Adversarial Examples & Neural Networks

Aalap Parimalkumar Rana
Maryland Applied Graduate Engineering
University Of Maryland
College Park, MD 20740
aalap321@umd.edu

Deepti Bisht
Department Of Computer Science
University Of Maryland
College Park, MD 20740
dbisht@umd.edu

Aisha Siddiqui
Department Of Computer Science
University Of Maryland
College Park, MD 20740
aisha10@umd.edu

Abstract

With the application of deep learning in the field of computer vision applications such as autonomous cars and robotic systems, higher accuracy on tasks like image classification, recognition, and prediction have become imperative. Convolutional neural network dominates the classification task in computer vision. However, convolutional neural networks often fail to classify "Adversarial Examples". The main focus of the project is understanding adversarial examples, methods for generating adversarial examples, and possible defense techniques through the study of different research papers.

1 Introduction

Deep learning models often achieve high accuracy on image processing involving benign input data while erring on adversarial examples. Adversarial examples modify the original input images with small enough perturbations to maintain perceptual similarity. This leads many machine learning models to fail to correctly classify these perturbed images.This observation indicates that the trained models are not adequately learning the true underlying concepts that determine the correct output label.
First, we will define adversarial examples and their various categories. The following two sections describe two different methods for generating adversarial examples: the Fast Gradient Sign method and the DeepFool method. Finally, we will discuss adversarial training, a method of defense against adversarial examples.

2 Adversarial Examples

Szegedy et al. first discovered that deep learning models have a high probability of misclassifying images introduced with a small degree of perturbation. These misclassified samples are adversarial examples. The perturbations are found by optimizing the input examples to maximize the prediction error.
Categories: Adversarial examples can be broadly classified as black box and white box attacks. In white box attacks the adversary has total knowledge of the model, including the gradient, while in black box attack it does not. In black box attacks, the adversary exploits a model by providing a series of carefully crafted inputs and observing outputs. Categories are further defined in the appendix.

Properties: Some of the properties of the adversarial examples are -

- Adversarial examples are transferable from one model to another.
- Rather than being scattered randomly in small pockets, adversarial examples occur in large, contiguous regions.
- They can also be used to train deep learning models and improve a model's generalizability.

3 Fast Gradient Sign Method

3.1 Linear & Non-Linear Explanation of Adversarial Example

Digital images are represented as a collection of pixels. To limit each pixel to 8 bits, any pixel information below a 1/255 precision value is tossed aside. If we consider a pure image "x" and a small perturbation "η" whose value is less than this precision and combine it with all pixel to form a new image "$\bar{x}$" such that $\bar{x}$ = x+η.
Claim: Classifier will predict the same class for the image x and $\bar{x}$.
Let us consider the weight vector to be ω. Thus, the product of the perturbed image "$\bar{x}$" and weight ω can be expressed as

$$\omega^\top * \bar{x} = \omega^\top * x + \omega^\top * \eta \tag{1}$$

The activation will grow by $\omega^\top * \eta$ due to the adversarial perturbation .We can maximize this increase with reference to the max norm constraint on η and assigning $\eta = sign(\omega)$. Such small changes in activation can grow linearly with the increase in dimensions. This increase will be m*n where n is the dimension of vector w with average magnitude of elements m.

For high dimensional problems, we can make many infinitesimal changes to the input that add up to one large change to the output, because the $||\eta||_\infty$ does not grow with the dimensionality of the problem. Rather, the change in the activation caused by perturbation by η grows linearly with n making it significant for the classifier to detect. Thus, this contradicts our claim and proves that linear models are vulnerable to adversarial examples.

Considering the non-linear model; most of the activation functions namely RelU, sigmoid and maxout network, and others are analyzed in the intervals where their behavior is linear in order to make the optimization process easy. But this also makes the non-linear model vulnerable to linear perturbations.

3.2 Fast Gradient Sign Method Algorithm

Fast gradient sign method exploits this linearity. If we consider θ to represent the model parameter, x be the input to the model,y to be the target label and the cost function to be optimized is $J(\theta, x, y)$. Following steps are followed to generate an adversarial example $\bar{x}$[6]

1. Linearize the cost function about x

$$J(\theta, x, y) = J(\theta, x_0, y) + \nabla_x J(\theta, x, y)^T * r \tag{2}$$

2. Maximize equation (2) wrt r

$$maximize\{J(\theta, x_0, y) + \nabla_x J(\theta, x_0, y)^T * r\}; ||r||_\infty <= \eta \tag{3}$$

3. Above equation is equivalent to

$$minimize\{-J(\theta, x_0, y) + -\nabla_x J(\theta, x_0, y)^T * r\}; ||r||_\infty <= \eta \tag{4}$$

The solution for the above minimization is

$$r = \eta * sign(\nabla_x J(\theta, x, y)) \tag{5}$$

Thus the adversarial example can be given as:

$$\bar{x} = x + \eta * sign(\nabla_x J(\theta, x, y)) \tag{6}$$

Below mentioned are our experimental results:

Table 1: Accuracy With FGSM advesarial examples

Type of Test Data	Accuracy(%)
Pure Test Data	97%
Adversarial Test Data	29%

4 Deep Fool

In this section we will elaborate on "Deep fool". This is an optimal method to generate a minimally perturbed image when compared to fast gradient sign method, as it produces a minimal perturbation vector. Deep fool can adversarially affect both complex multi-class and simple binary classifiers. We will discuss the binary case (refer to the appendix for the extension to multi-class classifiers).
For the purpose of our analysis we will consider the following assumptions:

- Let f be an arbitrary scalar-valued affine image classification function given by $f = \omega^\top * x + b$ which maps $f: R^n \rightarrow R$.
- Let the minimum perturbation required to wrongly classify be r_* and the measure of robustness is equal to the orthogonal distance from point. $\Delta(\boldsymbol{x}; \hat{k}) := \min_{\boldsymbol{r}} \|\boldsymbol{r}\|_2$ subject to $sign(\boldsymbol{x} + \boldsymbol{r}) \neq sign(\boldsymbol{x})$

The minimum perturbation r_* for x_0 can be computed by the below mentioned equation.

$$\boldsymbol{r}_*(\boldsymbol{x}_0) := \arg\min \|\boldsymbol{r}\|_2 \text{sign}\left(f(\boldsymbol{x}_0 + \boldsymbol{r})\right) \neq \text{ sign}\left(f(\boldsymbol{x}_0)\right) = -\frac{f(\boldsymbol{x}_0)}{\|\boldsymbol{w}\|_2^2}\boldsymbol{w} \tag{7}$$

We observe that the input to the deep fool algorithm is an input image x and the affine

Algorithm 1 DeepFool for binary classifiers

input: Image $\boldsymbol{x}$, classifier f.
output: Perturbation $\hat{\boldsymbol{r}}$.
Initialize $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$, $i \leftarrow 0$.
while $\text{sign}(f(\boldsymbol{x}_i)) = \text{sign}(f(\boldsymbol{x}_0))$ **do**
    $\boldsymbol{r}_i \leftarrow -\frac{f(\boldsymbol{x}_i)}{\|\nabla f(x_i)\|_2^2} \nabla f(\boldsymbol{x}_i)$,
    $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{r}_i$,
    $i \leftarrow i + 1$.
end while
return $\hat{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i$.

Figure 1: Deep Fool binary classifier[2]

classifier f.Furthermore, in each iteration "i" we linearize f around the current point x_i and compute the minimal perturbation of the linearized classifier by:

$$\arg\min_{\boldsymbol{r}_i} \|\boldsymbol{r}_i\|_2 \text{ subject to } f(\boldsymbol{x}_i) + \nabla f(\boldsymbol{x}_i)^T \boldsymbol{r}_i = 0. \tag{8}$$

The perturbation r_i is added to x_i to form a new perturbed image x_{i+1}.The algorithm terminates when x_{i+1} changes sign of the classifier and the final output is a perturbed image which is wrongly classified by the classifier given by x_{i+1}.

5 Defense to Adversarial Examples: Adversarial Training

Adversarially training deep learning models involves optimizing model parameters against a strong adversary. The stronger the adversary that the model is trained against, the more robust it will be against a wide range of attacks. To measurably ensure a classifier's robustness, we precisely define the

types of attacks it is resistant to. We model these attacks by defining a constraint on the perturbations that produce adversarial examples. The set of adversarial examples that are produced by modifying each input example are that input's evaluation set.

5.1 Saddle-Point Problem

We optimize the adversary by choosing the examples in each evaluation set that maximize loss. This is called the inner maximization problem. Then the model parameters are optimized against these inputs to minimize loss. This is called the outer maximization problem. The composition of these optimizations is a saddle-point optimization (eq. 9) that serves to optimize the robustness of a deep learning model.

Attack Model: evaluation sets $\mathcal{S}$ for each input x

$$\min_{\theta} \rho(\theta), \text{ where } \rho(\theta) = \mathbf{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y) \right] \tag{9}$$

5.2 Inner Maximization: Projected Gradient Descent

Projected gradient descent (PGD) is used to solve the inner maximization problem because it is an effective constrained optimization algorithm. It steps along the direction of the gradient of the loss function, then projects the optimized input back into the evaluation set. We can see that PGD is an effective adversary because networks trained with PGD have lower loss than those with standard training.
Although the inner maximization problem is non-concave, we see that it is tractable using PGD. Experimentally, we see that the maxima found by multiple restarts of PGD from inputs within the evaluation sets converge to similar cross-entropy values for both standard and adversarially trained networks. Since the maxima values are so similar, it is likely that it does not matter which of the maxima are chosen by PGD, thus making the inner maximization tractable. Furthermore, since any first-order adversary will also be optimizing by moving in the direction of the gradient, it is unlikely that they will find a better local maxima than PGD. The concentration of loss values of PGD's maxima after a large number of restarts implies that better maxima would at least be difficult to find. For these reasons, we conclude that networks trained with PGD will be almost universally robust to first-order adversaries.

5.3 Outer Maximization: Stochastic Gradient Descent

To solve the outer minimization problem, we replace the input data with their adversarial perturbations, and perform Stochastic Gradient Descent. Danskin's theorem implies that this is equivalent to moving in the right direction of the gradient of the saddle-point problem. The intuition involved is that the gradient of the loss function with the optimized perturbed inputs and the gradient of the full saddle-point problem are locally the same.

5.4 Network Capacity and Learning Rate

Since the deep learning model is being trained on perturbed data, the classifier is being optimized to learn a more complex decision boundary than if it were training on the unperturbed examples. For this reason, the network architecture needs to have a sufficient capacity to be successfully trained against a strong adversary. Networks with small capacities are unable to optimize against adversarial examples while maintaining accuracy on unperturbed examples. Increasing the capacity of the model during training leads to smaller loss from the saddle-point problem. Smaller learning rates also prevent overfitting to adversarial examples.

Acknowledgments

We would like to thank Dr Soheil Feizi for his constant support and guidance. We are grateful to have had his encouragement. We would also like to thank to the TA's: Nitin Balachandran ,Neha Kalibhat and Amir Nili.

References

[1] Goodfellow, I.J., Shlens, J., Szegedy, C. (2014). Explaining and Harnessing Adversarial Examples. CoRR, abs/1412.6572.

[2] Moosavi-Dezfooli, S., Fawzi, A., Frossard, P. (2016). DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2574-2582.

[3] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A. (2017). Towards Deep Learning Models Resistant to Adversarial Attacks. ArXiv, abs/1706.06083.

[4] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R. (2014). Intriguing properties of neural networks. CoRR, abs/1312.6199.

[5] https://harvard-iacs.github.io/2019-CS109B/

[6] Workshop2019_04A.pdf

Appendix

Adversarial Threat Model

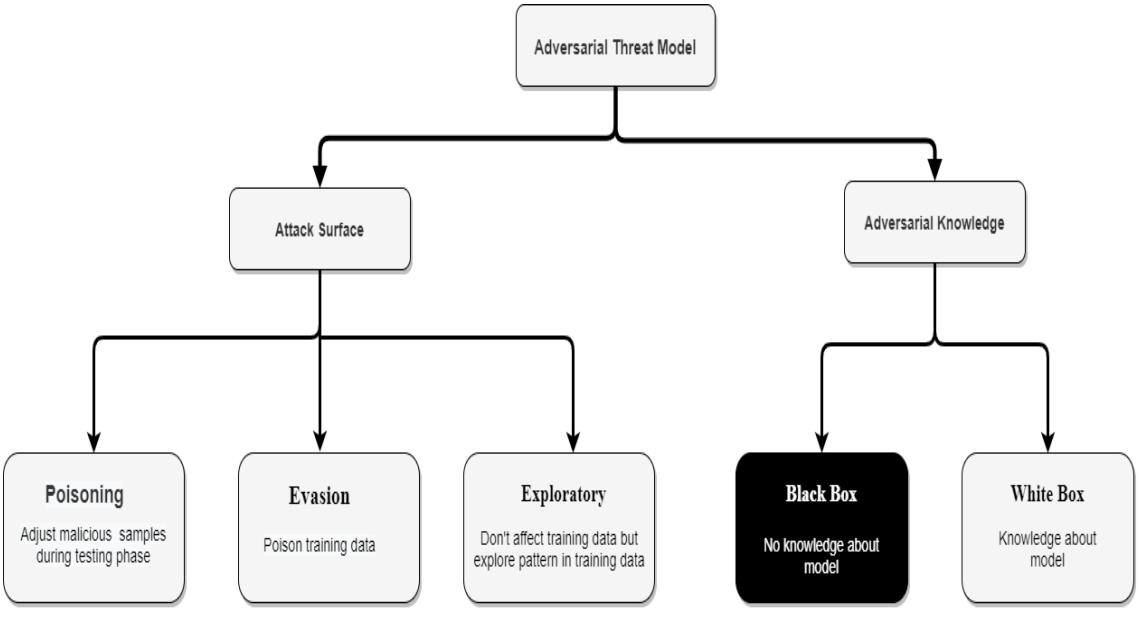


Figure 2: Threat Model of adversarial examples

First Order adversaries - These are the attacks that modify the input by moving along the gradient.They only utilize the local first order information about the network. Projected gradient descent (PGD) is a universal "first-order adversary".

Deep Fool- Multiclass classifiers

Multi-class classification can be considered a combination of many binary classifiers. Thus the main algorithm remains the same. However, we now have to select a hyper-plane which is the closest to the data point. We consider "n" hyperplanes that are closest to the data point, omitting the hyperplane for the data point's true class. We calculate the closest hyperplane by finding the minimum gradient difference between the original gradient(true label) and selected gradient (for n selected classes) and the difference between their respective labels.To obtain the minimal perturbation we project the point on the found nearest hyper plane.

Algorithm 2 DeepFool: multi-class case

input: Image $\boldsymbol{x}$, classifier f.
output: Perturbation $\hat{\boldsymbol{r}}$.

Initialize $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$, $i \leftarrow 0$.
while $\hat{k}(\boldsymbol{x}_i) = \hat{k}(\boldsymbol{x}_0)$ **do**
  for $k \neq \hat{k}(\boldsymbol{x}_0)$ **do**
    $\boldsymbol{w}'_k \leftarrow \nabla f_k(\boldsymbol{x}_i) - \nabla f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
    $f'_k \leftarrow f_k(\boldsymbol{x}_i) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
  end for
  $\hat{l} \leftarrow \arg\min_{k \neq \hat{k}(\boldsymbol{x}_0)} \frac{|f'_k|}{\|\boldsymbol{w}'_k\|_2}$
  $\boldsymbol{r}_i \leftarrow \frac{|f'_{\hat{l}}|}{\|\boldsymbol{w}'_{\hat{l}}\|_2^2} \boldsymbol{w}'_{\hat{l}}$
  $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{r}_i$
  $i \leftarrow i + 1$
end while
return $\hat{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i$

Figure 3: Deep fool multi-class classifiers[2]

Comparison- Fast Gradient Sign Method & Deep Fool

These are few experimental results which gives a comparative analysis between Deep fool and Fast gradient sign method. The experiments were carried out on MNIST , CIFAR-10, and ILSVRC2012 dataset.Model for ILSVRC2012 used pre-trained neural network namely CaffeNet and GoogLeNet, and in case of MNIST data set a two-layer fully connected network, and a two-layer LeNet was used.Moreover, for CIFAR-10 three-layer LeNet architecture, as well as a Network In Network (NIN) architecture was implemented. Fast gradient sign method requires ϵ, thus for consistency a small value was selected which gave a 90% misclassification on data. We observed the following:

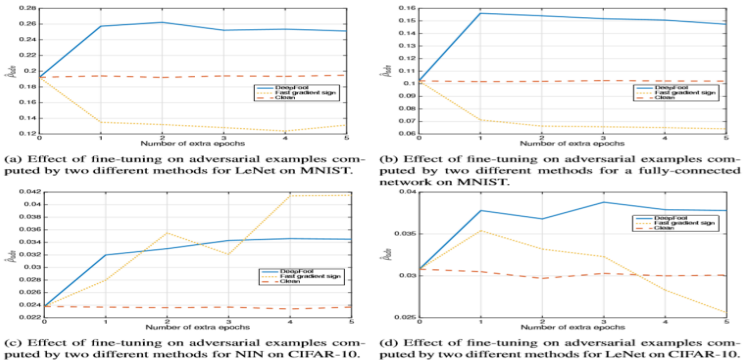
1. The time taken to generate a single adversarial example is more for Deep Fool than by fast gradient sign method.
2. The perturbation generated by deep fool were about 4 times smaller than that produced by fast gradient sign method.

Classifier	Test error	$\hat{\rho}_{\text{adv}}$ [DeepFool]	time	$\hat{\rho}_{\text{adv}}$ [4]	time	$\hat{\rho}_{\text{adv}}$ [18]	time
LeNet (MNIST)	1%	2.0×10^{-1}	110 ms	1.0	20 ms	2.5×10^{-1}	> 4 s
FC500-150-10 (MNIST)	1.7%	1.1×10^{-1}	50 ms	3.9×10^{-1}	10 ms	1.2×10^{-1}	> 2 s
NIN (CIFAR-10)	11.5%	2.3×10^{-2}	1100 ms	1.2×10^{-1}	180 ms	2.4×10^{-2}	>50 s
LeNet (CIFAR-10)	22.6%	3.0×10^{-2}	220 ms	1.3×10^{-1}	50 ms	3.9×10^{-2}	>7 s
CaffeNet (ILSVRC2012)	42.6%	2.7×10^{-3}	510 ms*	3.5×10^{-2}	50 ms*	-	-
GoogLeNet (ILSVRC2012)	31.3%	1.9×10^{-3}	800 ms*	4.7×10^{-2}	80 ms*	-	-

Table 1: The adversarial robustness of different classifiers on different datasets. The time required to compute one sample for each method is given in the time columns. The times are computed on a Mid-2015 MacBook Pro without CUDA support. The asterisk marks determines the values computed using a GTX 750 Ti GPU.

Figure 4: Comparison table[2]

It could be concluded from the above table that deep fool provides minimal perturbation and thus can be used as a baseline for measuring robustness for different models. Thus, different networks were fine tuned using adversarial examples generated using Deep fool and Fast gradient sign method by performing 5 additional epochs with a 50% decreased learning rate only on the perturbed training set.



(a) Effect of fine-tuning on adversarial examples computed by two different methods for LeNet on MNIST.

(b) Effect of fine-tuning on adversarial examples computed by two different methods for a fully-connected network on MNIST.

(c) Effect of fine-tuning on adversarial examples computed by two different methods for NIN on CIFAR-10.

(d) Effect of fine-tuning on adversarial examples computed by two different methods for LeNet on CIFAR-10.

Figure 5: Fine tuning comparison[2]

1. It was observed that the robustness for networks trained with deep fool examples was more than those with fast gradient sign method.
2. Fine tuning network with highly perturbed image may result in decreasing the robustness because it acts as a regularizer that does not represent the correct distribution of the data.

Our Experiment

We followed the lab-manual "Harvard-CS109B-Lab21" to generate adversarial example using Fast gradient sign method and tested the accuracy of the trained model on clean data and adversarial data generated for MNIST dataset.We carried out the same experiment for Deep fool too.

Tools

We used the following tools and library: Jupiter notebook, Cleverhans Lib, Keras, Tensorflow, numpy and matplotlib

Model

We formed a simple fully connected network with two layers. The first layer takes in the 28 x 28 input image, and has a hidden layer of size 512.It passes this to next fully connected layer which generates the output.Network specifications are displayed in the figure below:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 6: Model

Results

We observed that accuracy of the model dropped when tested with adversarial examples, the accuracy with clean data was noted around 97% while the accuracy with adversarial example was just 29%.In case of deep fool we obtained an accuracy of 1.04%.

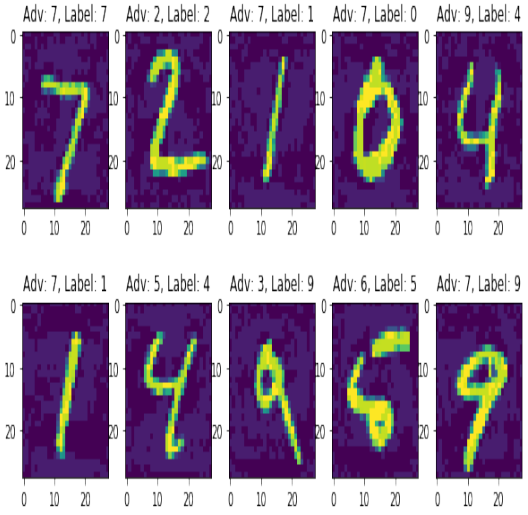


Figure 7: Adversarial example-FGSM

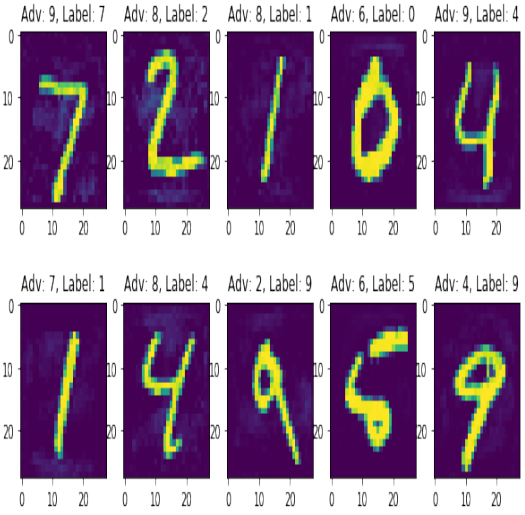


Figure 8: Adversarial example-Deep fool