Adversarial Attack on Sentiment Classification

Alicia Yi-Ting Tsai
aliciatsai@berkeley.edu

Tobey Yang
minchu.yang@berkeley.edu

Erica Chen
hanyu_chen@berkeley.edu

Abstract

In this paper, we propose a white-box attack algorithm called "Global Search" method and compare it with a simple misspelling noise and a more sophisticated and common white-box attack approach called "Greedy Search". The attack methods are evaluated on the Convolutional Neural Network (CNN) sentiment classifier trained on the IMDB movie review dataset. The attack success rate is used to evaluate the effectiveness of the attack methods and the perplexity of the sentences is used to measure the degree of distortion of the generated adversarial examples. The experiment results show that the proposed "Global Search" method generates more powerful adversarial examples with less distortion or less modification to the source text.

1 Introduction

In the past few decades, machine learning and deep learning techniques have been successful in several applications. However, these techniques developed so far are proven to be vulnerable given some manipulated inputs, which are called adversarial examples, that human can easily distinguish but algorithms can not (Szegedy et al., 2014; Goodfellow et al., 2015).

Current research have shown successful results in producing adversarial images that cause the algorithms to completely fail in computer vision (Kurakin et al., 2016). Studies of adversarial examples in the applications of natural language processing such as sentiment analysis, fake news detection and machine translation remain relatively low. Nonetheless, it is an emerging field that is worth exploring and has increased attention recently due to the success of adversarial learning in images. When generating an adversarial example, if the adversary does not have knowledge of the classifier or the training data, we call this a black-box setting. On the other hand, if the adversary has full knowledge of the classifier and the training data, we call this a white-box setting.

In a black-box setting, Belinkov and Bisk introduces a simple attack method by randomly replacing characters with their nearby key on the keyboard, which is similar to keyboard typos, to attack a machine translation system (Belinkov and Bisk, 2017). Similar idea can be found in the work of Hosseini et al., the authors generate adversaries that deceive Google Perspective API by misspelling the abusive words or adding punctuation to the letters (Hosseini et al., 2017). Another work from Alzantot et al. attempts to generate semantically and syntactically similar adversarial examples by word replacement (Alzantot et al., 2018). They develop an genetic algorithm that uses population-based gradient-free optimization, inspired by the process of natural selection. In the black-box setting, the adversary tries different perturbations and evaluates the quality of perturbations by querying the model to get the classification result or the output score. The adversary continues to altered the sentence until the model fails or until score reduces significantly.

In the white-box setting, the adversary has access to the model and thus is capable of generating more sophisticated adversarial examples. Ebrahimi et al. show that adversarial examples generated in a white-box setting achieve a higher success rate than examples generated in a black-box setting. The authors introduce a white-box adversary against differentiable classifiers that substitutes characters ("flips") in a sentence. When operating in a white-box setting, the adversary has full access to the gradients of the classifier, giv-

Proceedings of the Second BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, pages 233–240
Florence, Italy, August 1, 2019.

ing the adversary important information to find the classifier's weak points. Because white-box adversary has access to the gradients of the model, the adversary does not have to query the output score from the classifier every time. Using the gradients as a surrogate loss, the white-box adversary can efficiently find the best changes that maximize the surrogate loss simply by backpropagation.

Other white-box adversary includes word-level substitution. Kuleshov et al. try to replace 10-30% of words in the source text by solving an optimization problem that maximizes a surrogate loss subject using a greedy approach, which is similar to the "Greedy Search" baseline used in the experiment. A similar idea can be found in Samanta and Mehta where the authors apply different rules (insertion, replacement and deletion of words) to generate adversarial examples. Liang et al. later combine the strategies above and try to avoid introducing excessive modification or insertion to the original source text.

In this paper, we consider the task of white-box attack where the adversary has full knowledge of the model under attack. We propose a "Global Search" attack method that mitigates some of the problems faced in the commonly used greedy approach. A very simple misspelling noise baseline is also reported to show the effectiveness of the 2 white-box attack methods in the experiment.

2 Attack Method

2.1 Misspelling Noise

We first consider a very simple case by swapping two characters in a word (eg. pe**rf**ect $\rightarrow$ pe**fr**ect), which is similar to keyboard typos or misspelling. To maintain the readability of source text, the noise is only applied to word of length longer than 3 and 50% of the words in the source text are randomly swapped.

2.2 Greedy Search Approach

We follow a similar greedy optimization strategy in (Kuleshov et al., 2018) for constructing adversarial examples for sentiment classification. At each iteration, the algorithm considers k nearest neighbors of each word in the word embedding space. It then picks the one among the k neighbors that has the greatest impact on the prediction result. Consider a sentiment classifier f, the greedy approach forms the adversarial example by replacing the original word w with the candidates w'. If the label is positive, then the final sigmoid layer of the model should output a value σ greater than 0.5. The algorithm then replaces the original word w with each candidate w' among the k neighbors and see if the sigmoid value of the adversary x' is less than σ, indicating that replacing w with w' can contribute to flipping the prediction result to negative. The candidate w' that results in the maximum difference is then chosen to be the final replacement, i.e. smallest σ' for positive class (1) and the largest σ' for the negative class (0).

$$\begin{aligned} &\arg\min_{w'} f_{\text{sigmoid}}(x') \\ \text{s.t. } &f_{\text{sigmoid}}(x') < f_{\text{sigmoid}}(x) \quad \text{if positive class} \end{aligned}$$

$$\begin{aligned} &\arg\max_{w'} f_{\text{sigmoid}}(x') \\ \text{s.t. } &f_{\text{sigmoid}}(x') > f_{\text{sigmoid}}(x) \quad \text{if negative class} \end{aligned}$$

Algorithm 1 summarizes the "Greedy Search" approach. Although the objective of the algorithm is to fool the sentiment classifier, the grammar, semantic and sentiment should not be altered. Thus, some constraints are imposed in order to maintain the similarity between the original source text and the adversarial example.

Algorithm 1 Generate adversarial example via greedy search

$\sigma \leftarrow f_{\text{sigmoid}}(x)$
Initialize $x' \leftarrow x$, *count* $\leftarrow 0$
for w in x' **do**
    candidates $\leftarrow$ k nearest neighbors of w within distance d and have the same POS tag as w
    $\Sigma \leftarrow \emptyset$
    for w' in *candidates* **do**
        $\bar{x} \leftarrow$ replace w with w'
        $\sigma' \leftarrow f_{\text{sigmoid}}(\bar{x})$
        $\Sigma \leftarrow \Sigma$ append σ'
    if positive and $\min \Sigma < \sigma$ **then**
        $x' \leftarrow \text{argmin}_{\bar{x}} \Sigma$
        count $\leftarrow$ *count* $+1$
    else if negative and $\max \Sigma > \sigma$ **then**
        $x' \leftarrow \text{argmax}_{\bar{x}} \Sigma$
        count $\leftarrow$ *count* $+1$
    $\sigma \leftarrow f_{\text{sigmoid}}(x')$
    if positive and $\sigma < 0.5$ **then return** x'
    if negative and $\sigma > 0.5$ **then return** x'
    if *count*$/\text{len}(x) > r$ **then return** *None*

Word Choice. When picking the word candidates in each iteration, we consider the top k nearest neighbors in the word embedding space. The nearest neighbors in GloVE word embedding space usually appear in the same context as the original word. Thus, by picking nearest neighbors in GloVE vectors, we can ensure semantic similarity after word replacement. Although word embedding helps to find words that are used in similar context, it does not guarantee that the part of speech (POS) will remain the same after replacement. Therefore, we examine the part of speech of the original word and ensure the selected candidates have the same part of speech as the original word.

Hyper-parameters. There are 3 hyper-parameters, k, d, and r, used in Algorithm 1 to maintain the semantic similarity of the sentences. The first parameter k is used to decide how many nearest neighbors should be considered in the first place. When k is too small, there might not be enough candidates to successfully form an adversarial example. d represents the maximum distance allowed between the candidate words and the original word in the word embedding space. When d is large, the meaning of the altered sentence is generally farther from the original one. When d gets too small, the chance of generating a successful attack decreases. The last hyper-parameter, r, stands for the percentage of replacement allowed in the sentence. When the threshold r is too low, the chance of generating a successful adversarial example decreases. Similarly, when r is too large, too many words will be replaced and thus the generated sentence will be very dissimilar to the original one.

2.3 Global Search Approach

The greedy approach proposed above does not guarantee to produce the optimal results and is sometimes time consuming because the algorithm needs to search the candidate words for every iteration. To mitigate the problem, we propose to search for the candidates globally by computing a small perturbation, δ.

To learn the perturbation, δ, we define an objective function $J(\delta)$. The objective function tries to maximize the difference between the sigmoid value of the original input x and that of perturbed input $x + \delta$. Furthermore, we add two regularization terms in the objective function. The first regularization penalizes large perturbations and is controlled via the hyperparameters λ_1. The second regularization penalizes large distance between original word embedding and perturbed word embedding and is controlled by the hyperparameters λ_2. The two regularization terms are added to help keep the semantic of chosen words.

$$\begin{aligned}\mathcal{J}(\delta) =&(f_{\text{sigmoid}}(E_x) - f_{\text{sigmoid}}(E_x + \delta))^2 \\ &+ \lambda_1 \cdot \|\delta\|_2 \\ &+ \lambda_2 \cdot \|(E_x - (E_x + \delta))\|_2\end{aligned}$$

The attack algorithm is described in Algorithm 2. We first initialize the perturbation δ to be 0. Here, the original input embedding is denoated as E_x and the perturbed embedding is denoted as $E'_x = E_x + \delta$. For each iteration, the algorithm computes the gradient, ∇_δ, of the objective function $J(\delta)$ with respect to the perturbation δ, and update the perturbation δ via backpropagation. We then form a perturbed embedding E'_x. The perturbed embedding E'_x is a matrix and each row of E'_x is the perturbed word embedding for each word, denoted as e, in the source text. The perturbed embedding e usually does not have a actual word that can be mapped back to. Thus, we find the candidate words w' in the embedding space that is the closest to the perturbed word embedding e and record the w' in the candidate list k_e. The candidate words for replacement are recorded in another list W'. After computing the perturbed embedding, and record the candidate words, the algorithm checks if the current perturbed embedding E'_x flips the prediction result. The algorithm continues to compute new E'_x and record the candidate words if the previous E'_x fails to fool the model, otherwise, the algorithm returns the candidate words and the perturbation.

Next, we can use the generated candidate words list and the perturbation to generate the adversarial example. The algorithm is described in algorithm 3. The algorithm sorts the magnitude of the perturbation and replaces words in the positions that have higher perturbation magnitude. A higher perturbation magnitude indicates that the classifier is more sensitive to changes of the original word. Here, we have a hyperparameter d that controls the threshold for word distance. If the last candidate w'_n in the candidate words list is too far away from the original word, then the algorithm rejects the candidate w'_n and move to the previous candidate w'_{n-1} in the list. The hyperparameter r controls

Algorithm 2 Global Search Attack Function

$y \leftarrow \textit{Round}(f_{\text{sigmoid}}(x))$
Initialize $\delta \leftarrow 0$, *success = False*
$E_x \leftarrow$ input embedding
$W' \leftarrow \emptyset$
for e in E'_x **do**
    $k_e \leftarrow \emptyset$ ▷ Empty candidate list
    $W' \leftarrow W'$ *append* k_e
while not *success* **do**
    $\nabla_\delta \leftarrow$ via *back-propagation*
    $\delta \leftarrow \delta - \epsilon \cdot \nabla_\delta$
    $E'_x \leftarrow E_x + \delta$ ▷ perturbed embedding
    for e in E'_x **do**
        $k_e \leftarrow k_e$ *append* $\text{argmin}_{w'} \|e - E_{w'}\|_2$
    $\hat{y} \leftarrow \textit{Round}(f_{\text{sigmoid}}(E'_x))$
    $W' \leftarrow W' \cup \{k_e\}$
    if $\hat{y} \neq y$ **then**
        return W', δ

the percentage of changes allowed as the described earlier.

Algorithm 3 Global Search Generate Adversary Function

$y \leftarrow \textit{Round}(f_{\text{sigmoid}}(x))$
positions $\leftarrow$ *reversed*(argsort $\|\delta\|_2$)
Initialize *success = False*
for i in *positions* **do**
    $w \leftarrow i$-th word in the source text x
    candidates $\leftarrow W'_i$
    candidates $\leftarrow$ *reversed(candidates)*
    for w' in *candidates* **do**
        if $\|e_w - e_{w'}\|_2 < d$ **then**
            $E'_i \leftarrow e_{w'}$ ▷ replace embedding
            break
    $x'_i \leftarrow w'$ ▷ replace word
    $\hat{y} \leftarrow \textit{Round}(f_{\text{sigmoid}}(E'_x))$
    if $\hat{y} \neq y$ **then**
        success = True
        return x'
    if $\frac{i}{\text{len}(x)} > r$ **then**
        return *None*

3 Experiment

3.1 Dataset

We use a dataset of 25,000 informal movie reviews from the Internet Movie Database (IMDB) (Maas et al., 2011) and randomly select 80% of the dataset to include in the training set, and 20% in the testing set [1].

3.1.1 CNN Sentiment Classifier

In this experiment, we used the convolutional neural network classifier (Kim, 2014) as the target model to be attacked. We replicated the architecture of the CNN model from Kim work. In the CNN model, an embedding of a fixed dictionary and size serves as the very first layer. Convolutional layers with filter widths of 3, 4, 5 and 100 are added with a max-over-time pooling layer and a fully connected layer with dropout rate of 0.5. We trained the model with 20,000 reviews and tested it with another 5,000 reviews with batch size of 64, reaching testing accuracy of 0.9. The result is summarized in Table 1.

3.2 Evaluation

3.2.1 Success Rate

The end goal of the attack algorithms is to trick the model to make wrong prediction by manipulating the input. To access the effectiveness of the attack models, we select 500 examples that are correctly classified from the test set, so that the accuracy of the classifiers does not affect the evaluation. We then input these source text into the attack algorithms to generate adversarial examples. The adversarial examples are then fed into the CNN classifier to get the final prediction. The success rate of the attack algorithm is defined as the percentage of wrong prediction by the CNN classifier. Higher success rate means that the attack algorithm can generate more powerful adversaries that can cause the CNN classifier to misbehave. The experiment result in Table 2 shows that Global Search approach has 72% success rate while Greedy Search approach has 65%. Compared with the Global Search method, it generally requires higher percentage of words replacement for Greedy Search to successfully generate an adversary. Because of the greedy nature, the algorithm replaces words as long as changing the words can help confuse the classifier; hence, the algorithm can replace sub-optimal words in a earlier position that do not contribute the most to the end goal. Another limitation of the Greedy Search approach is that word replacements tend to locate in a close area of the sentence, especially in an earlier position. This greatly reduce the readability of

[1] https://pytorchnlp.readthedocs.io/en/latest/_modules/torchnlp/datasets/imdb.html

Accuracy	Training	Testing
CNN	0.99	0.90

Table 1: Sentiment Classifier Result

the sentences and can destroy the semantic meaning of the source text.

3.2.2 Hyper-parameters Choice

As mentioned above, there are 3 hyper-parameters in the greedy approach, which are k, the number of candidates to be considered; d, the maximum distance allowed between the original word and the candidate; r, the ratio of word changes allowed with respect to the length of the sentence. The global search algorithm also includes the hyper-parameters d and r. In our experiment of the greedy approach, k is set to 20. This parameter does not affect much since the maximum word distance allowed and the limitation of picking words with the same part-of-speech tag help to maintain similarity. The parameter is used to reduce the run-time complexity of the algorithm by limiting the number of candidates. As for d, we run the algorithm for a few different values, we decide to set d to 20 to reach a high success rate while not picking words too far from the original one. There exists a trade-off between similarity and success rate in choosing the value for d. While allowing words farther from the original word, the success rate increases but similarity decreases as a penalty. The threshold r controls how many words are allowed to change in a sentence; it is also shared by the two approaches. In greedy approach, r should be at least set to 0.2 to achieve a good result. When r is set to 0.1, the success rate is merely 0.18; when r is 0.2, the success rate increases to 0.65. So we decide to set the threshold r to 0.2 for greedy approach in the following experiment. On the other hand, the global search algorithm can reach a pretty good result when r is set to 0.1. In our experiment, we found that global search algorithm is more effective than the greedy approach since the global search algorithm looks for the word that contributes most to classification result while greedy algorithm swap words with any degree of contribution.

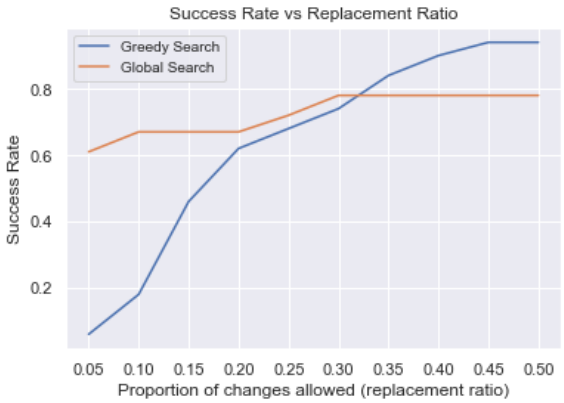

Figure 1: Success Rate vs Replacement Ratio

Figure 2: Success Rate vs Word Distance

3.2.3 Perplexity

In order to evaluate the adversarial examples generated from our models, we decide to measure the perplexity, which is widely used in assessing

Misspelling Noise	Greedy Search	Global Search
6%	65%	72%

Table 2: Success rate of attacking methods

Figure 3: Average Perplexity vs Word Distance

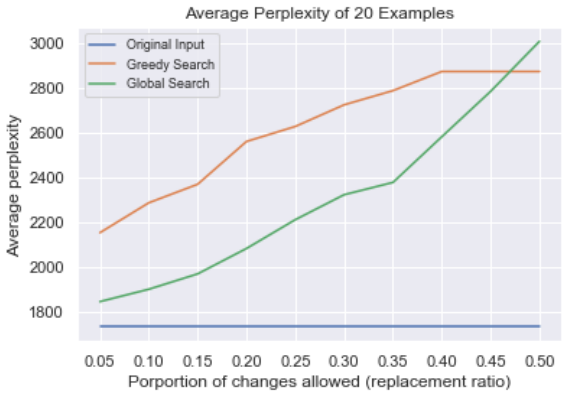

Figure 4: Average Perplexity vs Replacement Ratio

language models, of 20 examples with different hyper-parameters. As Figure 3 shows, the perplexity increases as the word distance increases in the greedy approach, meaning that the generated examples are less likely to occur in the corpus. However, in the global search approach, it is the opposite. It might be that the number of change decreases while allowing larger word distance. We also calculate the perplexity with different values of proportion of changes. The perplexity increases as the proportion of changes allowed in both attacking models. See result in Figure 4. When comparing the perplexity of the two attacking models, the global search approach generally does better than the greedy one. Since the greedy algorithm usually requires more changes than the global search approach even though we have set the threshold for the replacement rate.

3.2.4 Human Evaluation

We conduct a survey and propose two criteria to measure the performance of the adversarial examples from three attack methods. The first criteria is the sentiment classification accuracy of the adversarial examples which is predicted by human, and the second is the similarity between the adversarial examples and the original sentence.

Human Prediction Accuracy. Unlike adversarial examples in the context of image classification, natural language perturbation is generally perceptible since words are deleted, added, or replaced. Thus, we need to redefine imperceptibility in the context of natural language. Since we are crafting adversarial example to fool the sentiment classifier, we define it as imperceptible if human can still correctly classify without being fooled by the perturbation. Therefore, we ask some volunteers to evaluate adversarial examples and look at the percentage of responses that match the original classification.

Sentence Similarity. In addition to classification accuracy, we also care about how similar the generated adversaries examples are to the original unaltered sentences. we ask volunteers to rate the similarity, from 1 to 5, which means less similar to very similar between the adversarial sentence and original sentence.

We choose two sentences which originally classified as positive comment and negative comment, and then flipped to opposite sentiment result after applying three attack methods. After asking 15 volunteers and analyzing the survey data, we found that the adversarial example from Global Search is most similar to the original sentence, with an average similarity of 4.4, while the example from Greedy Search is less similar, with the score of 2.9.

Refer to human prediction accuracy, Global Search method also reaches the highest accuracy, which is 0.83. It means that although sentiment classifier make the wrong prediction on the Global

Search adversarial examples, human could still classify sentiment correctly. However, greedy search only has 0.30 accuracy because it alter too many words to fool the sentiment classifier. In this survey, some of the volunteers feel that the adversarial sentences from Greedy Search are hard to read and can not tell the sentiment of the senetence.

Examples of adversarial text generated
Original reviews: as long as you go into this movie knowing that it 's terrible : bad acting , bad " effects , " bad story , bad ... everything , then you 'll love it . this is one of my favorite " goof on " movies ; watch it as a comedy and have a dozen good laughs !
Global Search: as long as you go into this movie knowing that it 's terrible : worse acting , bad " effects , " bad story , bad ... everything , then you 'll love it . this is one of my favorite " goof on " movies ; watch it as a comedy and have a dozen good laughs yes
Greedy Search: as long as you leave into this blockbuster telling whether it 's horrendous : bad acting , bad " effects , " bad story , bad ... everything , then you 'll love it . this is one of my favorite " goof on " movies ; watch it as ...
Misspelling Noise: as lnog as you go into this mvoie knowing that it's terirble : bad atcing , bad " effects , " bad sotry , bad ... everything , then you 'll lvoe it . this is one of my favorite " goof on " movies ; watch it as a comedy and hvae a dzoen good laguhs !

4 Conclusion and Future Work

In this experiment, we generalize the concept of adversarial examples to the context of sentiment classification in natural language. We prove that some machine learning algorithms are vulnerable to adversarial examples. Some works have been done using the greedy approach and have proved the method to be effective; however, the global search algorithm is proved to be much more powerful than the greedy approach. The global search method requires less change to the original sentence and maintain a higher level of similarity in terms of both human evaluation and perplexity measure.

Both of the greedy and global search algorithms are operating in a white-box scenario; they are not as powerful as the black-box algorithms. The black-box character swapping algorithm could be further applied to CNN model with character-level embedding. Other word-level attacking models operating in black-box scenarios can be a way to improve the limitation of white-box models.

By far, we developed some successful strategies to attack the sentiment classifiers. Further studies can be done to strengthen the classifiers. By training the classifiers with generated adversarial examples, we hope to help defend the classifier against adversarial attack and improve the model accuracy.

References

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

Yonatan Belinkov and Yonatan Bisk. 2017. Synthetic and natural noise both break neural machine translation. *CoRR*, abs/1711.02173.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2017. Hotflip: White-box adversarial examples for NLP. *CoRR*, abs/1712.06751.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

Hossein Hosseini, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. 2017. Deceiving google's perspective API built for detecting toxic comments. *CoRR*, abs/1702.08138.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Volodymyr Kuleshov, Shantanu Thakoor, Tingfung Lau, and Stefano Ermon. 2018. Adversarial examples for natural language classification problems.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *CoRR*, abs/1607.02533.

	Misspelling Noise	Greedy Search	Global Search
Semantic Similarity (Scale: 1-5)	3.8	2.9	4.4
Human Prediction Accuracy	0.70	0.30	0.83

Table 3: Human Evaluation Survey Results

Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. 2018. Deep text classification can be fooled. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

S. Samanta and S. Mehta. 2017. Towards Crafting Text Adversarial Samples. *ArXiv e-prints*.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations*.