A Naive Study of Adversarial Examples

Wenjie Richie Zi

September 2017

1 Introduction

Convolutional Neural Networks(ConvNets) is a class of deep, feed-forward artificial neural networks that has successfully been applied to lots of computer vision related problems.
As pointed out in [5], lots of research on ConvNets reach a near "human-level performance", it is possible to change it almost imperceptibly to the human eye to fool ConvNet suddenly classifies the image as any other class of choice. Lots of interesting questions need to be answered:

- how do adversarial images fool ConvNets?
- how to generate adversarial images to fool ConvNets?
- knowing the existing problem, what can we do to improve the model being more robust to adversarial examples?

In that report, we will talked the data being used in section 2; we will cover some fundamental knowledge in section 3; in section 4, we will show the preliminary experiments we have done and dive into the reasons behind it; after that, we will have a better look at the results and discuss about the insights or future learning plans based on the problems we pinpointed, also the conclusions and references.

2 Dataset

In terms of dataset, this study is based on MNIST[7] dataset of handwritten digits. Examples can be seen at Figure 1. It contains a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in 28 * 28 image, and target classes are from 0 to 9.
Each of the pixel in the image is a number between 0 to 1 which represents the grey level of the grid.

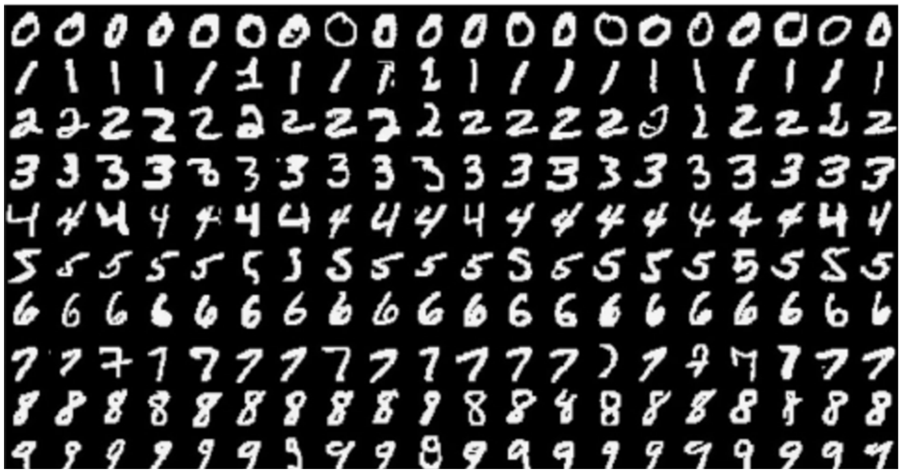

Figure 1: Examples of MNIST Dataset

3 Background

3.1 Convolutional Neural Networks

Convolutional Neural Networks are a specialized kind of neural network for processing data that has a known grid-like topology The name "Convolutional Neural Network" indicates that the network employs a mathematical operation called convolution which is a specialized kind of linear operation.[2] A typical layer of a convolutional network consists of three stages. In the rst stage, the layer performs several convolutions in parallel to produce a set of linear activation functions. In the second stage, each linear activation is run through a nonlinear activation function, such as the rectied linear(ReLU) activation function. In the third stage, we use a pooling function to output of the net at a certain location with a summary statistic of the nearby outputs, such as max pooling.

3.2 Adversarial Examples

3.2.1 What are Adversarial Examples

Adversarial examples are inputs formed by applying small but intentionally worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high confidence.[4]
Recent research even shown that after printing adversarial examples on normal paper and then photographed with a standard resolution smart phone, it can still cause a classifier mislabel the image[6].
Ignoring adversarial examples can be pretty dangerous. For example, attackers

could target autonomous vehicles by using stickers or paint to create an adversarial stop sign that the vehicle would interpret as a 'yield' or other sign, as discussed in [8].

3.2.2 How to Generate Adversarial Examples

The most used simple method to generate adversarial examples is "fast sign gradient method[4]:

$$x^{adv} = x - \epsilon(sign(\nabla(x, y_{true})) \tag{1}$$

This method reliably causes a wide variety of models to misclassify the input. Two other methods introduced in [6] including a iterative version of sign gradient

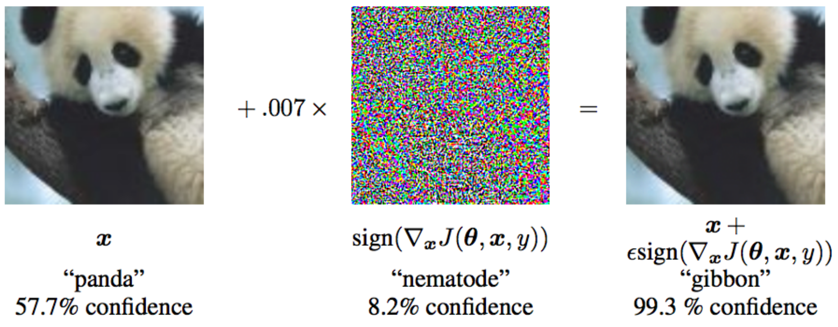
$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 2: Adversarial Example Generated Using Fast Sign Gradient Method

method (shown in (2)), and a iterative least likely method (shown in (3)).

$$\begin{aligned} x_0^{adv} &= x \\ x_{i+1}^{adv} &= clip(x_i^{adv} - \alpha * sign(\nabla(x_i^{adv}, y_{true}))) \end{aligned} \tag{2}$$

$$\begin{aligned} y_{ll} &= argmin\{P(y|x)\} \\ x_0^{adv} &= x \\ x_{i+1}^{adv} &= clip(x_i^{adv} - \alpha * sign(\nabla(x_i^{adv}, y_{ll}))) \end{aligned} \tag{3}$$

4 Methodology & Experiments

4.1 ConvNet on MNIST Dataset

The ConvNet architecture used here is the classic LeNet5 [7]. Breaking it down, LeNet5 contains:

- Two convolutional layers using ReLU as activation function.
- Two max pooling layers each followed after a convolutional layer with window size 2 * 2.
- Two fully connected layers after the last max pooling layer, the second one map the features to 10 dimensions which represent the output for each class

Input tensor will then flow to a softmax layer which output the probability of it being labeled as each class.
Dropout setting is also enabled to avoid over-fitting. After 5,000 iterations, we

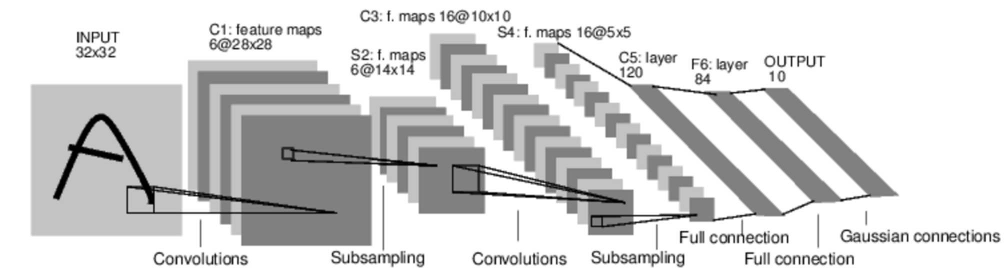

Figure 3: Architecture of LeNet5

have trained a model with a performance as below:

data set	accuracy
training set	100%
test set	98.8%

Table 1: Model Performance

4.2 Generate Adversarial Examples

For a list of examples from the MNIST dataset which were correctly labeled as ”2”, we would like to generate adversarial images that can fool ConvNet trained with high accuracy to be labeled as ”6”. Different from [4], instead of generating adversarial images which can be classified as any class, we have a target class we would like to fool ConvNet to label it to.

4.2.1 Iterative Fast Sign Gradient Method

Inspired by [6] mentioned in previous section, since now we have a target class to work with, we modified the equation a little bit, so the image will move small steps one by one to more towards the target class, until the first time it is misclassified by the ConvNet to the target class.

$$\begin{aligned} x_0^{adv} &= x \\ x_{i+1}^{adv} &= clip(x_i^{adv} - \alpha * sign(\nabla(x_i^{adv}, y_{target}))) \{0, 1\} \end{aligned} \tag{4}$$

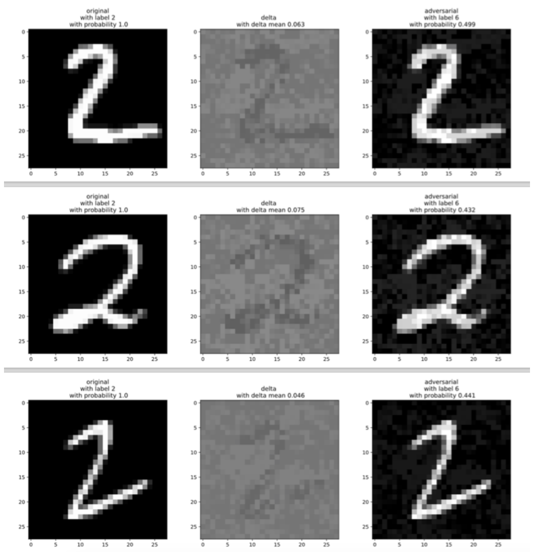

Figure 4: Comparison between adversarial examples generated using fast sign gradient method with original images

4.2.2 Direct Gradient Method

Differently from generic adversarial example generation with no specified target class, since we already know which target class we want move our image towards to, instead of using sign to generate "random" unit noise, we would also like to see how it likes to use gradient directly.

$$\begin{aligned} x_0^{adv} &= x \\ x_{i+1}^{adv} &= clip(x_i^{adv} - \alpha * \nabla(x_i^{adv}, y_{target})) * (x_i^{adv} + 0.1) \{0, 1\} \end{aligned} \tag{5}$$

The preliminary experiments we saw in 2and 5 are okay-ish. However, one obvious problem pops up: the adversarial images can be identified by human eyes easily because of the noises in the black background of original images. Of course, the reason that the change is obvious is related to simplicity of gray scale images with only 28 * 28 pixels. It is also because above "gradients" propagated are absolute changes, it only takes a look at sign of gradient or value of gradient

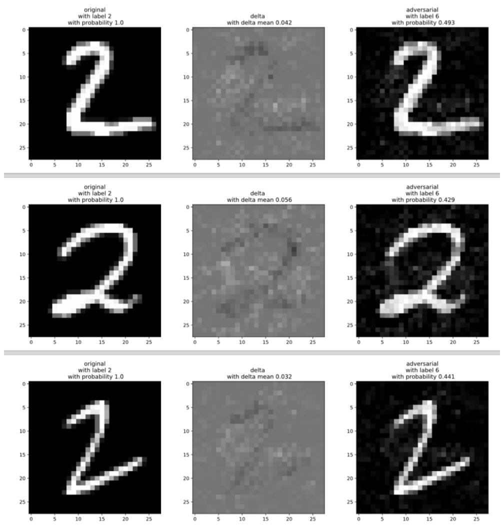

Figure 5: Comparison between adversarial examples generated using direct gradient method with original images

without considering what's the relative changes compared with original images. We believe human eyes works in a different way: when we compare two things, we turned to focus more from a relative perspective. The whiter the grid is, the more changes we can "ignore"; if the grid is purely black before and suddenly it has some color, it will be picked up by human eyes immediately.

4.2.3 Input Scaled Direct Gradient Method

Based on the observation made from results got using iterative fast sign gradient method and direct gradient method, we want to try "distort" the equation to propagate the "gradient" with consideration of original image.

$$\begin{aligned} x_0^{adv} &= x \\ x_{i+1}^{adv} = clip(x_i^{adv} - \alpha * \nabla(x_i^{adv}, y_{target}) * (x_i^{adv} &+ 0.1))\{0,1\} \end{aligned} \tag{6}$$

Changes made including:

- scale sign with the original x
- smooth the changes by adding 0.1 to all x, so the pure dark grid is not completely unchangeable.

Examples on how input scaled fast sign gradient method works on MNIST dataset can be see at 6.

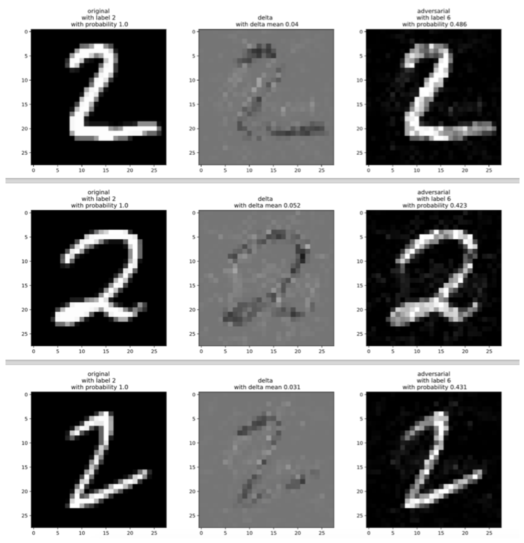

Figure 6: Comparison between adversarial examples generated using input scaled direct gradient method with original images

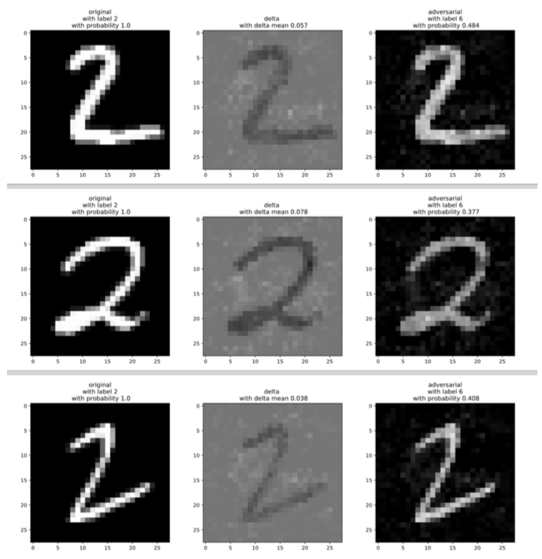

Figure 7: Comparison between adversarial examples generated using input regularized direct gradient method with original images

4.2.4 Input Regularized Gradient Method

Another approach worth a try is modifying the loss function to meet the new requirement of this task.

Another assumption here is that, if the whole image shifted from brighter to darker or from darker to brighter, it is harder to notice the trace of artificial perturbation than unevenly noises hidden in the background.
Inspired by the regularization used for controlling the change in weights, and follow the same idea, we would also prefer modifying the brighter grids rather than completely black ones.

$$\begin{gathered} \lambda = 1 \\ loss_{reg} = loss + \frac{\lambda}{2} * ||x||^2 \end{gathered} \tag{7}$$

Two most commonly used normalization methods are L1-Norm and L2-Norm. Based on the nature of linear function (L1) and quadratic function (L2), in our case, L2 will be a better fit.
Inside of doing L2-norm on weight, we create a new loss function with L2-norm on input itself.

4.2.5 Performance Comparison

Above, we have tried four different ways of generating adversarial examples, including: fast sign gradient, direct gradient, input scaled direct gradient and regularized gradient methods. To have a sense of which method is better, we use a simple measurement in 8. We average the score for the 10 example images chosen from MNIST dataset.

$$delta_{score} = \sum_i \sum_j ||delta_{ij}||^2/(28 * 28) \tag{8}$$

There is no doubt that this measurement doesn't capture all the targets we

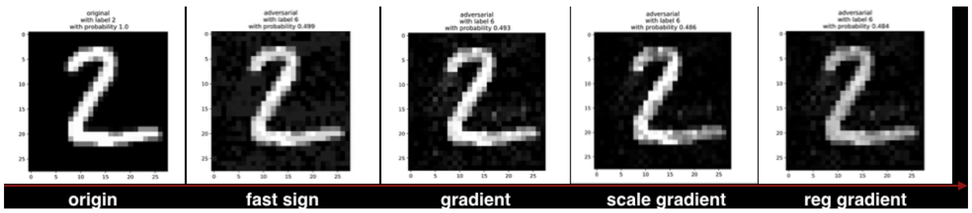

Figure 8: Comparison among all Methods Used for Generating Adversarial Examples in the Report

want to optimize or achieve in that task, and it is also true 10 examples can't represents the whole dataset. We just want to use this to roughly quantify all the experiments we have done so far so readers might have a better sense than just looking at the images shown above, no decisions should be made before we have an advanced evaluation system designed.

method	delta_{score}
fast sign	0.065
direct gradient	0.047
scale direct gradient	0.045
reg scale gradient	0.062

Table 2: Adversarial Example Quality Comparison

5 Conclusion

The whole experience is impressive. Start with limit knowledge on deep learning especially ConvNets and no background in computer vision, using the spare time of several days, the author was able to shallowly touching lots of fascinating fields, including the definition and design of ConvNets, the dangers of adversarial examples, the reason behind fragile of linear classifiers, and the ways to general adversarial examples.
The main study goal here is to research on how to generate "high quality" adversarial examples. We have tried some modified versions inspired from general regularization method and the classic iterative fast sigh gradient method. Among the methods we tried, the input scaled direct gradient delivered the best performance on the 10 example images we chosen from MNIST test set in terms of delta score, it is also the best in dealing with the background noise. However, it is at the same time pretty obvious that the color of the bright grids didn't not get evenly changed. The regularized direct gradient method may not be the best in terms of delta score, but it is pretty evenly shifted from the original picture.

6 Future Works

Since the time is limited, we were not able to go through all the existing works for generating adversarial examples for object detection types of problems nor experiments on all the ideas. In the future, in terms of generating high quality adversarial examples, what can be done including:

- comprehensive ways of evaluate the quality of adversarial examples
- more experiments on advanced methods for generating adversarial example.

For example, in [1], it mentioned about using total variation to improve the quality of adversarial examples. We would love to see how it will optimize the adversarial examples in MNIST dataset. In [9], it provides a pretty comprehensive study on the regularization tricks we can use. The author is planning to try it out later.
Finally, after knowing the dangerous of adversarial examples and how "easy" it

can be massively generated, what can we do with it?
Lots of papers mentioned about adversarial training and Generative Neural Networks[3] which has been really popular in the deep learning field recently. Due to the limit of our knowledge and time, we didn't dive into the details in the report, but it definitely worth a look in the future.

References

[1] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: Elastic-net attacks to deep neural networks via adversarial examples. *arXiv preprint arXiv:1709.04114*, 2017.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. 2016.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

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

[5] Andrej Karpathy. Breaking linear classifiers on imagenet.

[6] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[7] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[8] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017.

[9] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.