Searching the Adversarial Example in the Decision Boundary

Haoyang Jiang[1], Qingkui Song[2], Julien Le Kernec[1,3,4]
[1]School of Information and Communication, University of Electronic Science and Technology of China, Chengdu, China
[2]College of Mathematics and Informatics, South China Agricultural University, Guangzhou, China
[3]ETIS – Signal and Information Processing lab, University Cergy-Pontoise, Cergy, France
[4]James Watt School of Engineering, University of Glasgow, Glasgow, UK
Haoyang.Jiang@std.uestc.edu.cn, sqk@stu.scau.edu.cn, Julien.lekernec@glasgow.ac.uk

***Abstract*—Deep learning technology achieves state of the art result in many computer vision missions. However, some researchers point out that current widely used deep learning architectures are vulnerable to adversarial examples. Adversarial examples are inputs generated by applying small and often imperceptible perturbation to examples in the dataset, such that the perturbed examples can degrade the performance of the deep learning architecture.**

In the paper, we propose a novel adversarial examples generation method. Adversarial examples generated using this method can have small perturbation and have more diversity compare to adversarial examples generated by other method.

I. Introduction

Deep learning architecture are widely used in machine learning tasks and achieve state of the art result. In image classification [1]–[3] and detection domain, they are able to get near human performance. Deep neural networks are also applied in natural language processing [4] and playing games [5].

However, researchers recently discovered that existing deep learning architecture are highly vulnerable to an attack called adversarial examples [6]. That is, those neural networks will misclassify examples that are slightly different from the examples drawn from the natural data distribution.

(a) Clean image (b) Adversarial example (c) Perturbation

Fig. 1. The original image is classified as panda with 99.65% confidence. The adversarial example is classified as badger with 46.84% confidence. Perturbation is the difference between two images; it is almost imperceptible, so we scale it 10 times for visible.

After that, many adversarial examples generating method are proposed for attacking deep neural network. For example, [7] points out that the existence of adversarial examples is related to the highly linear property of the neural networks and proposed a fast gradient sign method(FGSM) to find adversarial examples. [8] proposes an effective method to find the minimal perturbations in L_2 distance metric. [9] finds the perturbations by solving a optimization problem. This method can compute perturbations in L_p metric for any p. [10] discover the existence of a called *Universal adversarial perturbations*, which can cause the classifiers misclassify most of the inputs from the database. [11], [12] find perturbations in sparse form. [13] utilize the Generative Adversarial Networks framework to generate perturbations, this method can generate perturbation very fast once the generative network is trained.

On the other hand, some method of defensing against the attack of adversarial examples are proposed. Defensive distillation [14], [15] is proposed to reducing the success rate of adversarial attack; [16] finds that adversarial examples tend to generate noise in the feature layers of the neural network, so filtering the features in the middle layers can protect network from adversarial attack. [17] observes that unprotected neural networks have a highly curved loss function, so it proposed to train a neural network has less curvature loss function to defense from attacking. [18] suggests using adversarial examples generated from other model to adversarial train the neural networks can improve the adversarial robustness. But for the best of our knowledge, none of these methods can fully defense from all kinds of attacking. Defensing method design is still an open problem.

Some researches [19], [20] show that certain defense method will cause a phenomena called *gradient-masking* or *obfuscated gradients*, which is known to be an incomplete defense to adversarial examples. When using some white box attacking methods to attack such kind of protected neural networks, Gradient-masking will provide usefulness gradient around the sample points, so that attackers cannot find the correct adversarial examples following the gradient direction. However, these kind of methods do not push the decision boundary away from the sample points, so the neural networks are still vulnerable to adversarial attack.

In this paper, we propose a novel method to computed the adversarial examples without accessing the gradient around the sample points. We used experiments to demonstrate that the adversarial examples generated by our method have small perturbations and can fool the neural networks in high rate.

II. Attack Method

A. Build the Optimization Problem

Formally, for a given classifier $F(\cdot)$ and an input x, the adversarial example of input x can be found by solving the optimization problem

$$\begin{aligned} &\min \|r\|_p, \ s.t. \ F(x) \neq F(x+r) \\ &F(x) = \underset{k}{\operatorname{argmax}} \ f_k(x), \ \text{for } k = 1, 2, \ldots, K \end{aligned} \tag{1}$$

where the $f_k(x)$ is the network output corresponding for the k^{th} class; the $\|\cdot\|_p$ is the L_p distance, defined as

$$\|v\|_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}} \tag{2}$$

In this paper, we choose $p = 2$.

The original from is not in the standard form of optimization problem preventing us from using optimization method, so we should change the original problem in to another equivalent form, which is

$$\begin{aligned} &\min \|r\|_p, \ s.t. \ L(x+r) < 0 \\ &L(x+r) = f_i(x+r) - f_j(x+r) \\ &i = F(x), j \neq i \end{aligned}$$

i is the original label of the input x, j is any other label. For simplicity, we can choose j to be the most possible wrong label for input x, which is

$$j = \underset{k}{\operatorname{argmax}} \ f_k(x), \ \text{for } k = 1, 2, \ldots, K \text{ and } k \neq i$$

Assuming $\lim_{x \to x_0} L(x) = L(x_0)$, which means $L(\cdot)$ is a continue function, and $L(x) > 0$, it can be known that the solution r^{opt} for this optimization problem is near the boundary of the function $L(\cdot)$. That is $L(x + r^{opt}) = -\varepsilon$, and ε is a small positive number. So original optimization problem is transferred into the following form:

$$\begin{aligned} &\min \|r\|_p \\ &s.t. \ L(x+r) = f_i(x+r) - f_j(x+r) - \varepsilon = 0 \\ &i = F(x) \\ &j = \underset{k}{\operatorname{argmax}} \ f_k(x), \ \text{for } k = 1, 2, \ldots, K \text{ and } k \neq i \end{aligned} \tag{3}$$

Although we transfer the original optimization problem into a standard form, solving this optimization problem is still difficult, because the function $L(x+r) = f_i(x+r) - f_j(x+r) - \varepsilon$ is a highly non convex function. So instead of finding the global optimal solution, our method find a local optimal solution, and we use experiments to show that this suboptimal is small enough comparing adversarial example generated using existing method, and can fool the neural network in high rate.

B. Boundary Search Method

As we mention above, $L(x+r) > 0$ indicates input $(x+r)$ is the same class as x; $L(x+r) < 0$ indicates different class. The adversarial example can only exist in the decision boundary, which is $L(x + r) = 0$

1) Initial Point: Directly compute an input x' satisfying $L(x') = 0$ would be difficult. However, we can utilize the continuity of the function $L(\cdot)$ to compute it. Given an clean input x, it is obvious that $L(x) > 0$. Then we randomly select another input $\bar{x}$ from the dataset. As long as $\bar{x}$ is not the same class as x, $L(\bar{x}) < 0$ mast satisfied. Because $L(\cdot)$ is continue function, there must be an input $x' = \theta x + (1-\theta)\bar{x}$ satisfying $L(x') = 0$ for $0 < \theta < 1$. So we can use linear search or binary search to find the x'. And the initial perturbation is $r_0 = x' - x$.

Given a known starting point r_0, such that $L(x+r_0) = 0$,we need to find a new perturbation r_1 satisfying $L(x + r_1) = 0$ and $\|r_1\|_2 < \|r_0\|_2$

2) Linear Case: We first propose the algorithm for linear $L(\cdot)$, and then extend to nonlinear case. Given a linear function $L(x) = w^T x + b$, and a known starting point r_0, where $L(x + r_0) = 0$, the r_0 can be decomposed into two parts: $r_0 = r^{opt} + r^{step}$, where r^{opt} is the optimal solution of the optimization problem and is given by the closed-form formula:

$$\begin{aligned} r^{opt} &= \frac{{r_0}^T w}{\|w\|^2} * w \\ w &= \frac{\partial L(x + r_0)}{\partial (x + r_0)} \end{aligned}$$

Notice that w can be easily computed using existing deep learning machine learning framework.

r^{opt} is the projection of r_0 on the gradient direction $\nabla L(x + r_0)$, and r^{step} is the projection of r_0 on the decision boundary. Notice that r^{step} and r^{opt} are orthogonal to each other, so that for any $r_1 = r^{opt} + \alpha r^{step}$, $0 \leq \alpha < 1$,

$$\begin{aligned} \|r_1\|_2 &= \left\|r^{opt} + \alpha r^{step}\right\|_2 < \left\|r^{opt} + r^{step}\right\|_2 = \|r_0\|_2 \\ L(x + r_1) &= w^T(x + r_1) + b \\ &= w^T(x + r^{opt} + \alpha r^{step}) + b \\ &= w^T(x + r_0 + (\alpha - 1) r^{step}) + b \\ &= w^T(x + r_0) + (\alpha - 1) w^T r^{step} + b \\ &= L(x + r_0) + 0 \\ &= 0 \end{aligned}$$

The r_1 is proved to be a better solution than r_0. When $\alpha = 0$, r_1 degenerates to r^{opt}, which is the optimal solution.

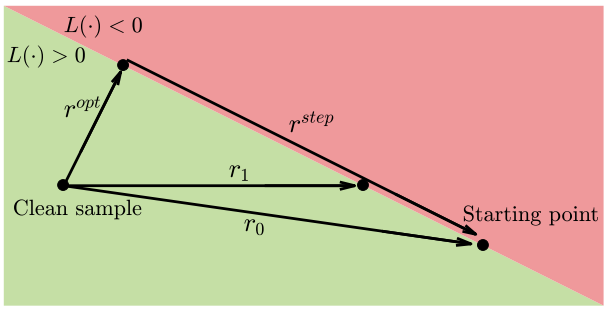

Fig. 2. Linear case

3) General Case: For most of widely used deep neural networks, the function $L(x)$ is highly non convex, but we can adopt the idea from the linear case: in each iteration, we find perturbation r_{i+1}, which is a better solution than the r_i, and finally it converges to a local optimal solution. The iterative formula is given by

$$\begin{aligned} r_{i+1} &= {r_i}^{opt} + \alpha {r_i}^{step} \\ &= r_i^{opt} + \alpha \left(r_i - r_i^{opt}\right) \\ &= \alpha r_i + (1-\alpha) {r_i}^{opt} \\ &= \alpha r_i + (1-\alpha) \frac{{r_i}^T w}{\|w\|^2} * w \\ w &= \frac{\partial L\left(x + r_i\right)}{\partial \left(x + r_i\right)} \end{aligned} \tag{4}$$

where $(1-\alpha)$ is the learning rate coefficient and should be smaller than 1. Large learning rate may cause no convergence; small learning rate will lead to slow convergence.

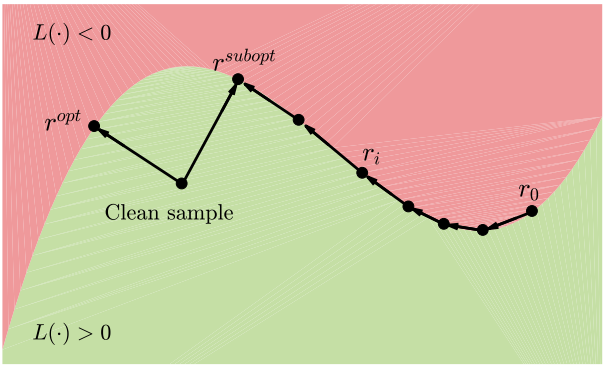

Fig. 3. Iterative method finding the adversarial examples for nonlinear $L(\cdot)$

The iteration stops when the norm of the perturbation convergence. Formally, it stops when

$$\|r_i\|_2 - \|r_{i+1}\|_2 < \sigma \tag{5}$$

for a small positive σ.

Although this the perturbation may converge to a local optimal solution, we use experiments to show that the local optimal solution is good enough to fool the classifier in high rate.

Algorithm 1 Boundary Search Attack Method

function BSA(x,L)
  binary search the starting point r_0
  $i \leftarrow 0$
  do
    $w \leftarrow w = \frac{\partial L(x+r_i)}{\partial(x+r_i)}$
    $r_{i+1} \leftarrow \alpha r_i + (1-\alpha)\frac{{r_i}^T w}{\|w\|^2} w$
    $i \leftarrow i + 1$
  while $\|r_i\|_2 - \|r_{i-1}\|_2 > \sigma$
  return r_i
end function

III. Experiment Results

A. Setup

We now test our attack algorithm on three deep convolutional neural networks architectures.

- LeNet: We trained a classic LeNet architecture applied on the MNIST dataset. The network achieves 99.1% accuracy.
- VGG16: We trained a small version of VGG16 network applied on the CIFAR10, achieving 88.63% accuracy.
- ResNet18: Small version of ResNet18 applied also on CIFAR10, achieving 89.9% accuracy.

In order to verify the effectiveness of our attacking method, we find adversarial example for images in the testing set. We compute the fooling rate and the average norm of the perturbation. A successful attack is defined as

$$\begin{aligned} \|r\|_2 &< \rho \\ F(x) &\neq F(x+r) \end{aligned} \tag{6}$$

The norm of the perturbation should be smaller than certain small value ρ in order to be imperceptible and the perturbation should change the predicted label of the original image. We choose $\rho = 3$ for MNIST dataset and $\rho = 1$ for CIFAR10 dataset.

We also perform the same experiments for FGSM, DeepFool(DF), Basic Iteration Method(BIM) for comparison.

B. Results

The experiments results are reported in Table I, II, and III. From the results, we can see that FGSM method cannot find the adversarial examples effective, because it is designed to be *fast* instead of finding the optimal adversarial examples. The perturbations found by BIM have the smallest average perturbations. However in our experiments we found that the BIM is the most time consuming method. Our method have the highest fooling rate in all three model, while keeping the average norm of the perturbations as small as the other three methods. But the smallest perturbation found by FGSM, DeepFool and BIM method are much better than ours, because these three method searching the adversarial examples starting from the sample points, which makes them easier to find the minimal perturbations.

TABLE I
LeNet-MNIST

Attack method	Fooling rate	Average norm	Min norm	Max norm
FGSM	0.32	1.84	0.048	2.99
DF	0.89	1.98	0.044	2.99
BIM	0.89	1.89	0.043	2.99
Our	0.98	1.96	1.24	2.99

TABLE II
VGG16-CIFAR10

Attack method	Fooling rate	Average norm	Min norm	Max norm
FGSM	0.67	0.28	2.8e-3	1.00
DF	0.83	0.28	8.4e-4	0.98
BIM	0.84	0.20	5.7e-6	0.99
Our	0.99	0.26	0.15	0.99

TABLE III
RESNET18-CIFAR10

Attack method	Fooling rate	Average norm	Min norm	Max norm
FGSM	0.76	0.25	2.7e-3	1.00
DF	0.99	0.21	1.3e-4	0.94
BIM	0.98	0.17	1.6e-4	0.97
Our	0.99	0.23	0.16	0.86

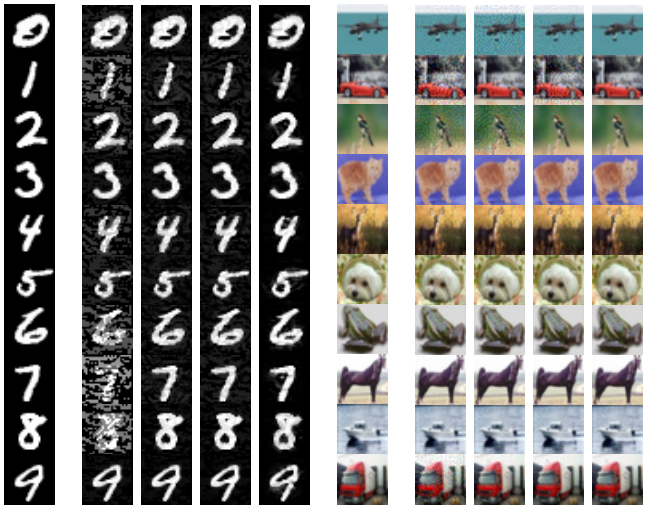

Fig. 4. An illustration of four attack methods applied on two datasets. The leftmost column contains the clean images. The next four columns show the adversarial examples generated by using FGSM, DeepFool, BIM and our method, respectively.

IV. CONCLUSION

In this paper, we propose a novel method to compute the adversarial examples without accessing the gradient around the sample points and we use experiments to demonstrate that the adversarial examples generated using our method have small perturbations and can fool the neural networks in high rate.

In the future research, we should focus on the effectiveness of our attacking method applying on the neural networks protected by defense method, accelerating the design of neural network with high robustness for adversarial attacking.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[4] D. Andor, C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and M. Collins, "Globally normalized transition-based neural networks," *arXiv preprint arXiv:1603.06042*, 2016.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[6] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[7] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[8] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. 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.

[9] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57, IEEE, 2017.

[10] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1765–1773, 2017.

[11] A. Modas, S.-M. Moosavi-Dezfooli, and P. Frossard, "Sparsefool: a few pixels make a big difference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9087–9096, 2019.

[12] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, 2019.

[13] O. Poursaeed, I. Katsman, B. Gao, and S. Belongie, "Generative adversarial perturbations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4422–4431, 2018.

[14] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 582–597, IEEE, 2016.

[15] N. Papernot and P. McDaniel, "On the effectiveness of defensive distillation," *arXiv preprint arXiv:1607.05113*, 2016.

[16] C. Xie, Y. Wu, L. v. d. Maaten, A. L. Yuille, and K. He, "Feature denoising for improving adversarial robustness," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 501–509, 2019.

[17] S.-M. Moosavi-Dezfooli, A. Fawzi, J. Uesato, and P. Frossard, "Robustness via curvature regularization, and vice versa," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9078–9086, 2019.

[18] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv preprint arXiv:1705.07204*, 2017.

[19] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *arXiv preprint arXiv:1802.00420*, 2018.

[20] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," *arXiv preprint arXiv:1611.03814*, 2016.