Report on *Evaluation and Design of Robust Neural Network Defenses*

Reporter: Duc-Anh Nguyen, PhD Student
VNU Vietnam National University

Disclaimer

This work is done based on my understanding of the chosen author' publications in particular and the foundation of machine learning in general. Therefore, what I presented here may be incorrect to some extent. If readers have any comment on my report, please feel free to contact me. The comment of other readers are valuable to me.

About author

Nicholas Carlini[1] is a research scientist at Google Brain working at the intersection of machine learning and computer security. His most recent line of work studies properties of neural networks from an adversarial perspective. He received Ph.D. from UC Berkeley in 2018, and B.A. in computer science and mathematics (also from UC Berkeley) in 2013. Generally, he is interested in developing attacks on machine learning systems; most of his work develops attacks demonstrating security and privacy risks of these systems. He has received best paper awards at ICML and IEEE S&P, and his work has been featured in the New York Times, the BBC, Nature Magazine, Science Magazine, Wired, and Popular Science. Previously he interned at Google Brain, evaluating the privacy of machine learning; Intel, evaluating Control-Flow Enforcement Technology (CET); and Matasano Security, doing security testing and designing an embedded security CTF.

From my perspective, both his research and my research interests are related to the topic "testing for AI systems". His academic background is really good. The chosen thesis is developed from the paper "Towards Evaluating the Robustness of Neural Networks" presented at IEEE S&P in 2017. With this paper, he received the best student paper award.

I am really execited to read the research of Nicholas Carlini et al. to strengthen my research background in this topic. The expectation includes but not limited to (1) mathematics, (2) the way he organized the thesis to convince the council, and (3) dig into new idea behind this thesis.

[1] https://nicholas.carlini.com

Contents

List of Figures

List of Tables

1 Introduction

Neural network has been used widely in solving many problems related to classification, ranging from speech processing [1], image processing [2], to medical diagnosis [3], etc. In most of the cases, if neural network models are trained with appropriate hyperparameters, the accuracy of these models usually outperform traditional machine learning teachniques such as Naive Bayes, K-nearest neighbors (KNN), etc. However, neural network models are usually unexplainable. Therefore, the testing of neural network models should be taken into account to ensure the correctness of these models.

Although the testing for neural network models is urgent and important, it has just been at the early stage from the research community and ML software companies [4]. *First*, to the best of my knowledge, to evaluate the reliability of neural network models for classification problem, most of ML researchers/developers usually make use of traditional criteria. Specifically, ML practitioners use precision, accuracy, F1-score, ROC, etc. to evaluate the quality of neural network models. However, these kinds of criteria just consider the input and output of neural network models without its internal structure. As a result, good results of traditional criteria do not mean that neural network models have no fault. Therefore, although traditional criteria seem to be effective to evaluate the reliability of neural network models, it is not enough. *Second*, the testing phase of neural network models is usually performed manually by ML practitioners most of the time. These practitioners give a little concern about how to test neural network models automatically. Even if they spend more efforts on testing neural network models automatically, there are only a limited set of automated testing tools for neural network models up to now.

To address the above problems, Nicholas Carlini et al. aimed to focus on testing the correctness of neural networks by attacking these models. The authors realized that neural network models, even these models are proved good enough, are prone to evasion attack. Evasion attack is a type of attack in which neural network models are fed with adversarial samples. In this context, evasion attack, known as *adversarial samples*, can be described as follows: Given a sample x (may be classified correctly or not by a neural network model) and a target class t, it is possible to find x' which is classified as class t, and x' is close to x. Term *close* means that there is no significant differences between x and x' in terms of norm distance such as $||L||_0$, $||L||_1$, $||L||_2$, or $||L||_\infty$.

From the given problem statement, the authors proposed three kinds of attacks, namely $||L||_2$ attack, $||L||_0$ attack, and $||L||_\infty$ attack. However, this report only focus on $||L||_2$ attack and $||L||_0$ attack which I understand the best and can spend more time on digging the implementation of these attacks. $||L||_2$ is highly recommended among the three proposed attacks. In order to evaluate the effectiveness of the proposed attacks, their experiment was performed on three type of applications including (i) breaking defenses of existing neural network models, (ii) speech recognition, and (iii) malware classification. The source code of the thesis is fully available here[2].

This remaining part of this report includes 7 sections. Firstly, the report presents related works which are synthesised from the resources I have covered so far, including the thesis of the authors. After that, I describe the problem formulation which is summarized in the thesis. The next section provides the background about the related techniques/mathematics to adversarial samples. The report continues with dataset section giving the description about the dataset used in the experiment, and the result of rebuit model made by me. The proposed attacks section presents the attacks which the author suggested. Finally, the report concludes in the last section.

2 Related Works

Currently, some works have been proposed for testing neural network model by several research groups. Focusing only on the most recent and closest ones, we can refer to [4], [8], [9], [10], [11], [12], and [13].

Pei et al. proposed DeepXplore [4] which is known as the first white-box testing technique for *deep neural network* (DNN) models. The authors introduced a new coverage criterion, namely neuron coverage, which is used to assert the correctness of DNN models. The authors also present a method to generate adversarial data set from the original training set to enrich the training set.

Ma et al. [8] claimed that many state-of-the-art AI systems suffer from vulnerabilities which may result in serve consequences when applied to real-world applications. The authors claimed that measuring the quality of deep learning systems in terms of traditional measurements (i.e., accuracy, precision, or F1 score) is not enough. Therefore, the authors proposed a set of coverage criteria to evaluate the correctness of DNN models at the unit level,

[2] https://github.com/carlini/nn_robust_attacks

the integration level, and the system level. The authors made a comparison between neuron coverage presented in DeepXplore [4] and their proposed criteria. As a result, the proposed criteria are better than neuron coverage.

Tian et al. proposed Deeptest [11] to test automatically the quality of DNN models used in autonomous cars. The authors aimed to find out erroneous behaviours of DNN-driven vehicles that can potentially lead to fatal crashes. In the experiment, DeepTest found thousands of erroneous behaviours under different realistic driving conditions (e.g., blurring, rain, fog, etc.).

Sun et al. [12] suggested the application of concolic testing in DNN models evaluation. The authors utilized neuron coverage, modified condition-decision coverage, and neuron boundary coverage to check the quality of DNN models.

Zhang et al. proposed DeepRoad [10] to automatically testing the consistency of DNN-based autonomous driving systems. DeepRoad tries to generate driving scenes under various weather conditions to enrich the training data. To do that, the authors apply Generative Adversarial Networks (GANs) [14]. The experiments have shown that DeepRoad can detect thousands of inconsistent behaviours to improve the robustness of ML systems.

Ma et al. proposed DeepCT [9] to evaluate the robustness of deep learning systems. DeepCT stands for deep combinatorial testing. The authors also introduce a set of criteria for testing deep learning systems. With the proposed coverage criteria, DeepCT tests the interactions of input to reduce the size of the test suite rather than exploring all the possible combinations of input space.

Li et al. investigated the advantages and disadvantages of structural coverage criteria [13]. The study has shown the inconsistency between the distribution of adversarial examples and the distribution of samples. Therefore, the current coverage criteria should be improved to reduce this inconsistency and increase the robustness of adversarial examples generation.

3 Problem formulation

The author started from the original formula of adversarial sample generations, proposed by Szegedy et. al. [7], defined as follows:

Given a n-dimensional vector $x \in \mathbb{R}^n$ in which x is classified incorrectly: $C(x) \neq t$, where t is the true class of x. We need to find $x' = x + \delta \in \mathbb{R}^n$

such that x' is valid and predicted correctly while x' is close to x as follows:

$$\begin{aligned} \text{minimize } & D(x, x+\delta) \\ \text{such that } & x+\delta \in [0..1]^n \\ \text{and } & C(x+\delta) = t \end{aligned} \tag{1}$$

However, the formula (1) is hard to solve in practice due to the high level complexity of $C(x + \delta) = t$. In fact, $C(x + \delta) = t$ is highly non-linear which struggles to solve. Therefore, Szegedy et. al. suggested that formula (1) should be redefined to make it easier to solve.

Normally, formula (1) is that $C(x+\delta) = t$ is removed while adding a new formula $f(x + \delta) <= 0$: $C(x + \delta) = t$ if and only if $f(x + \delta) <= 0$.

$$\begin{aligned} \text{minimize } & D(x, x+\delta) \\ \text{such that } & f(x+\delta) <= 0 \\ \text{and } & x+\delta \in [0..1]^n \end{aligned} \tag{2}$$

In formula (2), we have many choices to the function f, for example, cross-entropy function.

Szegedy et. al. used a similar approach by proposing that formula (2) is written in another way as follows:

$$\begin{aligned} \text{minimize } & D(x, x+\delta) + c \cdot f(x+\delta) \\ \text{such that } & x+\delta \in [0..1]^n \end{aligned} \tag{3}$$

, where $c > 0$ is a constant and $f(x + \delta) <= 0$.

4 Background

This section provides background on adversarial samples generation mentioned in the original paper. Firstly, this report describes $||L||_p$ norm which is used to compute the distance between two k-dimensional vectors. After that, the detail description of standard gradient descent and its variants are presented. Finally, the report covers box-constraints technique used to generate valid images.

4.1 $||L||_p$ norm

Let $D(x, x') \in [0, \infty]$ be the distance between sample $x \in \mathbb{R}^n$ and $x' \in \mathbb{R}^n$ in which x' is an adversarial example of x. $D(x, x') = 0$ if and only if both x and x' are identical.

The are many ways to calculate the distance between x and x', known as $||L_p||$ norm. The popular types of distance caculation are L_0 (p=0), L_2 (p=2), and L_∞ (p=∞).

The formula of $||L_p||$ is defined as follows:

$$||L_p|| = (\sum_{i=0}^{n-1}(x_i - x'_i)^p)^{1/p}$$

$||L_0||$ distance is interpreted as the number of different pixels when comparing between x and x'.

$||L_2||$ distance is known as Euclidean distance. One major weakness of $||L_2||$ distance is that it may be a small value even when there are many changes in value of pixels between x and x'.

$||L_\infty||$ distance measure the maximum change in value of pixels in every axis of the coordinate. Goodfellow et al. claimed that this metric is optimal to use in adversarial sample generation [6].

$$||L_\infty|| = max(|x_0 - x'_0|, |x_1 - x'_1|, .., |x_{n-1} - x'_{n-1}|)$$

4.2 Gradient Descent

Given the objective function $f(x)$, find x to minimize $f(x)$. Gradient descent is a effective technique to seek for the x. This section covers standard gradient descent and its variants known as *momentum gradient descent* and *nesterov acclerated gradient*.

4.2.1 Standard gradient descent

The main idea of *standard gradient descent* is that x is modified iteratively in the opposite direction of $f'(x)$. The formula of the standard gradient descent can be written as follows:

$$x_{i+1} = x_i - \eta * f'(x)(i >= 0) \tag{4}$$

, where η is learning rate, x is a k-dimensional vector, $x_0 = x$.

In formula (4), learning rate η is a hyperparameter and should be fine-tuned. There are several main points about the formula (4).

- Initialize η. Usually, η is initialized at a relatively small positive value, e.g., $\eta = 0.01$ or $\eta = 0.001$. If the value of η is too small, the process of gradient descent may take larger number of iterations, leading to a waste of time. Otherwises, in the case that the value of η is large, gradient descent process may go over the optimal value of x due to the large jump of x.

- Constant η. Depending on the type of gradient descent technique, the value of learning rate η is a constant (e.g., standard gradient descent, Nesterov momentum) or can be changed during gradient descent process (e.g., step decay, exponential decay, $1/t$ decay).

- Sign of $f'(x)$. There are two cases happening with the sign of $f'(x)$:
  - If the value of derivative function $f'(x)$ is positive, $f'(x)$ is going up. In this case, in order to minimize $f(x)$, the value of x should be reduced by a small amount.
  - If the value of derivative function $f'(x)$ is negative, $f'(x)$ is going down. In this case, in order to minimize $f(x)$, the value of x should increase further.

4.2.2 Momentum gradient descent

Momentum gradient descent improves standard gradient descent by converging to the optimal x faster.

$$\begin{aligned} v_t = \mu * v_{t-1} + \eta * f'(x) \\ x = x - v_t \end{aligned} \tag{5}$$

, where η is learning rate, x is a k-dimensional vector.

Equivalently, the formula (5) can be written as follows:

$$x = x - (\mu * v_{t-1} + \eta * f'(x)) \tag{6}$$

Comparing to formula (4), formula (8) adds $\mu * v_{t-1}$ representing the previous change of x, called *momentum*. Momentum $\mu * v_{t-1}$ is accumulated over iterations.

4.2.3 Nesterov accelerated gradient

Nesterov accelerated gradient (NAG) improves momentum gradient descent by converging to the optimal x faster. In order to do that, NAG will look ahead one iteration.

$$\begin{aligned} v_t = \mu * v_{t-1} + \eta * f'(x - \mu * v_{t-1}) \\ x = x - v_t \end{aligned} \tag{7}$$

Equivalently, the formula (7) can be written as follows:

$$x = x - (\mu * v_{t-1} + \eta * f'(x - \mu * v_{t-1})) \tag{8}$$

4.3 Box Constraints

The essense of adversarial sample generations is that the original sample x is modified into $x' = x + \delta$. This modification may lead to an invalid sample x'. Therefore, we need to align x' to the available range of value. In this thesis, the available range of each pixel is in [0..1].

The author investigated there method to deal with this problem:

- *Projected gradient descent.* At each step of the standard gradient descent, all the pixels out of available range is cut off.

  This strategy has a limit: The cut-off pixel is the input of the next step in standard gradient descent, the result of adversarial sample generation may be affected.

- *Clipped gradient descent.* Rather than performing the normalization of pixel value at each step of SGD, the clipping step is put into funtion $f(x + \delta)$. Specifically, they replace $f(x + \delta)$ with $f(min(max(x + \delta, 0), 1))$.

- *Changes of variables.* Instead of using variable δ, they use a new variable w as follows:

  $$\delta = 1/2 * (tanh(w) + 1) - x \tag{9}$$

  , where w, δ, x is a n-dimensional vector.

  Because $tanh(w) \in [-1, 1]^n$, $1/2 * (tanh(w) + 1) \in [0, 1]^n$; hence $x + \delta \in [0, 1]^n$. This formula (**??**) ensures that x' is a valid image.

Definition 1. *Objective function*

5 Dataset

The authors used the two primary datasets: MNIST, CIFAR to evaluate their proposed attacks. In this report, I just use dataset MNIST to reevaluate their experiments. I re-run the training process of MNIST dataset and report the results below.

5.1 MNIST

MNIST is a dataset of digit recognizer which can be found officially on Kaggle[3]. The training set contains 60,000 samples. The test set has 10,000 samples. Each sample on the dataset is an image with 28 pixel in width and 28 pixel in height. The value of each pixel is in range of 0 and 255, which indicates the lightness or darkness of that pixel.

5.2 CNN model

Table 1 illustrates the configuration of cnn model used to train MNIST. Figure 1 described the corresponding implementation of this model in python. For example, starting with the input having shape *(size of batch, height = 28, width = 28, n_channel = 1)*, this input is passed through a convolution layer with kernel *(3 x 3)*, 32 filters, and *valid* padding. A convolution layer with *valid padding* option ensure that the kernel of convolution layer only move in the input (i.e., not go out of the border). Therefore, the output of convolution layer with this padding has:

$$width\_output = width\_input - width\_kernel + 1 = 28 - 3 + 1 = 26, \text{ and}$$
$$height\_output = height\_input - height\_kernel + 1 = 28 - 3 + 1 = 26$$

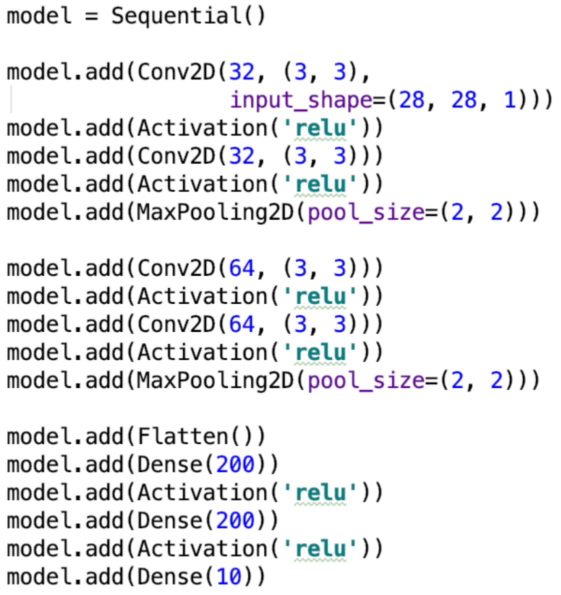
Therefore, the output of the first convolutional layer has shape (width = 26, height = 26, n_filters = 32).

[3] https://www.kaggle.com/c/digit-recognizer/data

Table 1: CNN architecture of MNIST

Layer Type	**MNIST**
Convolution + ReLU activation	kernel (3x3), 32 filters
Convolution + ReLU activation	kernel (3x3), 32 filters
Max Pooling	kernel (2x2)
Convolution + ReLU activation	kernel (3x3), 64 filters
Convolution + ReLU activation	kernel (3x3), 64 filters
Max Pooling	kernel (2x2)
Fully Connected + ReLU activation	200
Fully Connected + ReLU activation	200
Softmax	10

Figure 1: The implementation of CNN architecture used to train MNIST



```
model = Sequential()

model.add(Conv2D(32, (3, 3),
                 input_shape=(28, 28, 1)))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dense(10))
```

5.3 Training

The model is trained with the following fine-tuning hypeparameters: learning rate $\eta = 0.01$, decay = 1e-6, momentum = 0.9. To overcome optimal solution during batch gradient descent, they applied Nesterov momentum. The training process is taken in 50 iterations with batch = 128 samples. Like the orginal experiment, there is no data-set augmentation in the re-evaluation.

The achieved accuracy of CNN model is 0.9985%. While figure 2 shows the accuracy over iterations when training MNIST, figure 3 illustrates the value of loss function over 50 iterations.

Figure 2: Accuracy over iterations when training MNIST

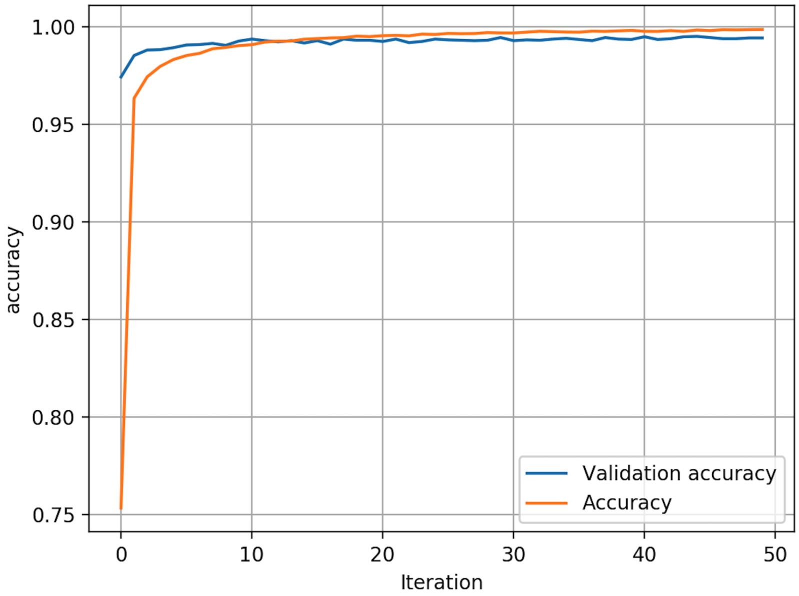
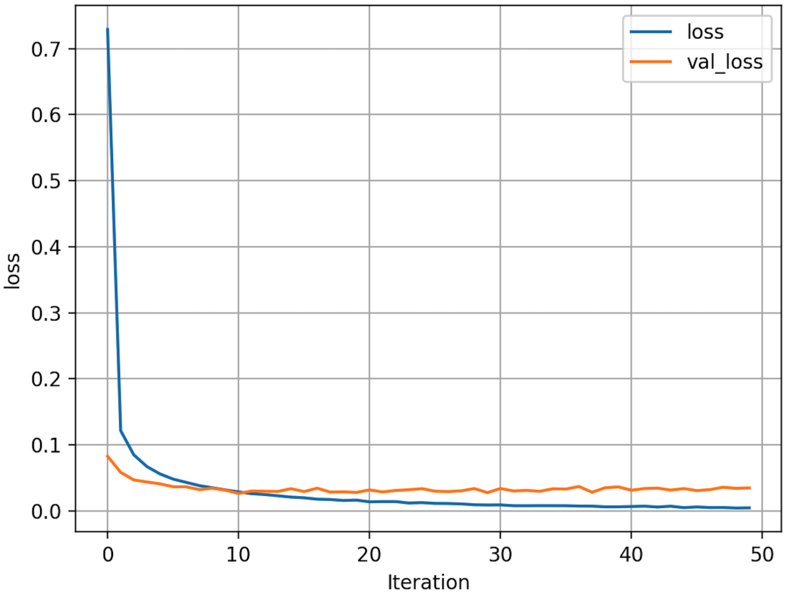


Figure 3: Loss over iterations when training MNIST



6 The proposed attacks

The problem statement is as follows:

$$\begin{aligned} \text{minimize } & D(x, x+\delta) + c \cdot f(x+\delta) \\ \text{such that } & x+\delta \in [0..1]^n \end{aligned} \tag{10}$$

, where $c > 0$ is a constant, $f(x+\delta) <= 0$, x is the original sample, $x+\delta$ is a adversarial sample; δ is the difference between x and x', D is norm distance.

For simplicity, the problem statement is rewritten as follows:

$$\begin{aligned} \text{minimize } & ||\delta||_p + c \cdot f(x+\delta) \\ \text{such that } & x+\delta \in [0..1]^n \end{aligned} \tag{11}$$

In the above objective function (12), there are many formula of function $f(x+\delta)$ which have been proposed by many different research groups. In these formula, $f_1(x')$ is the most popular function f, which is known as cross-entropy function.

$$
\begin{aligned}
f_1(x') &= -\mathrm{loss}_{F,t}(x') + 1 \\
f_2(x') &= (\max_{i \neq t}(F(x')_i) - F(x')_t)^+ \\
f_3(x') &= \mathrm{softplus}(\max_{i \neq t}(F(x')_i) - F(x')_t) - \log(2) \\
f_4(x') &= (0.5 - F(x')_t)^+ \\
f_5(x') &= -\log(2F(x')_t - 2) \\
f_6(x') &= (\max_{i \neq t}(Z(x')_i) - Z(x')_t)^+ \\
f_7(x') &= \mathrm{softplus}(\max_{i \neq t}(Z(x')_i) - Z(x')_t) - \log(2)
\end{aligned}
$$

, where s is the correct classification; $(e)_+$ is short-hand for $max(e, 0)$; $softplus(x) = log(1 + exp(x))$.

The authors took expriments to find the best function among $f_1(x')$ to $f_6(x')$. They conclude $f_6(x')$ showed the most significant result compared to the other functions.

Hence, the problem statement is refined as follows:

$$
\begin{aligned}
\text{minimize } ||\delta||_p + c \cdot max(\max_{i \neq t}(Z(x')_i) - Z(x')_t), 0) \\
\text{such that } x + \delta \in [0..1]^n
\end{aligned}
\tag{12}
$$

The authors made use of "change of variable" method (formula 9) which replaces $x + \delta$ with w.

$$
\begin{aligned}
\text{minimize } ||1/2 * (tanh(w) + 1) - x||_p + \\
c \cdot max(\max_{i \neq t}(Z(1/2 * (tanh(w) + 1))_i) - Z(1/2 * (tanh(w) + 1))_t), 0) \\
\text{such that } 1/2 * (tanh(w) + 1) \in [0..1]^n
\end{aligned}
\tag{13}
$$

6.1 $||L||_2$ attack

Given the formula (13), the author tried to generate adversarial samples. The original idea of $||L||_2$ attack is as follows:

- *Step 0.* Start with target class t, the original sample x, and the neural network model N trained before; $C^*(x)$ is the true class of x; $C(x) \neq t$ is the predicted class of x given model N.

- *Step 1.* Calculate the derivative of the objective function (13) with respect to w. Denote the objective function as G, we need to compute $G'(w)$.

- *Step 2.* Use standard gradient descent to compute $x_{i+1} = x_i - \eta * G'(w)$ iteratively, where constant η is learning rate. At each iteration, compute the prediction of x_{i+1}. If $C(x_{i+1}) = t$, the algorithm terminates and report on x_{i+1}, denoted by x'.

The above $||L||_2$ attack has a major disadvantage. The returned x' may be a local optimal solution which is caused by the chosen learning rate η. Since the research just used standard gradient descent, choosing small value of η is just enough though it may take more iterations.

Instead of changing η, the authors proposed *multiple starting-point gradient descent*. The idea of this technique is that rather than starting at x, $||L||_2$ attack starts with x^* where x^* is close to x. To reduce the cost of adversarial sample generation, the number of iterations in standard gradient descent is fixed since there are many similar point x^*.

An example of $||L||_2$ attack on MNIST is shown in Fig. 4. The result of adversarial sample generation for s digit include three parts: (i) a source digit, (ii) target digit, and (iii) an adversarial image. Almost all attacks are visually indistinguishable from the original digit.

6.2 $||L||_0$ attack

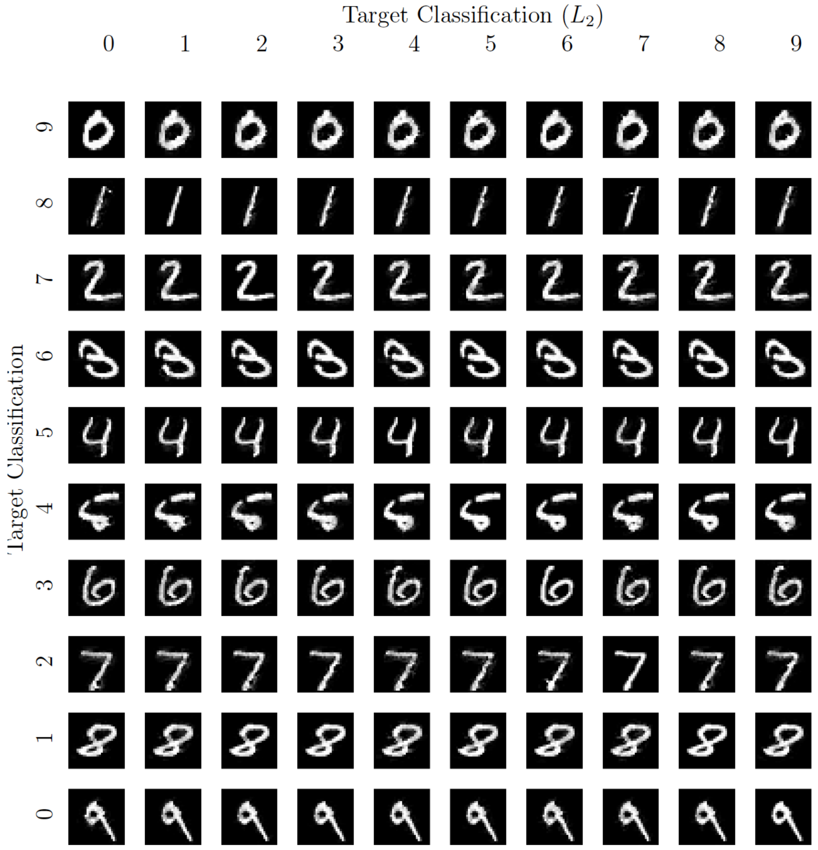
Because $||L||_0$ is non-differentiable, therefore, the authors proposed an interative algorithm to perform $||L||_0$ attack.

Rather than computing the derivative of both terms in formula (13, they just considered the second terms. The first term $||L||_0$ is ignored.

- *Step 0.* Start with target class t, the original sample x, and the neural network model N trained before; $C^*(x)$ is the true class of x; $C(x) \neq t$ is the predicted class of x given model N. Denote the allowed set S containing all pixels in x. All the pixels out of S are fixed (can not be changed). S only contains the pixels which can be changed to generate adversarial samples x'.

- *Step 1. Shrink the allowed set S.* Perform $||L||_2$ attack above with respect to the pixels in S. There are two cases happening:

Figure 4: Perform $||L||_2$ attack on MNIST



- Case 1: Can not find adversarial sample. In this case, the algorithm terminates.

- Case 2: Find an adversarial sample x_{i+1}. Remove the pixel j-th in S which has the least influence on creating adversarial sample x'

$$j = \underset{j \in S}{\operatorname{argmin}}(f'_6(x_{i+1})_j * (x_{i+1} - x)_j) \tag{14}$$

, where $(f'_6(x_{i+1})_j * (x_{i+1} - x)_j)$ tell us how much the reduction to $f'_6(x_{i+1})$ when moving x to x_{i+1}; $f'_6(x_{i+1})$ is a 1-D vector when computing the value of $f'_6(x)$ at adversarial sample x_{i+1}; pixel j-th in S has the change per unit $f'_6(x_{i+1})_j$.

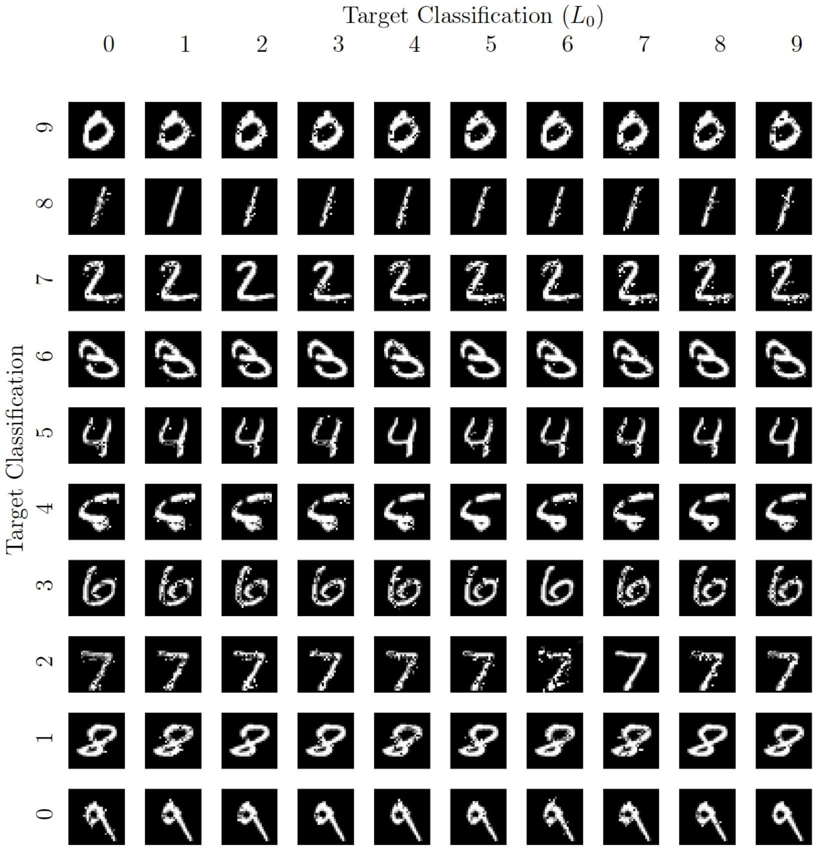
For the pixel j-th having the least reduction, they removed it from S. They argued that this pixel does not have much influence on classifier output.

Then repeat step 1.

When performing $||L||_0$ attack, the value of constant c is initialized to a very low value (e.g., 10^{-4}). The threshold of c should be high ()e.g., 10^{10}]). At step 1, if $||L||_2$ attack can not found any adversarial sample, the value of c is double until it reaches the limitation.

An example of $||L||_0$ attack on MNIST is shown in Fig. 6.2. The attacks are east to realize the diference because the $||L||_0$ attack is more difficult than $||L||_2$.

Figure 5: Perform $||L||_0$ attack on MNIST



7 Conclusion

The report covers the two efficient adversarial attacks: $||L||_0$ attack and $||L||_2$ attack. These types of attack aim to generate new sample from original samples in dataset which make wrong prediction. $||L||_0$ attack is an iterative algorithm which includes $||L||_2$ attack. The author stated that $||L||_2$ attack is better than $||L||_0$ attack via the expriment. Given an original sample and a target, $||L||_2$ attack generate an adversarial sample which is closer to the original sample than $||L||_0$ attack.

In order to have a better understanding about their proposals, I re-evaluate their experiments and dig deep into their source code. This thesis enriched my background about adversarial example generation in particular and machine learning in general. I am planning to take the following steps: (i) evaluate the quality of generated adversarial samples by using the current criteria such as neuron coverage [4], kmnc/sbc/snac/tknc coverage [8]. I am implementing this kinds of criteria and push to github[4].

References

[1] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal Processing Magazine*, vol. 29, pp. 82–97, Nov 2012.

[2] D. Ciregan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3642–3649, June 2012.

[3] D. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, “Deep neural networks segment neuronal membranes in electron microscopy images,” in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 2843–2851, Curran Associates, Inc., 2012.

[4] K. Pei, Y. Cao, J. Yang, and S. Jana, “Deepxplore: Automated whitebox testing of deep learning systems,” *CoRR*, vol. abs/1705.06640, 2017.

[4] https://github.com/ducanhnguyen/mydeepconcolic

[5] H. F. Eniser, S. Gerasimou, and A. Sen, “Deepfault: Fault localization for deep neural networks,” *CoRR*, vol. abs/1902.05974, 2019.

[6] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. D. McDaniel, “Adversarial perturbations against deep neural networks for malware classification,” *CoRR*, vol. abs/1606.04435, 2016.

[7] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” in *International Conference on Learning Representations*, 2014.

[8] L. Ma, F. Juefei-Xu, J. Sun, C. Chen, T. Su, F. Zhang, M. Xue, B. Li, L. Li, Y. Liu, J. Zhao, and Y. Wang, “Deepgauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems,” *CoRR*, vol. abs/1803.07519, 2018.

[9] L. Ma, F. Juefei-Xu, M. Xue, B. Li, L. Li, Y. Liu, and J. Zhao, “Deepct: Tomographic combinatorial testing for deep learning systems,” in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 614–618, Feb 2019.

[10] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, “Deeproad: Gan-based metamorphic autonomous driving system testing,” *CoRR*, vol. abs/1802.02295, 2018.

[11] Y. Tian, K. Pei, S. Jana, and B. Ray, “Deeptest: Automated testing of deep-neural-network-driven autonomous cars,” *CoRR*, vol. abs/1708.08559, 2017.

[12] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, “Concolic testing for deep neural networks,” *CoRR*, vol. abs/1805.00089, 2018.

[13] Z. Li, X. Ma, C. Xu, and C. Cao, “Structural coverage criteria for neural networks could be misleading,” in *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER ’19, (Piscataway, NJ, USA), pp. 89–92, IEEE Press, 2019.

[14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,”

in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, (Cambridge, MA, USA), pp. 2672–2680, MIT Press, 2014.