A New Black Box Attack Generating Adversarial Examples Based on Reinforcement Learning

Wenli Xiao
College of Electronic and Information
Wuhan University
Wuhan, Hubei, China
xiaowenli@whu.edu.cn

Hao Jiang
College of Electronic and Information
Wuhan University
Wuhan, Hubei, China
jh@whu.edu.cn

Song Xia
College of Electronic and Information
Wuhan University
Wuhan, Hubei, China
songxia@whu.edu.cn

***Abstract*—Machine learning can be misled by adversarial examples, which is formed by making small changes to the original data. Nowadays, there are kinds of methods to produce adversarial examples. However, they can not apply non-differentiable models, reduce the amount of calculations, and shorten the sample generation time at the same time. In this paper, we propose a new black box attack generating adversarial examples based on reinforcement learning. By using deep Q-learning network, we can train the substitute model and generate adversarial examples at the same time. Experimental results show that this method only needs $7.7ms$ to produce an adversarial example, which solves the problems of low efficiency, large amount of calculation and inapplicable to non-differentiable model.**

Keywords-adversarial examples; black box attack; adver-sarial reinforcement learning; deep neural network

I. INTRODUCTION

With the fast development of machine learning, people can do various things in many aspects, such as image recognition, increasing video pixels and so on [1]. Undoubtfully, it is really convenient and popular to achieve our goals with machine learning. However, as machine learning heavily relies on input data, the result can be easily influenced with the change of them [2]. Adversarial examples are classic examples. When we change just little of the input data, the output of a machine learning model will be different. e.g. An input figure is primarily classified by machine learning as a panda. When we add some noise on that, it will be highly classified as a gibbon [3]. And it is called adversarial examples attack.

Methods of adversarial attack can be classified as white box and black box [4]. White box attack is under the scene where attackers know the whole structure and all parameters of model that they are ready to attack. In other words, they clearly and totally know everything about the model [5]. The main algorithm that white box generating adversarial examples is to get the gradient of the model. This method has strong limitations as attackers usually don't know much about the model that they are going to attack and not all models are differentiable [6].

Black box attack can be described as the condition that attackers know little about the targeted mode [7]. Generally,the main algorithm is to make a substitute model with the output data from the targeted model, and then using adversarial examples generated by substitute model to attack the targeted model [8]. i.e. When attackers know the targeted model is Deep Neural Network (*DNN*), and they make another *DNN* to fit in the targeted model, so that they can use it to generate useful adversarial examples, which can successfully mislead the targeted model. As the adversarial examples generated by black box attack don't totally rely on the targeted model, they can fool a lot of different models. And that means black box attack has great portability.

However, black box attackers generally regard the generating adversarial examples as a white box attack after getting the substitute model, by exploiting the whole information about substitute model [9]. i.e. They will get the gradient of substitute model to generate adversarial examples to attack the targeted model. But as we referred above, not all models are differentiable, therefore, it may not be suitable for all substitute models to generate adversarial examples. Under this circumstance, we propose a method based on reinforcement learning to generate adversarial examples in black box attacks. The main contributions of this work are summarized as follows:

- We propose an improved method that generating adversarial examples based on reinforcement learning in black box attacks.
- We define the targeted Convolution Neural Networks (*CNNs*) as environment to train the substitute model, and in the process of training substitute *CNNs*, adversarial examples are also crafted. i.e. At the end of training, we will get the substitute model and adversarial model together.
- This paper proposes a dynamic feedback method to generate adversarial examples based on the substitute model instead of using static gradient based approaches to generate adversarial examples.
- This work improves the efficiency of generating adversarial examples, which can generate an adversarial example in less than 10ms. And it has great portability as it can fit in more models.

The rest of the paper is organized as follows: section II presents the related work about methods of generating adversarial examples in black box attack; section III introduces our system model; section IV identifies experiments we did;

978-1-7281-6776-3/20/$31.00 ©2020 IEEE

section V concludes our work and gives a prospect.

II. Related Work

There are many scholars working on black box attack and its way to generate adversarial examples. They correspondingly come up kinds of methods to generate adversarial examples after they get the substitute model. Some attackers generate the adversarial examples by obtaining the gradient of the substitute model, some attackers generate the confrontation samples by using *GAN* network, and others use other methods.

As for the fast gradient method, Nicolas Papernot, Patrick McDaniel and Ian Goodfellow [10] studied the transferability between deep neural networks and other models such as decision tree, kNN, etc. They use the fast gradient sign method introduced in [11] and the Jacobian-based iterative approach proposed in [12] for *DNN* to generate adversarial examples. Briefly speaking, they add noise on the input data under its perturbation remains in distinguishable to humans. And the noise is defined depend on the gradient of the targeted *DNN*. It actually obtained an error rate of an error rate of 90.4% with a confidence of 97.8% on fast gradient sign adversarial examples. But every time it is going to generate an adversarial example, it will calculate the gradient of model. That means it has a greatly high computational cost and doesn't fit in non-differentiable models. Huanqian Yan, Xingxing Wei, Bo Li [13] use FGSM and NES [14] to build attack algorithm and generate adversarial examples in black box attack. However, its average accuracy isn't high and it also can't apply on non-differentiable models.

For *GAN* method, Weiwei Hu and Ying Tan [15] present a method named Mal*GAN* to generate adversarial for black box attack based on *GAN*. It is able to decrease the detection rate to nearly zero and make the retraining based defensive method against adversarial examples hard to work when comparing with traditional methods. Although Mal*GAN* generates adversarial examples based on the dynamic feedback from the black box detector instead of using static gradient based approaches to generate adversarial examples, it has more complex architecture, more computation, and needs longer time to generate an adversarial example.

And for other methods, Ji Gao, Jack Lanchantin, Mary Lou Soffa and Yanjun Qi [16], [17] does this by making minor changes to important words to mislead the classifier, and they use the Levenshtein distance to measure the similarity between sequences. But the problem is that restricting the edit distance does not guarantee a cogent, it can't absolutely guarantee that the changes would not be discovered by human. Sizhe Chen, Zhengbao He, Chengjin Sun, Xiaolin Huang [18] aim to improve the transferability of black attck, so they use Attack On Attention(*AOA*) to produce adversarial examples to reduce query when attacking. Although its adversarial examples have high misleading rate, it consumes much time and lots of calculation when extracting common features of *DNNs*.

Our method proposes to generate adversarial examples by reinforcement learning when training the substitute model to solve these problems. Compared with methods above, it not only improves the generating efficiency, but also works for non-indifferentiable models. Importantly, we rely on the dynamic response of the targeted model to generate the counter examples, which avoid making big changes to the original data.

III. System Model

A. Framework of Our System

In this section, we will introduce the framework of our model, as shown in Fig. 1. In black box attack, the attacked model is mystery and the only information we can get is its output. To address the lack of information, we train a substitute model with the dynamic feedback from targeted model. And the function of substitute model is to generate adversarial examples aiming to attack the targeted model.

In our system, we choose *CNNs* as our targeted model. At the same time, considering the most common model is *CNNs* in the field of image recognition, we also set the structure of substitute model as *CNNs*. As for the algorithm, we use Deep Q-learning Network (*DQN*) [19] to train the substitute *CNNs* and generate adversarial examples together. The difference between *DQN* and Q-learning is that *DQN* uses deep learning networks to learn Q values online without the need to learn and store a huge Q table in advance.

As we all know, Q-learning rely on the Q-table and the formula to update Q. In (1), Q is the value function, s is the current state, a is the action to be taken in current state, α is the learning rate, γ represents the effect of future statues on the present , s' and a' shows the future of state and action.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)] \quad (1)$$

However, without the Q-table, it is a problem that we need to know the value of next state. *DQN* use deep learning network to solve this problem. *DQN* outputs $r + \gamma maxQ(s',a')$ as prediction, regards current value $Q(s,a)$ as label of current state and sets loss function as (2).

$$L(w) = E[(r + \gamma max_{a'} Q(s',a') - Q(s,a))^2] \quad (2)$$

In the experiment, we will draw an example from the example set and input it as a state in substitute *CNNs*. Then substitute *CNNs* will output the prediction $maxQ(s',a')$, and send current state and action to targeted *CNNs* to gain its output which is a reward r of current action. And the specific process of *DQN* is in algorithm 1.

The step of algorithm can be described as follows:

1. At first, we set up an experience pool and make its size N, aiming to avoid huge volatility caused by adjacent data. At the same time, we set state $s_t = x_t, \phi(s_t) = Q(s_t)$ and initialize Q and θ with random weights, where θ is the parameter of substitute *CNNs*.

2. Secondly, with learning rate ϵ choose action, with the probability of $1 - \epsilon$ choose random action, and execute $a_t = max_a Q'(\phi(s_t), a'; \theta)$.

3. Thirdly, setting $s_{t+1} = s_t, a_t, x_{t+1}, \phi_{t+1} = \phi(s_{t+1})$ and store them in experience pool D. Then, sampling random

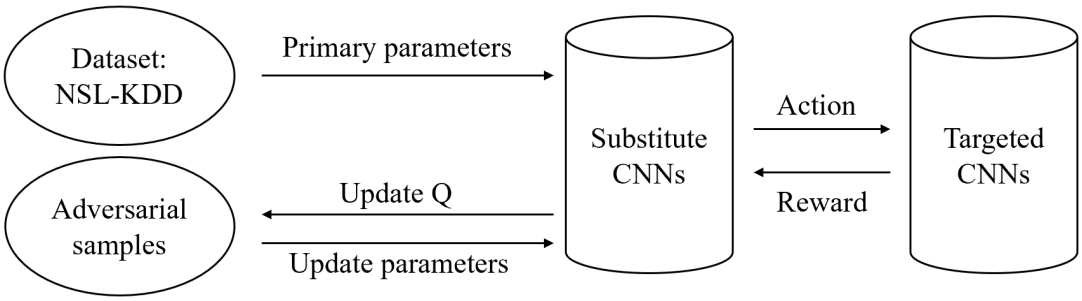

Figure 1. Framework of attacking.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights

For episode=1, M **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

For t=1, T **do**

With probability ε select a random action a_t

Otherwise select $a_t = max_a Q^*(\phi(s_t), a; \theta)$

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sample random minibatch of transition $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & for\ terminal\ \phi_{j+1} \\ r_j + \gamma max_{a'} Q(\phi_{j+1}, a'; \theta) & for\ non-terminal\ \phi_{j+1} \end{cases}$

Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ according to (2)

End for

End for

Algorithm2 total process of our system

Initialize set the structure of evaluation network to equal with $CNNs$

Initialize the state s and action a with value 0

For episode in max range

Randomly select one sample from the dataset

Set s=features of current sample

With the probability of ϵ to select a random a

Otherwise choose an a with the $max\ Q$

Change the features of the sample to the trained $CNNS$

Return the classification result l in $CNNS$

Compare l with the label

If true

Set $reward = 0$

Else

Set $reward = 1$

Return reward

Compare the probability of mis-scored

If error rate increases

Set $Trans_{reward} = 2$

If error probability>Correct probability

Set $Trans_{reward} = 5$

If error rate decreases

Set $Trans_{reward} = -2$

Return $Trans_{reward}$

Call algorithm 1

Update θ

End for

minibatch of transitions $(\phi_j, \alpha_j, r_j, \phi_{j+1})$ from experience D, and set output as $r_j + \gamma max_a(\phi_{j+1}, a'; \theta)$ if ϕ_{j+1} is terminal, otherwise set output as r_j.

4. Finally, performing a gradient descent step on loss function $(y_j - Q(\phi_j, a_j; \theta))^2$. Until loss function is minimized, we will get the parameters of substitute *CNNs* and adversarial examples.

B. The Details of Our Model

III-A is the key process of training alternative models and generating adversarial examples. III-B describes details about parameters' setting and total process of our system. The total process of our system including reward updating is in algorithm 2. The action represents the way to modify the feature of the example. And the value of the reward depends on whether the action will mislead the targeted model. At the beginning, we just set reward depending on the classified result,i.e. When assuming that the original classification was correct, if the example is classified wrongly, the reward will be one, if the example is classified right, the reward will be zero. However, it doesn't consider the probability of mis-scored but result. Therefore, we add settings of $trans_{reward}$ which is in direct proportion to the example's mis-scored probability. And Q update relies on (3).

$$q_{action} = trans_{reward} * q_{action} + 0.6 * q_{action} + \gamma * q_{prime} \quad (3)$$

We choose NSL-KDD [20] as our dataset, each record of it contains 42 different parameters, separated by the comma. The first 41 parameters are expressed as the features to describe the network connection. Those features can be divided into four categories: basic features of TCP connection (1-9), content features of TCP connections (10-22), time-based network traffic statistical features (23-31), and host-based network traffic statistical features (32-41). The last parameter indicates the type of connection. Excepting the normal records, others are different kinds of malicious connection, the threats coming from the internet. As every record has too many features, we just choose two features of them to modify for convenience. And the two features are 18^{th} and 21^{th} feature, whose names are *"dst_host_srv_diff_host_rate"* and *"dst_host_same_srv_rate"*.

Here is the actual action and reward we set. We totally set

TABLE I.
ACTION SETTING

Action	$18^{th} Feature$	$21^{th} Feature$
1	*"dst_host_srv_diff_host_rate"**0.9	*"dst_host_same_srv_rate"**0.9
2	*"dst_host_srv_diff_host_rate"**0.9	*"dst_host_same_srv_rate"**1.1
3	*"dst_host_srv_diff_host_rate"**1.1	*"dst_host_same_srv_rate"**0.9
4	*"dst_host_srv_diff_host_rate"**1.1	*"dst_host_same_srv_rate"**1.1

TABLE II.
REWARD SETTIG

Effect	***Reward***	$trans_{reward}$
Example is classified right	0	0
Example is classified wrong	1	0
Error probability increases	0	2
Error probability $>$ right probability	0	5
Error probability decreases	0	-2

four kinds of action to modify the two features. The action is like Table I. Action 1 is to modify the eighteenth feature to 0.9 times of the original, and the twenty-first feature to 0.9 times of the original. Action 2 is to modify the eighteenth feature to 0.9 times of the original, and the twenty-first feature to 1.1 times of the original. Action 3 is to modify the eighteenth feature to 1.1 times of the original, and the twenty-first feature to 0.9 times of the original. Action 4 is to modify the eighteenth feature to 1.1 of times the original, and the twenty-first feature to 1.1 times of the original.

As for the reward and $trans_{reward}$, the specific value corresponding to the effect is in Table II. Assuming that the original classification was correct, if the example is classified wrong, the reward will be 1; if the example is classified right, $trans_{reward}$ will be 0. And if the probability of classifier making an error increases, setting the $trans_{reward}$ to 2; if error rate becomes larger than right rate, setting the $trans_{reward}$ as 5; but if the probability of classifier making an error decreases, the $trans_{reward}$ will be -2.

After we get reward r, prediction $maxQ(s', a')$, label or current value $Q(s, a)$, we can update Q and parameters of substitute *CNNs*. Until the loss function is minimized, we have obtained the most confusing adversarial example and the most familiar substitute *CNNs* with targeted *CNNs*.

IV. EXPERIMENT RESULT

Our device *CPU* performance is *I7-6700HQ* quad-core processor, and *CPU* performance is *NVIDIA GTX980M 8G*. When we use dataset *NSL-KDD*, we can gain 6000 adversarial examples in $46.207s$. More intuitively talking, it only takes $7.7ms$ to generate one adversarial example. And here is the accuracy of the classifier's decision before and after the data set is modified to become an adversarial example. From Fig.2, we can find that when the dataset is before modification, the misleading rate of classifier is at 1% around, but after modification, it can up to 20% around. Although the absolute miscalculation rate was only 20%, the relative miscalculation rate increased by a factor of 20. From Fig.3, we can find

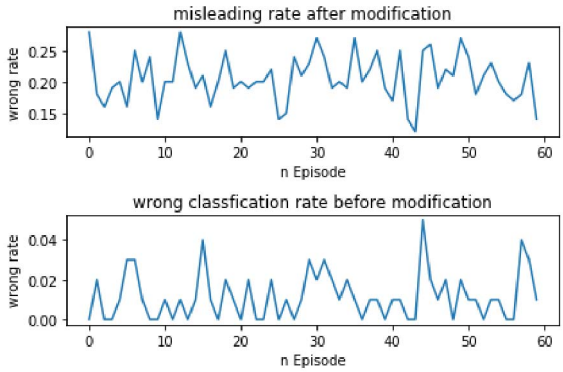

Figure 2. Misleading rate.

Figure 3. Total reward and loss.

that the total reward by episode, which means total wrong classification examples in one episode are around 300, and loss by episode, which means the gap between the model's prediction and the ideal state is around 1.0.

As for the generated adversarial examples, we pick a concrete example at random and features of the random data is in Fig. 4. In the process of generating adversarial example, it finally changes the 18th feature of this data from 3.1 to 0.5, and change its 21^{th} feature from 2.69 to 0.44, and the classification of classifier is as table III. We can learn that before modification the confidence of the adjudicator that the example is correct is 97.3%, however, after modification, the example is classified as wrong at 97% rate. It proves that the adversarial example can mislead the classifier at high rate.

A random data:

0.81	0.02	0.57	4.65	4.56	1.79	0.01	1.02	1.17	1.16	4.39	4.41	4.23
0.35	1.52	0	0.01	3.1	0.42	1.41	2.69	1.31	1.25	5.11	4.81	Label 0

After modification:

0.81	0.02	0.57	4.65	4.56	1.79	0.01	1.02	1.17	1.16	4.39	4.41	4.23
0.35	1.52	0	0.01	0.50	0.42	1.41	0.44	1.31	1.25	5.11	4.81	Label 1

Figure 4. Features of a random data.

Plus, we also use MINIST as dataset to do the same experiment. We tested the noise in three cases, producing $10,000$ counter examples in each case to determine the accuracy of

TABLE III.
CLASSIFICATION RATE OF THE RANDOM DATA

Data	Classification rate	
	Classified right	*Classified wrong*
Before modification	97.30%	2.70%
After modification	2.95%	97.00%

successful attacks. When the noise is obvious, or easily to be detected by human, the rate of successfully attacking rate is 70%, and the concrete adversarial example is posted in Fig.5. When the noise is moderate and the human eye can distinguish it, the rate of successfully attacking rate is 30%, and the concrete example is like Fig.6. When the noise is small and the human eye can not recognize it, the rate of successfully attacking rate is 10%, and the concrete example is like Fig.7.

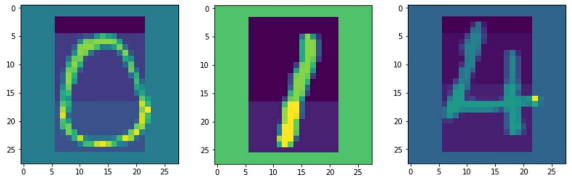

Figure 5. Adversarial example with high noise.

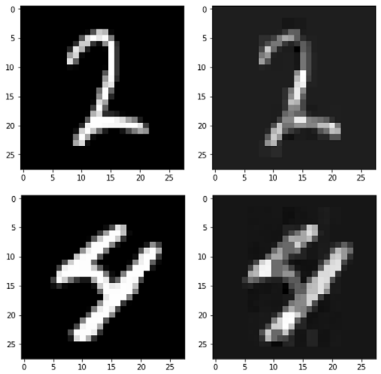

Figure 6. Adversarial example with moderate noise.

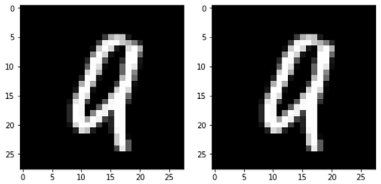

Figure 7. Adversarial example with small noise.

V. CONCLUSION

We propose a new method to generate adversarial examples based on reinforcement learning for black box attack. Its rate of generating adversarial example is $7.7ms$ for each. The high rate of example generation is conducive to the training of the defense network, which making up for the shortcomings of slow and heavy computation when generating adversarial examples in most black box attacks. Besides, because we train substitute *CNNs* and generate adversarial examples with *DQN* instead of getting the gradient of model, our method to generate adversarial examples for black box attack has greater portability as it also works on non-differentiable models.

However, we also have a lot of space for progress and development. First of all, the accuracy of the example attack is not high enough. Although successful attacking rate increased to 17.913 times, the absolute success rate isn't high. And we consider that is the result of modifying two features of input data. As Chaowei Xiao et.al [21] propose to generate adversarial examples with AdvGAN, which can not only produce adversarial example in less than 10ms, but also has accuracy of 92.3%, we know that we have to try to modify more features to improve our method. Apart from this, the optimization of model parameters is insufficient, so we will try to integrate with the existing attack methods to increase the vitality of the model.

REFERENCES

[1] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv e-prints*, p. arXiv:1607.02533, Jul 2016.

[2] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[4] A. Kurakin, I. Goodfellow, S. Bengio, Y. Dong, F. Liao, M. Liang, T. Pang, J. Zhu, X. Hu, C. Xie, J. Wang, Z. Zhang, Z. Ren, A. Yuille, S. Huang, Y. Zhao, Y. Zhao, Z. Han, J. Long, Y. Berdibekov, T. Akiba, S. Tokui, and M. Abe, "Adversarial attacks and defences competition," in *The NIPS '17 Competition: Building Intelligent Systems*, S. Escalera and M. Weimer, Eds. Cham: Springer International Publishing, 2018, pp. 195–231.

[5] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 506–519. [Online]. Available: https://doi.org/10.1145/3052973.3053009

[6] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples," *arXiv e-prints*, p. arXiv:1802.00420, Feb 2018.

[7] M. Cheng, T. Le, P.-Y. Chen, J. Yi, H. Zhang, and C.-J. Hsieh, "Query-Efficient Hard-label Black-box Attack:An Optimization-based Approach," *arXiv e-prints*, p. arXiv:1807.04457, Jul 2018.

[8] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Transactions on Neural Networks and Learning Systems*, Jan 2019. [Online]. Available: http://par.nsf.gov/biblio/10084436

[9] N. Akhtar and A. Mian, "Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey," *arXiv e-prints*, p. arXiv:1801.00553, Jan 2018.

[10] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Examples," *arXiv e-prints*, p. arXiv:1605.07277, May 2016.

[11] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014.

[12] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proceedings - 2016 IEEE European Symposium on Security and Privacy, EURO S and P 2016*, ser. Proceedings - 2016 IEEE European Symposium on Security and Privacy, EURO S and P 2016. United States: Institute of Electrical and Electronics Engineers Inc., 5 2016, pp. 372–387.

[13] H. Yan, X. Wei, and B. Li, "Sparse Black-box Video Attack with Reinforcement Learning," *arXiv e-prints*, p. arXiv:2001.03754, Jan 2020.

[14] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box Adversarial Attacks with Limited Queries and Information," *arXiv e-prints*, p. arXiv:1804.08598, Apr 2018.

[15] W. Hu and Y. Tan, "Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN," *arXiv e-prints*, p. arXiv:1702.05983, Feb 2017.

[16] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi, "Black-box generation of adversarial text sequences to evade deep learning classifiers." *2018 IEEE Security and Privacy Workshops (SPW), Security and Privacy Workshops (SPW), 2018 IEEE, SPW*, pp. 50 – 56, 2018. [Online]. Available: http://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.8424632&lang=zh-cn&site=eds-live

[17] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2013.

[18] S. Chen, Z. He, C. Sun, and X. Huang, "Universal Adversarial Attack on Attention and the Resulting Dataset DAmageNet," *arXiv e-prints*, p. arXiv:2001.06325, Jan 2020.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[20] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, July 2009, pp. 1–6.

[21] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song, "Generating Adversarial Examples with Adversarial Networks," *arXiv e-prints*, p. arXiv:1801.02610, Jan 2018.