no need of given texts.

3 Methodology

In this section, we propose a novel method to generate adversarial texts for the text classification model on a large scale. Though trained with labeled data in a pair-wise way, after it is trained, our model can generate an unlimited number of adversarial examples without any input data. Moreover, like other traditional pair-wise generation methods, our model can also transform a given text into an adversarial one. Unlike the existing methods, our model generates adversarial texts without querying the attacked model, thus the generation procedure is quite fast.

3.1 Overview

Figure 2 illustrates the overall architecture of our model. The model has three components: a generator $\mathcal{G}$, discrimators $\mathcal{D}$, and a targeted model f. $\mathcal{G}$ and $\mathcal{D}$ form a generative adversarial network (GAN). When training, we feed an original input to the generator $\mathcal{G}$, which transforms x to an adversarial output x'. The procedure can be defined as follows:

$$\mathcal{G}(x) : x \in \mathcal{X} \rightarrow x' \tag{1}$$

$\mathcal{G}$ aims to generate x' to reconstruct x. Then, we feed the generated x' to the targeted model f, and f will classify x' as a certain class, which we hope is a wrong label. Thus we have the following equation:

$$f(x') = y_t \in \mathcal{Y} \tag{2}$$

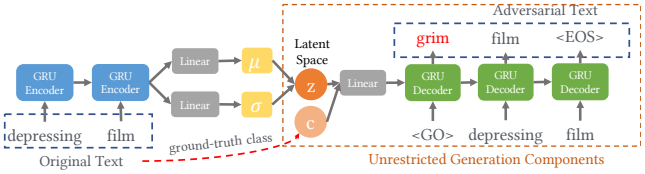


Figure 3. The generator $\mathcal{G}$. When training, we need input texts to train $\mathcal{G}$ After $\mathcal{G}$ is trained, we only need to sample z from the latent space, and use the decoder to generate adversarial texts unrestrictedly without original texts.

where $y_t \neq f(x)$ and $\mathcal{Y}$ is the label space of the targeted classification model.

In order to keep x' being classified as the same class as x by human, we add one discrimator for each class $y \in \mathcal{Y}$. With the help of the min-max training strategy of GAN framework, each class y's discrimator can make x' close to the distribution of real class y data, thus x' is made to be compatible with human congnition.

We now proceed by introducing these components in further details.

3.2 Generator

In this subsection, we describe the generator $\mathcal{G}$ for text generation. We use the variational autoencoder (VAE) [14, 27] as the generator. The VAE is a generative model based on a regularized version of the standard autoencoder. This model supposes the latent variable z is sampled from a prior distribution.

As shown in Figure 2, the VAE is composed of the encoder $q_\theta(z|x)$ and the decoder $p_\tau(x|z)$, where τ is the parameters of p and θ is the parameters of q. q_θ is a neural network. Its input is a text x, its output is a latent code z. q_θ encodes x into a latent representation space $\mathcal{Z}$, which is a lower-dimensional space than the input space. p_τ is another neural network. Its input is the code z, it outputs an adversarial text x' to the probability distribution of the input data x.

In our model, we adopt the gated recurrent unit (GRU) [7] as the encoder and the decoder. As in Figure 3, The input x is a sentence of words, we formulate the input for neural networks as follows: for a word at the position i in a sentence, we first transform it into a word vector v_i by looking up a word embedding table. The word embedding table is randomly initialized and is updated during the model training. Then the word embedding vectors are fed into the GRU encoder. In the i-th GRU cell, a hidden state h_i is emitted.

We use h_N to denote the last GRU cell's hidden state, where N is the length of the encoder input. In order to get latent code z, we feed h_N into two linear layers to get μ and σ respectively. Following the Gaussian reparameterization trick [14], we sample a random sample ε from a standard Gaussian ($\mu = \overrightarrow{0}, \sigma = \overrightarrow{1}$), and compute z as:

$$z = \mu + \sigma \circ \varepsilon \tag{3}$$

Computed in this way, z is guaranteed to be sampled from a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$.

Then, we can decode z to generate an adversarial text x'. Before feeding z to the decoder, we adopt a condition embedding c_k to guide the decoder to generate text x' of a certain class y_k, which can be chosen arbitrarily. Suppose in a text classification task, there are $|\mathcal{Y}|$ classes. Specifically, we randomly initialize a class embedding table as a matrix $C \in R^{|\mathcal{Y}| \times d}$ and look up C to get the corresponding embedding c_k of class y_k. Then, we feed $[z, c_k]$ into a linear layer to get another vector representation. The vector encodes the information of the input text and a desired class.

The decoder GRU uses this vector as the initial state to generate the output text. Each GRU cell generates one word. The computation process is similar to that of the GRU encoder, except the output layer of each cell. The output $\mathbf{O}_i$ of the i-th GRU cell is computed as:

$$\mathbf{u}_i = W_h \cdot h_i \tag{4}$$

$$\mathbf{O}_{i,k} = \frac{e^{\mathbf{u}_{i,w_k}}}{\sum_{j=1}^{|\mathcal{V}|} e^{\mathbf{u}_{i,w_j}}} \tag{5}$$

where $W_h \in \mathbb{R}^{d_{h_i} \times |\mathcal{V}|}$ is the transformation weights, $\mathcal{V}$ is the word vocabulary, $w_k \in \mathcal{V}$ and $u_i, \mathbf{O}_i \in \mathbb{R}^{|\mathcal{V}|}$. $\mathbf{O}_{i,k}$ is the probability of the i-th GRU cell emitting the k-th word w_k in the vocabulary.

In the training phase, the GRU cell chooses the word index with the highest probability to emit:

$$w_k = \arg\max_k \mathbf{O}_{i,k} \tag{6}$$

When training, the loss function of the VAE is calculated as:

$$\begin{aligned} \mathcal{L}_{VAE}(\theta, \phi) = &- \mathbb{E}_{q_\theta(z|x)}(\log p_\tau(x|z)) \\ &+ \alpha \mathbb{KL}(q_\theta(z|x)||p(z)) \end{aligned} \tag{7}$$

The first term is the reconstruction loss, or expected negative log-likelihood. This term encourages the decoder to learn to reconstruct the data. So the output text is made to be similar to the input text. The second term is the Kullback-Leibler divergence between the latent vector distribution $q_\theta(z|x)$ and $p(x)$. If the VAE were trained with only the reconstruction objective, it would learn to encode its inputs deterministically by making the variances in $q(z|x)$ vanishingly small [25]. Instead, the VAE uses the second term to encourages the model to keep its posterior distributions close to a prior $p(z)$, which is generally set as a standard Gaussian.

In the training phase, the input to the GRU decoder is the input text, appended with a special <GO> token as the start word. We add a special <EOS> token to the input text as the ground truth of the output text. The <EOS> token represents the end of the sentence. When training the GRU decoder to generate texts, the GRU decoder tends to ignore the latent code z and only relies on the input to emit output text. It actually degenerates into a language model. This situation is called KL-vanishing. To tackle the KL-vanishing problem in training GRU decoder, we adopt the KL-annealing mechanism [2]. KL-annealing mechanism gradually increase the KL weight α from 0 to 1. This can be thought of as annealing from a vanilla autoencoder to a VAE. Also, we randomly drop the input words into the decoder with a fixed keep rate $k \in [0, 1]$, to make the decoder depend on the latent code z to generate output text.

Notably, if we randomly sample z from a standard Gaussian, the decoder can also generate output text based on z. The difference is that there is no input to the GRU decoder, but we can send the word generated by the i-th GRU cell to the $(i+1)$-th GRU cell as the $(i+1)$-th input word. Specifically, in the inference phase, we use beam-search to generate words. The initial input word to the first GRU cell is the <GO> token. When the decoder emits the <EOS> token, the decoder stops generating new words, and the generation of one complete sentence is finished.

In this way, after $\mathcal{G}$ is trained, theoretically, we can sample infinite z from the latent space and generate infinite output texts based on these z. This is part of the superiority of our method.

Algorithm 1 Text Adversarial Examples Generation

Input: Training data of different classes $\mathbf{X}_0$, ..., $\mathbf{X}_{|\mathcal{Y}|-1}$
Output: Text Adversarial Examples

Train a VAE by minimizing $\mathcal{L}_{VAE}$ on $\mathbf{X}_0$, ..., $\mathbf{X}_{|\mathcal{Y}|-1}$ with KL-annealing mechanism and word drop
Initialize $\mathcal{G}$ with the pretrained VAE
Initialize the targeted model with a pretrained TextCNN
Freeze the weights of the targeted model
repeat
  for $y_k = y_0, y_1, ..., y_{|\mathcal{Y}|-1}$ **do**
    sample a batch of n texts $\{x_i\}_{i=0}^n$ of class y_k from $\mathbf{X}_k$
    $\mathcal{G}$ generates $\{x'\}_{i=0}^n$ with condition c_k
    Compute $\mathcal{L}_{disc}^k = \frac{1}{n}\sum_{i=1}^{n} \log \mathcal{D}_k(x) + \frac{1}{n}\sum_{i=1}^{n} \log(1 - \mathcal{D}_k(x'))$
  end for
  Update weights of $\mathcal{D}_0$, $\mathcal{D}_1$, ..., $\mathcal{D}_{|\mathcal{Y}|-1}$ by minimizing $-\sum_{k=1}^{|\mathcal{Y}|-1} \mathcal{L}_{disc}^k$
  Update weights of $\mathcal{G}$ by minimizing $\mathcal{L}_{joint}$
until convergence
if With inputs for the encoder **then**
  Encode inputs and decode the corresponding adversarial texts
else
  Randomly sample $z \in \mathcal{N}(0, 1)$ and choose a class $y_k \in \mathcal{Y}$
  The decoder takes $[z, c_k]$ and generates the adversarial text from scratch

3.3 Targeted Model

Since the TextCNN model has good performances and is quite fast, it is one of the most widely used methods for text classification task in industrial applications [34]. As we aim to attack models used in practice, we take the TextCNN model [13] as our targeted model.

Suppose we set the condition of the VAE to be y_k, the decoder generates the output text x', then we feed the text into the targeted model, and the targeted model will predict a probability $P_{target}(y_i)$ for each candidate class y_i. We conduct targeted attack and aim to cheat the targeted model to classify x' as class y_t ($y_t \neq y_k$), we can get the following adversarial loss function:

$$\mathcal{L}_{adv} = -\mathbb{E}_{p_\tau(x|z)}(\log P_{target}(y_t)) \tag{8}$$

This is a cross entropy loss that maximize the probability of class y_t.

Recall that words in the adversarial text x' are computed in Equation 6, in which Function $\arg\max$ is not derivative. So we can not directly feed the word index computed in Equation 6 into the targeted model. In this paper, we utilize the Gumbel-Softmax [12] to make continuous value approximate discrete word index. The embedding matrix W fed to TextCNN is calculated as:

$$\widetilde{\mathbf{O}}_{i,k} = \frac{\exp(\log((\mathbf{u}_{i,w_k}) + g_k)/t)}{\sum_{j=1}^{|\mathcal{V}|} \exp(\log((\mathbf{u}_{i,w_j}) + g_k)/t)} \tag{9}$$

$$W_i = \widetilde{\mathbf{O}}_i \cdot E \tag{10}$$

where $E \in \mathbb{R}^{|V| \times d_w}$ is the whole vocabulary embedding matrix, u_i is from Equation 4, g_k is drawn from $Gumbel(0, 1)$ distribution [12] and t is the temperature.

3.4 Discrimator Model

Until this point, ideally, we suppose the generated x' should have many same words as x of class y_k (thus be classified as y_k by humans) and be classified as class y_t by the targeted model. But this assumption is not rigorous. Most of the time, x' is not classified as y_k by humans. In natural language texts, even a single word change may change the whole meaning of a sentence. A valid adversarial example must be imperceptible to humans. That is, humans must classify x' as class y_k.

Suppose X_k is the distribution of real data of class y_k and ${X_k}'$ is the distribution of generated adversarial data transformed from $x \in X$. We utilize the idea of GAN framework to make x' similar to data from X_k. Thus x' will be classified as y_k by humans and classified as y_t at the same time.

Specifically, we adopt one discrimator $\mathcal{D}_k$ for each class $y_k \in \mathcal{Y}$. $\mathcal{D}_k$ aims to distinguish the data distribution of real labeled data x of class y_k and adversarial data x' generated by $\mathcal{G}$ with desired class y_k:

$$\mathcal{L}_{disc}^k = \mathbb{E}_{x \sim X_k}[\log(\mathcal{D}_k(x))] + \mathbb{E}_{x' \sim {X_k}'}[\log(1 - \mathcal{D}_k(x'))] \tag{11}$$

The overall training objective is a min-max game played between the generator $\mathcal{G}$ and the discrimators $\mathcal{L}_{disc}^0$, $\mathcal{L}_{disc}^1$, ..., $\mathcal{L}_{disc}^{|\mathcal{Y}|-1}$, where $|\mathcal{Y}|$ is the total number of classes:

$$\min_{\mathcal{G}} \max_{\mathcal{D}_k} \mathcal{L}_{disc}^k \tag{12}$$

$\mathcal{D}_k$ tries to distinguish X_k and ${X_k}'$, while $\mathcal{G}$ tries to fool $\mathcal{D}_k$ to make $x' \in {X_k}'$ be classified as real data by $\mathcal{D}_k$. Trained in this adversarial way, the generated adversarial text distribution ${X_k}'$ is drawn close to distribution X_k, which is of class y_k. Thus x' is mostly likely to be similar to data from X_k and is classfied as y_k by human as a result.

We implement the discrimators with multi-layer perceptions (MLPs). Because $\arg\max$ function is not derivable, similar to Equation 9 and 10 in Section 3.3, we first use Gumbel-Softmax to transform the decoder output u_i from Equation 4 into a fixed-sized matrix $V = [w_1, w_2, \ldots, w_m]^T$. Then, $\mathcal{D}_k$ calculate the probability of a text being true data of class y_k as:

$$\mathcal{D}_k(x) = \text{MLP}(V) \tag{13}$$

3.5 Model Training

Combining Equations 7, 8, 12, we obtain the joint loss function for model training:

$$\mathcal{L}_{joint} = \mathcal{L}_{VAE} + \phi\mathcal{L}_{adv} + \sum_{k=1}^{|\mathcal{Y}|-1} \mathcal{L}_{disc}^k \tag{14}$$

We first train the VAE and the targeted model f with training data. Then we freeze weights of the targeted model and initialize the $\mathcal{G}$'s weights with the pretrained VAE's weights. At last, the generator $\mathcal{G}$ and all the discrimators $\mathcal{L}_{disc}^0$, $\mathcal{L}_{disc}^1$, ..., $\mathcal{L}_{disc}^{|\mathcal{Y}|-1}$ are trained in a min-max game with loss $\mathcal{L}_{joint}$. The whole training process is summarized in Algorithm 1.

4 Experiments

We report the performances of our method on attacking TextCNN on sentiment analysis task, which is an important text classification task. Sentiment analysis is widely applied to helping a business understand

the social sentiment of their products or services by monitoring online user reviews and comments [23, 4, 21]. In several experiments, we evaluate the quality of the text adversarial examples for sentiment analysis generated by the proposed method.

Experiments are conducted from two aspects. Specifically, we first follow the popular settings and evaluate our model's performances of transforming an existing input text into an adversarial one. We observe that our method has higher attack success rate, generates fluent texts and is efficient. Besides, we also evaluate our method on generating adversarial texts from scratch unrestrictedly. Experimental results show that we can generate large-scale diverse examples. The generated adversarial texts are mostly valid, and can be utilized to substantially improve the robustness of text classification models.

We further report ablation studies, which verifies the effectiveness of the discrimators. Defense experiment results demonstrate that generating large-scale can help to make model more robust.

4.1 Experiment Setup and Details

Experiments are conducted on two popular public benchmark datasets. They are both widely used in sentiment analysis [32, 19, 8] and adversarial example generation [15, 29, 30].

Rotten Tomatoes Movie Reviews (RT) [22]. This dataset consists of $5,331$ positive and $5,331$ negative processed movie reviews. We divide 80% of the dataset as the training set, 10% as the development set and 10% as the test set.

IMDB [17]. This dataset contains 50,000 movie reviews from online movie websites. It consists of positive and negative paragraphs. 25,000 samples are for training and 25,000 are for testing. We held out 20% of the training set as a validation set as [15].

4.2 Comparing With Pair-wise Methods

In most of the existing work [26, 18, 1], text adversarial examples are generated through a pair-wise way. That is, first we should take a text example, and then transform it into an adversarial instance.

To compare with the current methods fairly, we limit our method to pair-wise generation. In this experiment, we set $\phi = 9$. Specifically, we first feed an input text into the GRU encoder, and set the condition c_k as the ground-truth class of the text. After that, the decoder can decode $[z, c_k]$ to get the adversarial output text.

We choose four representative methods as baselines:

- **Random**: Select 10% words randomly and modify them.
- **Fast Gradient Sign Method (FGSM)** [10]: First, perturbation is computed as $\varepsilon \text{sign}(\bigtriangledown_x J)$, where J is the loss function and x is the word vectors. Then, search in the word embedding table to find the nearest word vector to the perturbed word vector. FGSM is the fastest among gradient-based replacement methods.
- **DeepFool** [20]: This is also a gradient-based replacement method. It aims to find out the best direction, towards which it takes the shortest distance to cross the decision boundary. The perturbation is also applied to the word vectors. After that, nearest neighbor search is used to generate adversarial texts.
- **TextBugger** [15]: TextBugger is a gradient-free replacement method. It proposes strategies such as changing the word's spelling and replacing a word with its synonym, to change a word slightly to create adversarial texts. Gradients are only computed to find the most important words to change.

Attack Success Rate. Following the existing literature [10, 20, 15], we evaluate the attack success rate of our method and four baseline methods.

Table 1. Attack success rate of transforming given texts in a pair-wise way.

Method	RT	IMDB
Random	1.5%	1.3%
FGSM+NNS	25.7%	36.2%
DeepFool+NNS	28.5%	23.9%
TextBugger	85.1%	90.5%
Ours ($\phi = 5$)	$\mathbf{87.1}\%$	$\mathbf{92.8}\%$

We summarize the performances of of our method and all baselines in Table 1. From Table 1, we can observe that randomly changing 10% words is not enough to fool the classifier. This implies the difficulty of attack. TextBugger and our method both achieve quite high attack success rate. While our method performs even better than TextBugger, which is the state-of-the-art method.

We show some adversarial examples generated by our method and TextBugger to demonstrate the differences in Figure 4.

We can observe that TextBugger mainly changes the spelling of words. The generated text becomes not fluent and easy to be detected by grammar checking systems. Also, though humans may guess the original meanings, the changed words are treated as out of vocabulary words by models. For example, TextBugger changes the spelling of 'awful', 'cliches' and 'foolish' in Figure 4. These words are important negative sentiment words for a negative sentence. It is natural that changing these words to unknown words can change the prediction of models. Unlike TextBugger, our method generates meaningful and fluent contents. For example, in the first example of Figure 4, we replace 'read the novel' with 'love the book', the substitution is still fluent and make sense to both humans and models.

Generation Speed. It takes about one hour and about 3 hours to train our model on RT dataset and IMDB dataset respectively. We also evaluate the time cost of generating one adversarial example. We take the FGSM method as the representative of gradient-based methods, as FGSM is the fastest among them. We measure the time cost of generating $1,000$ adversarial examples and calculate the average time of generating one. Results are shown in Table 2.

Table 2. Time cost of generating one adversarial text.

Method	FGSM+NNS	TextBugger	Ours ($\phi = 5$)
Time	0.7s	0.05s	**0.014s**

We can observe that our method is much faster than others. That is mainly because our generative model is trained beforehand. After the model is trained, the generation of one batch just requires one feed-forward.

4.3 Unrestricted Adversarial Text Generation

As mentioned in Section 3.2, after our model is trained, we can randomly sample z from latent space, choose a desired class $y_k \in \mathcal{Y}$, get the embedding vector c_k of y_k, then feed $[z, c_k]$ to the decoder to generate adversarial texts unrestrictedly with no need of labeled text.

Attack Success Rate. When training, we can tune ϕ in Equation 14 to affect the model. After trained with different ϕ, we observe the generated texts are different. We randomly generate 50,000 examples and compute the proportion of adversarial examples with different ϕ. The results are shown in Figure 5(a). Notice if we set $\phi = 0$,

Dataset: RT. **Method:** Ours($\phi = 9$). **Ground-truth:** Positive. **Original prediction:** 0.95 Positive. **Adversarial prediction:** 0.68 Negative.

Text: ~~Inside~~ one the films ~~conflict~~ like powered plot there is a decent moral trying to get out but its not that it's the ~~tension~~ first that ~~keeps~~ makes you ~~in~~ feel your seat affleck and jackson ~~are good~~ is magnificent sparring partners

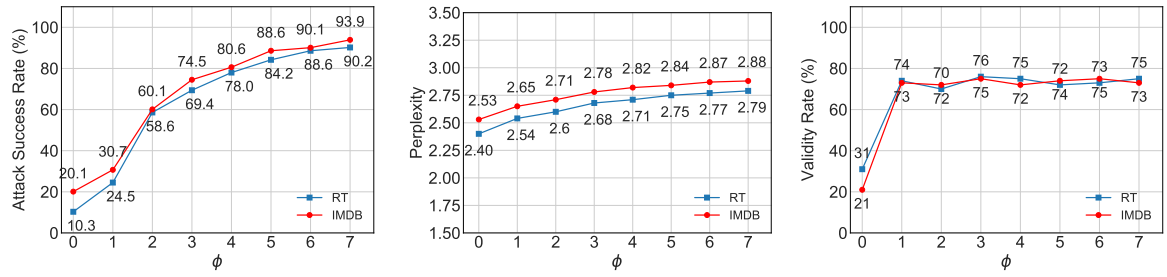
Dataset: IMDB. **Method:** Ours($\phi = 9$). **Ground-truth:** Negative. **Original prediction:** 0.98 Negative. **Adversarial prediction:** 0.94 Positive.

Text: i ~~read the novel~~ love the book some years ago and i ~~liked~~ loved it a lot when i ~~saw the~~ read this movie i ~~couldnt believe~~ was cared it ~~they changed~~ was thrown everything i ~~liked~~ expected about the ~~novel~~ book even the plot i wonder ~~what~~ if did isabel allende ~~author~~ did say about ~~the~~ this movie but i think it sucks

Dataset: IMDB. **Method:** TextBugger. **Ground-truth:** Negative. **Original prediction:** 0.99 Negative. **Adversarial prediction:** 0.81 Positive.

Text: I love these ~~awful~~ awf ul 80's summer camp movies. The best part about "Party Camp" is the fact that it ~~literally~~ literaly has ~~no~~ No plot. The ~~cliches~~ clichs here are limitless: the nerds vs. the jocks, ..., the secretly horny camp administrators, and the ~~embarrassingly~~ embarrassing1y ~~foolish~~ fo0lish sexual innuendo littered throughout. This movie will make you laugh, but never intentionally. I repeat, never.

Figure 4. Adversarial texts generated in a pair-wise way. In texts, the crossed out contents are from the original texts, while the red texts are the substitute contents in the adversarial examples.



(a) Attack Success Rate (b) Perplexity (c) Validity Rate

Figure 5. The attack success rate, perplexity and validity of unrestricted adversarial text generation from scratch. Randomly sample z to generate adversarial texts from scratch with different ϕ. Note that when $\phi = 0$, the model is a vanilla VAE

Dataset: RT. **Method:** Vanilla VAE ($\phi = 0$). **Chosen Emotional Class:** Negative. **Model prediction:** 0.53 Positive.

Text: this is the kind of movie that might have been benefited from a movie that is not more than a movie

Dataset: IMDB. **Method:** Vinilla VAE ($\phi = 0$). **Chosen Emotional Class :** Positive. **Model prediction:** 0.54 Negative.

Text: i had never heard of this movie until the end of the first half hour or minutes we were glued to the edge of your seat throughout the entire movie i thought it was going to be a good idea to see a movie about a bunch of people trying to find out what happened to their ... see if you want to see a movie that is going to happen next to the end

Dataset: RT. **Method:** Ours ($\phi = 1$). **Chosen Emotional Class :** Positive. **Model prediction:** 0.99 Negative

Text: theres no reason to be disappointed

Dataset: RT. **Method:** Ours ($\phi = 7$). **Chosen Emotional Class :** Negative. **Model prediction:** 0.89 Positive

Text: sandra bullock fish out into a dark and poorly executed story about about about which he doesnt manage to be a joyful teacher

Dataset: IMDB. **Method:** Ours ($\phi = 1$). **Chosen Emotional Class :** Negative. **Model prediction:** 0.97 Positive

Text: this was the first time i saw this movie when i was a kid i was expecting it to be the first time i saw this movie i was thoroughly impressed with this movie was that it was so bad

Dataset: IMDB. **Method:** Ours ($\phi = 7$). **Chosen Emotional Class :** Positive. **Model prediction:** 0.93 Negative

Text: a lot of fun to watch this movie is about a virus who crashes in the himalayas unlucky enough to take a trip to the old house in the woods in the himalayas unlucky enough to be a photographer and wanted to prevent the freezing man in a limb in a limb in a limb in a limb in a limb in his assignment to stop him he decides to take him out of his apartment with his wife

Figure 6. Adversarial examples generated from scratch unrestrictedly. Humans should classify adversarial texts as the chosen emotional class y_k.

the model is a vanilla VAE and it is not trained continually after pre-trained.

From Figure 5(a), we can observe that the attack success rate of the vanilla VAE is only 10.3% and 20.1% respectively, this implies that only randomly generating texts can hardly fool the targeted model. When ϕ is greater than 0, the attack success rate is consistently better than the vanilla VAE. This reflects the importance of $\mathcal{L}_{adv}$.

Also, the attack success rate increases as ϕ becomes larger. It is because the larger ϕ is, the more important role $\mathcal{L}_{adv}$ will plays in the final joint loss $\mathcal{L}_{joint}$. So, the text generator $\mathcal{G}$ is more easily guided by the $\mathcal{L}_{adv}$ to generate an adversarial example.

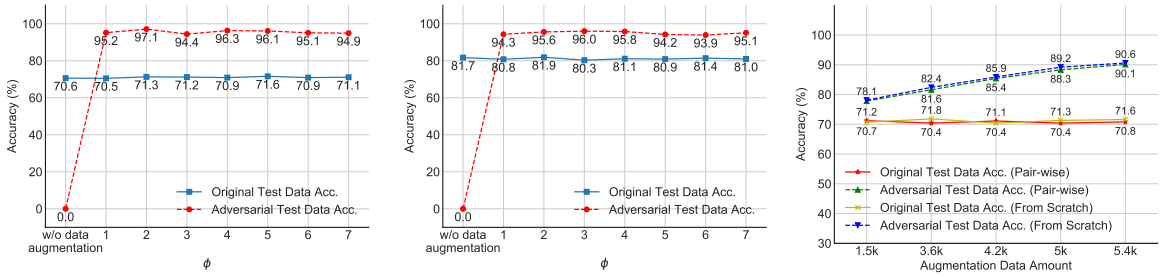
To evaluate the quality of the generated adversarial texts with different ϕ, we adopt three metrics : perplexity, validity and diversity.

Perplexity. Perplexity [3] is a measurement of how well a probability model predicts a sample. A low perplexity indicates the language model is good at predicting the sample. Given a pretrained language model, it can also be used to evaluate the quality of texts. Similarly, a low perplexity indicates the text is more fluent for the language model. We compute perplexity as:

$$Perplexity = -\frac{1}{|word\_num|}\sum_{x\in X'}\sum_{j=1}^{V}\log P(x'_j|x'_0,...,x'_{j-1}) \tag{15}$$

where V is the number of words in one sentence. $P(x'_j)$ is the probability of j-th word in x' computed by the language model.

We train a language model with the training data of IMDB and RT, and use it as P in Equation 15. We measure and compare the perplexity of the generated 50,000 texts and data of the original training set. Results are shown in Figure 5(b). We can observe that the perplexity is only a bit higher than the original data's, which means that the quality of generated texts are acceptable. Also, as ϕ gets larger, the perplexity gets bigger. This is perhaps because $\mathcal{L}_{adv}$ can distort the generated texts.



(a) RT (b) IMDB (c) Data augmentation compare

Figure 7. Defense with adversarial training in different settings. (a) and (b) On RT and IMDB datasets, data augmentation with adversarial data generated from scratch under different ϕ. (c) On RT dataset, accuracy of models trained with equal size of augmentation adversarial data, which is generated in pair-wise way and unrestricted generation way respectively.

Validity. If we feed $[z, c_k]$ to the decoder, then a valid generated adversarial text is supposed to be classified as class y_k by humans but be classified as class $y_t \neq y_k$ by the targeted model. We randomly select 100 generated texts for each ϕ and manually evaluate their validity. The results are shown in Figure 5(c).

From Figures 5(c), we can observe that the validity rates of our method on both datasets are higher than 70% and much higher than that of the vanilla VAE. This implies our methods can generate high-quality and high-validity texts with high attack success rate.

Diversity. We first generate one million adversarial texts. To compare generated texts with train data, we extract all 4-grams of train data and generated texts. On average, for each generated text, less than 18% of 4-grams can be found in all 4-grams of train data on all datasets. This shows that there exists some similarity and our model can also generate texts with different words combinations. To compare generated texts with each other, we suppose that if over 20% of 4-grams of one generated text don't exist at the same time in any one of the other generated texts, the text is one unique text. We observe more than 70% of generated texts are unique. This proved that the generated texts are diverse.

Adversarial Examples. We show some valid adversarial examples generated by our method in Figure 6. We can view that the adversarial examples generated by the vanilla VAE is more likely neutral, and the confidence of the targeted model is not huge. On the contrary, the generated examples of our method have high confidence of the targeted model. This shows $\mathcal{L}_{adv}$ is important to attack success rate. Besides, the fluency and validity of texts generated by our method are acceptable.

4.4 Ablation Study

In this section, we further demonstrate the effectiveness of discrimators. We now report the ablation study.

We first remove discrimators and $\mathcal{L}_{disc}$, then train our model. We compare it with the model trained with $\mathcal{L}_{joint}$ in a min-max game. We evaluate their attack success rate, perplexity and validity. Results are show in Table 3.

Table 3. Performance of our model trained with and without $\mathcal{L}_{disc}$.

Dataset	Method	Attack Success Rate	Perplexity	Validity
RT	with $\mathcal{L}_{disc}$	90.2%	2.79	75%
	without $\mathcal{L}_{disc}$	94.1%	7.32	15%
IMDB	with $\mathcal{L}_{disc}$	93.9%	2.88	73%
	without $\mathcal{L}_{disc}$	94.3%	7.41	12%

The attack success rates of models trained with and without $\mathcal{L}_{disc}$ are close. But the validity of the model trained without $\mathcal{L}_{disc}$ is much lower than that of the model with $\mathcal{L}_{filter}$. The reason of this phenomenon is as follows. When training the generator $\mathcal{G}$ with only $\mathcal{L}_{VAE}$ and $\mathcal{L}_{adv}$, suppose we want to generate positive adversarial texts and the targeted model must classify it as negative, the easiest way to achieve this goal is to change a few words in the generated text to negative words, such as "bad". But texts generated this way can not fool humans. If we add discrimators to draw distribution of adversarial texts close to the distribution of real data, this phenomenon can be controlled. This shows that discrimators and the min-max game $min\,max\,\mathcal{L}^k_{disc}$ can improve the validity greatly.

4.5 Defense With Adversarial Training

Using the adversarial examples to augment the training data can make models more robust, this is called adversarial training.

On RT dataset, we randomly generate 4k adversarial texts to augment the training data and 1k to test the model. On IMDB dataset, we randomly generate 10k, of which 8k for training and 2k for testing. Results are shown in Figure 7(a) and Figure 7(b).

Through adversarial data augmentation, test accuracy on the original test data is stable. Also, the accuracy on the adversarial data is improved greatly (from 0 to $> 90\%$). It implies that adversarial training can make models more robust without hurting its effectiveness.

Then, on RT dataset, we first augment training data with adversarial examples generated by pair-wise generation. The adversarial examples are generated through transforming training data. Note that we have 8k training data in RT dataset. When we set bigger ϕ, the attack success rate is higher, so we can generate more adversarial examples in the pair-wise way. But with any ϕ, unrestricted generation from scratch can result in infinite adversarial data. We compare the adversarial data augmentation performances of pair-wise and unrestricted generation from scratch. We use the same number of adversarial examples generated by the two modes, and hold out 20% of generated data for testing. Results are shown in Figure 7(c).

We can see that with pair-wise generation, if training data is limited, we need to generate more adversarial examples to improve the adversarial test accuracy. Higher adversarial test accuracy requires higher ϕ. But higher ϕ results in bigger perplexity, which means low text quality. Differently, with unrestricted generation from scratch, we can generate infinite adversarial texts using very small ϕ, with high fluency and similar adversarial test accuracy. Thus, under similar adversarial test accuracy, the text fluency of pair-wise generation

is worse than that of unrestricted generation from scratch. This indicates the advantage of our proposed method.

5 Conclusion

In this paper, we have proposed a scalable method to generate adversarial texts from scratch attacking a text classification model. We add an adversarial loss to enforce the generated text to mislead the targeted model. Besides, we use discrimators and GAN-like training strategy to make adversarial texts mimic real data of the desired class. After the generator is trained, it can generate diverse adversarial examples of a desired class on a large scale without real-world texts. Experiments show that the proposed method is scalable and can achieve higher attack success rate at a higher speed compared with recent methods. In addition, it is also demonstrated that the generated texts are of good quality and mostly valid. We further conduct ablation experiments to verify effects of discrimators. Experiments of data augmentation indicate that our method generates more diverse adversarial texts with higher quality than pair-wise generation, which can make the targeted model more robust.

REFERENCES

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

[2] Samuel Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio, 'Generating sentences from a continuous space.', in *Proceedings of the Twentieth Conference on Computational Natural Language Learning (CoNLL)*., (2016).

[3] Peter F Brown, Vincent J Della Pietra, Robert L Mercer, Stephen A Della Pietra, and Jennifer C Lai, 'An estimate of an upper bound for the entropy of english', *Computational Linguistics*, **18**(1), 31–40, (1992).

[4] Erik Cambria, 'Affective computing and sentiment analysis', *IEEE Intelligent Systems*, **31**(2), 102–107, (2016).

[5] Nicholas Carlini and David Wagner, 'Towards evaluating the robustness of neural networks', in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57. IEEE, (2017).

[6] Minhao Cheng, Jinfeng Yi, Huan Zhang, Pin-Yu Chen, and Cho-Jui Hsieh, 'Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples', *arXiv preprint arXiv:1803.01128*, (2018).

[7] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio, 'On the properties of neural machine translation: Encoder-decoder approaches', *arXiv preprint arXiv:1409.1259*, (2014).

[8] Ari Firmanto, Riyanarto Sarno, et al., 'Prediction of movie sentiment based on reviews and score on rotten tomatoes using sentiwordnet', in *2018 International Seminar on Application for Technology of Information and Communication*, pp. 202–206. IEEE, (2018).

[9] Zhitao Gong, Wenlu Wang, Bo Li, Dawn Song, and Wei-Shinn Ku, 'Adversarial texts with gradient methods', *arXiv preprint arXiv:1801.07175*, (2018).

[10] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy, 'Explaining and harnessing adversarial examples', *arXiv preprint arXiv:1412.6572*, (2014).

[11] Weiwei Hu and Ying Tan, 'Generating adversarial malware examples for black-box attacks based on gan', *arXiv preprint arXiv:1702.05983*, (2017).

[12] Eric Jang, Shixiang Gu, and Ben Poole, 'Categorical reparameterization with gumbel-softmax', *arXiv preprint arXiv:1611.01144*, (2016).

[13] Yoon Kim, 'Convolutional neural networks for sentence classification', in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1746–1751, (2014).

[14] Diederik P Kingma and Max Welling, 'Auto-encoding variational bayes', *arXiv preprint arXiv:1312.6114*, (2013).

[15] J Li, S Ji, T Du, B Li, and T Wang, 'Textbugger: Generating adversarial text against real-world applications', in *26th Annual Network and Distributed System Security Symposium*, (2019).

[16] Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi, 'Deep text classification can be fooled', in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 4208–4215. AAAI Press, (2018).

[17] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts, 'Learning word vectors for sentiment analysis', in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pp. 142–150. Association for Computational Linguistics, (2011).

[18] Paul Michel, Xian Li, Graham Neubig, and Juan Miguel Pino, 'On evaluation of adversarial perturbations for sequence-to-sequence models', *arXiv preprint arXiv:1903.06620*, (2019).

[19] Melody Moh, Abhiteja Gajjala, Siva Charan Reddy Gangireddy, and Teng-Sheng Moh, 'On multi-tier sentiment analysis using supervised machine learning', in *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 1, pp. 341–344. IEEE, (2015).

[20] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard, 'Deepfool: a simple and accurate method to fool deep neural networks', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2574–2582, (2016).

[21] Juan Antonio Morente-Molinera, Gang Kou, Konstantin Samuylov, Raquel Ureña, and Enrique Herrera-Viedma, 'Carrying out consensual group decision making processes under social networks using sentiment analysis over comparative expressions', *Knowledge-Based Systems*, **165**, 335–345, (2019).

[22] Bo Pang and Lillian Lee, 'Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales', in *Proceedings of the 43rd annual meeting on association for computational linguistics*, pp. 115–124. Association for Computational Linguistics, (2005).

[23] Bo Pang, Lillian Lee, et al., 'Opinion mining and sentiment analysis', *Foundations and Trends® in Information Retrieval*, **2**(1–2), 1–135, (2008).

[24] Nicolas Papernot, Patrick McDaniel, Ananthram Swami, and Richard Harang, 'Crafting adversarial input sequences for recurrent neural networks', in *MILCOM 2016-2016 IEEE Military Communications Conference*, pp. 49–54. IEEE, (2016).

[25] Tapani Raiko, Mathias Berglund, Guillaume Alain, and Laurent Dinh, 'Techniques for learning binary stochastic feedforward neural networks', *arXiv preprint arXiv:1406.2989*, (2014).

[26] Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che, 'Generating natural language adversarial examples through probability weighted word saliency', in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1085–1097, (2019).

[27] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra, 'Stochastic backpropagation and approximate inference in deep generative models', in *International Conference on Machine Learning*, pp. 1278–1286, (2014).

[28] Suranjana Samanta and Sameep Mehta, 'Towards crafting text adversarial samples', *arXiv preprint arXiv:1707.02812*, (2017).

[29] Motoki Sato, Jun Suzuki, Hiroyuki Shindo, and Yuji Matsumoto, 'Interpretable adversarial perturbation in input embedding space for text', *arXiv preprint arXiv:1805.02917*, (2018).

[30] Congzheng Song and Vitaly Shmatikov, 'Fooling ocr systems with adversarial text images', *arXiv preprint arXiv:1802.05385*, (2018).

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

[32] Prayag Tiwari, Brojo Kishore Mishra, Sachin Kumar, and Vivek Kumar, 'Implementation of n-gram methodology for rotten tomatoes review dataset sentiment analysis', *International Journal of Knowledge Discovery in Bioinformatics (IJKDB)*, **7**(1), 30–41, (2017).

[33] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song, 'Spatially transformed adversarial examples', *arXiv preprint arXiv:1801.02612*, (2018).

[34] Wei Emma Zhang, Quan Z Sheng, A Alhazmi, and C Li, 'Adversarial attacks on deep learning models in natural language processing: A survey', *arXiv preprint arXiv:1901.06796*, (2019).