Data Privacy: CS573 Final Project
Evaluation of Adversarial Attacks on Neural Networks

Zexi Yuan
Emory University
zexi.yuan@emory.edu

Yuchun Liu
Emory University
yuchun.liu@emory.edu

December 11, 2018

Abstract

Neural networks are vulnerable to adversarial examples, which are inputs crafted by human-imperceptive perturbations such that the output of the perturbed inputs is misclassified with high confidence. Generating adversarial examples, called adversarial attacks, are beneficial to identify the vulnerability of neural networks and improve their robustness. In order to understand the mechanism of adversarial attacks, classic adversarial attacks are implemented and compared to investigate their strength and weakness. The experimental results show that the iterative fast gradient sign method outperforms the one-step fast gradient sign method.

I. Introduction

Neural networks have been widely used in various fields such as Image Recognition and Text translation because of their effectiveness. For example, face recognition algorithms today can achieve a 99.5% accuracy, which is close to the accuracy of humans. However, neural networks are often vulnerable to adversarial examples [1], which are crafted by small perturbations such that they are only slightly different from correctly classified examples. Although these tiny differences are imperceptible to human, they are easily misclassified by fine-trained neural networks. Moreover, the same adversarial example can even be misclasssified by a wide variety of models with different architectures trained on different subsets of the training data. Given this vulnerability, the attacker can easily perform adversarial attack by generating adversarial examples to mislead the targeted model to output the attacker-desired predictions. In order to defend adversarial attacks and strengthen the robustness of neural networks, researchers recently pay an increasing attention to understand adversarial examples because it helps to identify the vulnerability of the models.

The reason why adversarial examples work is still unknown, and speculative explanations are given by researchers. In the past, it is considered to be the extreme non-linearity of deep neural networks, or perhaps combined with insufficient regularization of the purely supervised learning problem. Recently, Goodfellow et al. [2] however suggested that the primary cause of neural networks' vulnerability to adversarial perturbation is their linear nature.

To defend adversarial attack, traditional techniques for improving the robustness of neural network against adversarial perturbations, such as weight decay and dropout, generally do not provide a solid support. Instead, an effective defense can be provided by adversarial training and defensive distillation. These two methods, especially the adversarial training, require a deep understanding of existed adversarial attacks because the best way to defend is to understand the way to offend. Thus, the main purpose of this paper is to implement and compare classic adversarial attacks to deeply understand adversarial attacks.

Based on the knowledge of the structure and parameters of a given model, adversarial attacks can be divided into white box attacks and black/gray box attacks. If the attacker has known the knowledge of the structure and parameters of a given model, the attacker can perform white box attacks, including optimization-based methods such as L-BFGS [1] and fast/approximate methods such as fast gradient sign method (FGSM) [2] and iterative fast gradient sign method (I-FGSM) [3]; otherwise, the attacker can only perform black/gray box attacks. This paper mainly focuses on the white box attacks and its fast/approximate methods to investigate their performance on attacking neural networks.

II. Backgrounds

In this section, the background knowledge are provided and the related works are reviewed for the paper. Given a classifier $f(\boldsymbol{x}) : \boldsymbol{x} \in \boldsymbol{x} \to y \in Y$ that outputs a label y as the prediction for an input x, adversarial attacks aim to seek an adversarial example $\boldsymbol{x}^*$ that slightly differs from $\boldsymbol{x}$ but is misclassified by the classifier. Based on the adversarial target, the adversarial task can be divided into two classes: non-targeted and targeted tasks. For a correctly classified input $\boldsymbol{x}$ with ground-truth label y_{true} such that $f(\boldsymbol{x}) = y_{true}$, the non-targeted task misleads the classifier as $f(\boldsymbol{x}^*) \neq y_{true}$ by adding small noise to $\boldsymbol{x}$ without changing the label; and the targeted task fool the classifier by outputting a specific label as $f(\boldsymbol{x}^*) = y_{target}$, where y_{target} is the target label specified by the adversary, and $y_{target} \neq y_{true}$. In most cases, the difference between $\boldsymbol{x}^*$ and $\boldsymbol{x}$ should be subject to the constraint that the L_p norm of the adversarial noise is required to be less than an allowed value ϵ as $\|\boldsymbol{x}^* - \boldsymbol{x}\|_p \leq \epsilon$, where p could be 0, 1, 2, ∞. In this paper, the discussed adversarial attacks are based on the fast/approximate method under the white box scenarios, such as FGSM and I-FGSM.

i. Fast Gradient Sign Method

Fast gradient sign method (FGSM) [2] is a one-step gradient-based attack that generates an adversarial example $\boldsymbol{x}^*$ within the L_∞ norm bound to be misclassified by the classifier.

For the non-targeted attack, the adversarial example $\boldsymbol{x}^*$ can be generated by the following equation.

$$\boldsymbol{x}^* = \boldsymbol{x} + \epsilon \cdot sign(\nabla_x J(\boldsymbol{x}, y_{true})) \tag{1}$$

where $\nabla_x J(\boldsymbol{x}, y_{true})$ is the gradient of the loss function with respect to $\boldsymbol{x}$, and the $\boldsymbol{x}^*$ is subject to the L_∞ norm constraint that $\|\boldsymbol{x}^* - \boldsymbol{x}\|_\infty \leq \epsilon$. The above equation aims to find an adversarial example $\boldsymbol{x}^*$ by maximizing the loss function $J(\boldsymbol{x}^*, y_{true})$, where J is often the cross-entropy loss.

For a targeted FGSM attack, the adversarial example $\boldsymbol{x}^*$ can be generated by the following equation.

$$\boldsymbol{x}^* = \boldsymbol{x} - \epsilon \cdot sign(\nabla_x J(\boldsymbol{x}, y_{target})) \tag{2}$$

According to the above two equations, the difference between non-targeted and targeted attacks on generating adversarial examples is the direction of adding crafted perturbations. The non-targeted attacks aim to make the output of the input far away from the ground truth y_{true}, while the targeted attacks make the output of the input close to the target y_{target}.

ii. Iterative Fast Gradient Sign Method

Instead of adding the perturbations in a single step, the iterative fast gradient sign method (I-FGSM) [3] iteratively performs FGSM multiple times with a small step size α.

For the non-targeted attacks, the generation of adversarial examples in I-FGSM can be defined as the following equation.

$$\begin{aligned} &\boldsymbol{x}_0^* = \boldsymbol{x}, \\ &\boldsymbol{x}_{t+1}^* = Clip_{x,\epsilon}\{\boldsymbol{x}_t^* + \alpha \cdot sign(\nabla_x J(\boldsymbol{x}_t^*, y_{true}))\} \end{aligned} \tag{3}$$

In order to satisfy the L_∞ norm constraint $\|\boldsymbol{x}_t^* - \boldsymbol{x}\|_\infty \leq \epsilon$, the intermediate results $\boldsymbol{x}_t^*$ are clipped after each step into the ϵ vicinity of $\boldsymbol{x}$, which means that the α can be set as $\alpha = \frac{\epsilon}{T}$ with T being the nu mber of iterations and $t \in \{1, 2, \cdots, T\}$.

For the targeted attacks, which is similar to FGSM, the generation of adversarial examples in I-FGSM can be expressed as

$$\begin{aligned} x_0^* &= x, \\ x_{t+1}^* &= Clip_{x,\epsilon}\{x_t^* - \alpha \cdot sign(\nabla_x J(x_t^*, y_{target}))\} \end{aligned} \tag{4}$$

It has been shown that the I-FGSM outperforms the FGSM at the cost of worse transferability [3, 4].

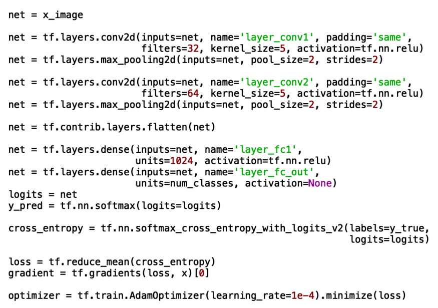
III. Experiment Setup

In our project, different adversarial attacks are implemented with Python3 and the Tensorflow framework. Tensorflow is an open source software library that widely used for machine learning applications such as neural networks. It is beneficial for the design of neural networks and efficiently performing the related computations. Besides, optimization on GPU computations within Tensorflow provide a solid support for the time-consuming computations on computer vision and neural networks. In order to obtain better computational resources, Google Cloud Platform will be used to deploy the GPU. Due to the difficulty to setup the environment in the Google Cloud Platform, Docker is used to prepare the required environment containers. Meanwhile, various adversarial attack tutorials based on Tensorflow made a great contributions to our project such that classic adversarial attacks are successfully implemented.

As the database widely used for training and testing various image processing systems in the field of machine learning,the MNIST dataset is used as our experimental dataset. The MNIST database contains 60,000 training images and 10,000 testing images, where the size of each image is 28 * 28 [5]. The dataset has only 10 classes and the ground truth of each image is clear. As a result, it is easy and intuitive for us to training and evaluating the models.

IV. Experiments and Results

Several experiments have been done to compare these FGSM-based methods. The experiments are mainly inspired by two papers published in 2016[6][7]. The project implements the "label leaking" property mentioned in the first paper and compare the accuracy of different adversarial attacks following the steps illustrated in the next paper.



```
net = x_image

net = tf.layers.conv2d(inputs=net, name='layer_conv1', padding='same',
                       filters=32, kernel_size=5, activation=tf.nn.relu)
net = tf.layers.max_pooling2d(inputs=net, pool_size=2, strides=2)

net = tf.layers.conv2d(inputs=net, name='layer_conv2', padding='same',
                       filters=64, kernel_size=5, activation=tf.nn.relu)
net = tf.layers.max_pooling2d(inputs=net, pool_size=2, strides=2)

net = tf.contrib.layers.flatten(net)

net = tf.layers.dense(inputs=net, name='layer_fc1',
                      units=1024, activation=tf.nn.relu)
net = tf.layers.dense(inputs=net, name='layer_fc_out',
                      units=num_classes, activation=None)
logits = net
y_pred = tf.nn.softmax(logits=logits)

cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_true,
                                                           logits=logits)

loss = tf.reduce_mean(cross_entropy)
gradient = tf.gradients(loss, x)[0]

optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(loss)
```

Figure 1: *The structure of classification model*

i. Classification model

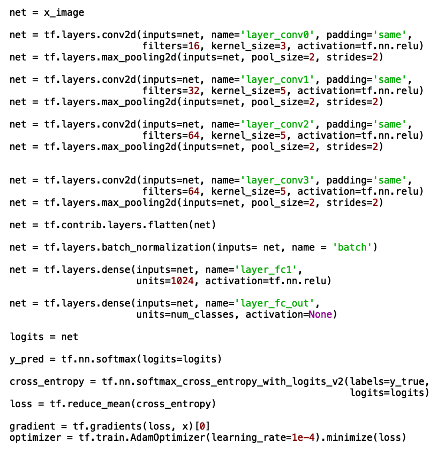
To train the basic classification model, we used the the structure in the tutorial[8]. The input data is 28*28 images of pixel values in [0-1] and the output is an one-hot matrix of the prediction result. The neural network's structure is relatively simple, and we choose it because its prediction accuracy at the test dataset reaches 98.3%. The performance of this model is enough for us to conduct further experiments. The code of defining the classification model is shown in Figure 1.

ii. Label Leaking

Label leaking is an interesting property. When a model is trained on FGSM adversarial examples and then evaluated using FGSM adversarial examples, the accuracy on adversarial images becomes much higher than the accuracy on clean images.

The paper claimed that the effect only exists when we generate adversarial examples with one-step methods that based on the true class label.

To implement this property, we use the one-step non-targeted FGSM method to generate 7000 adversarial examples and train a more robust model to resist the attacks. The modified



```
net = x_image

net = tf.layers.conv2d(inputs=net, name='layer_conv0', padding='same',
                       filters=16, kernel_size=3, activation=tf.nn.relu)
net = tf.layers.max_pooling2d(inputs=net, pool_size=2, strides=2)

net = tf.layers.conv2d(inputs=net, name='layer_conv1', padding='same',
                       filters=32, kernel_size=5, activation=tf.nn.relu)
net = tf.layers.max_pooling2d(inputs=net, pool_size=2, strides=2)

net = tf.layers.conv2d(inputs=net, name='layer_conv2', padding='same',
                       filters=64, kernel_size=5, activation=tf.nn.relu)
net = tf.layers.max_pooling2d(inputs=net, pool_size=2, strides=2)


net = tf.layers.conv2d(inputs=net, name='layer_conv3', padding='same',
                       filters=64, kernel_size=5, activation=tf.nn.relu)
net = tf.layers.max_pooling2d(inputs=net, pool_size=2, strides=2)

net = tf.contrib.layers.flatten(net)

net = tf.layers.batch_normalization(inputs= net, name = 'batch')

net = tf.layers.dense(inputs=net, name='layer_fc1',
                      units=1024, activation=tf.nn.relu)

net = tf.layers.dense(inputs=net, name='layer_fc_out',
                      units=num_classes, activation=None)

logits = net

y_pred = tf.nn.softmax(logits=logits)

cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_true,
                                                           logits=logits)
loss = tf.reduce_mean(cross_entropy)

gradient = tf.gradients(loss, x)[0]
optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(loss)
```

Figure 2: *The structure of adversarial training model*

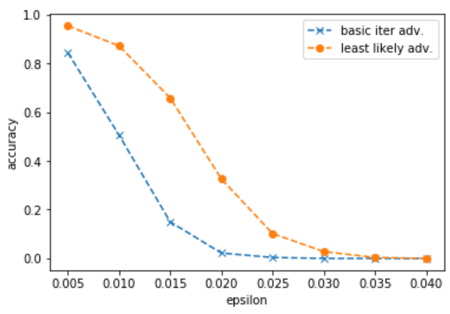


Figure 3: *The accuracy of classification model under attacks by basic iter. and iter. l.l FGSM methods and different ϵ values. The accuracy is based on 500 test images from the MNIST dataset.*

neural network structure is illustrated in Figure 2. We trained the model on 7000 generated adversarial examples and 13000 clean images with 2000 iterations. For each iteration, the batch size is 64. As a result, the prediction accuracy on 3000 adversarial examples is 98.4%. On the contrary, the accuracy on clean images is only 96.0%. The result of our implementation draws the same conclusion as the paper, which demonstrates that the label leaking effect exists when we train models on adversarial examples generated from targeted one-step attack.

The reason for this property is that one-step methods which utilize the true label perform a very simple and predictable transformation that the model can learn to recognize. If we evaluate the robustness of trained model based on these adversarial examples, the result is much more higher than it should be. Consequently, we suggest that we should not use the one-step FGSM based on true label to evaluate the robustness of adversarial training model.

iii. Comparison of Different Adversarial Attacks

In the paper[7], the author used a different targeted iterative FGSM method, which is called iterative least-likely class method ("iter. l.l").

$$\begin{aligned} &\boldsymbol{x}_0^* = \boldsymbol{x}, \\ &\boldsymbol{x}_{t+1}^* = Clip_{\boldsymbol{x},\epsilon}\{\boldsymbol{x}_t^* - \alpha \cdot sign(\nabla_{\boldsymbol{x}} J(\boldsymbol{x}_t^*, y_{LL}))\} \end{aligned} \tag{5}$$

The least-likely class is usually highly dissimilar from the predict class, so the attack will cause more serious mistake for the original model.

First, we compare the basic iterative FGSM method with the iter. l.l. We set the iteration number for each ϵ to 10 and the step size α is set to 0.06. The result is shown in Figure 3. The accuracy is calculated by the performance of classification model to resist the attacks and predict correctly the true label. Both basic anf l.l iterative FGSM attack can achieve great attack results when ϵ comes to 0.03; However, the basic iterative FGSM method has higher accuracy at the smaller ϵ values. This result can be explained intuitively, because generating the specific least likely class is much more difficult, especially when limited pixel values can be changed during each iteration.

We also tried the one-step FGSM method and compare the result with different ϵ values.

In Figure 4 and Figure 5, the value of ϵ is largely different because the pixel values of ImageNet images are in range (0,255) and the processed MNIST images are in range (0,1). The figures illustrates the performances of different adversarial attacks with various ϵ values. In Figure 4, basic iter. attack cannot effectively

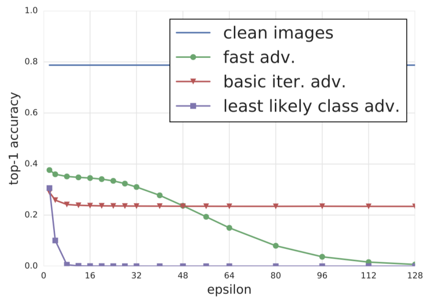


Figure 4: *The accuracy of Inception v3 under attacks by different adversarial methods and different ϵ values in paper[7]. The accuracy is based on 50000 validation images from the ImageNet dataset.*

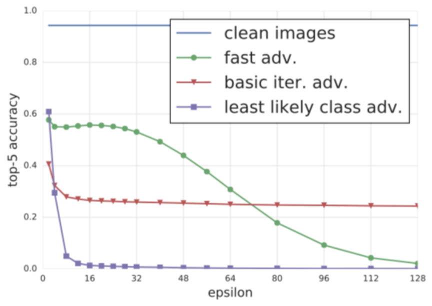


Figure 7: *The top-5 accuracy of Inception v3 under attacks by different adversarial methods and different ϵ values in paper[7].*

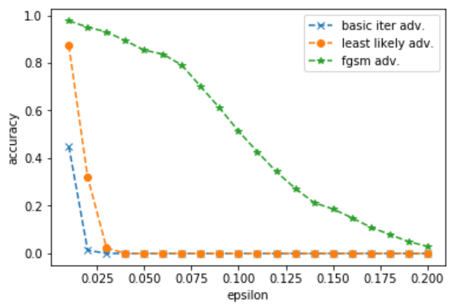


Figure 5: *The accuracy of classification model under attacks by different adversarial methods and different ϵ values in paper[7].*

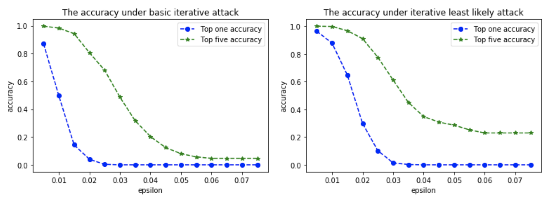


Figure 6: *The Top-1 and Top-5 accuracy under different attack methods the ϵ equals to 0.06 and iteration number equals to 10.*

cause the classification model to make mistakes. Even the ϵ value becomes larger and larger, about 1/5 images can still be predicted correctly. However, in the Figure 5, the attack succeed with the accuracy of even zero.

Second, we calculate the top-5 accuracy of classification model under different attack methods. In the case of top-5 accuracy, we need to check if the true label is one of the top-5 predictions. If the true label is in the top-5 predictions, the accuracy will be set to 1; otherwise, it is set to zero. The Figure 6 shows the top-5 accuracy is higher than the top-1 accuracy, which is easy to explain because making the prediction far away from the true label is very hard.

In Figure 7 and Figure 8, the one-step FGSM is also taken into consideration. However, in the Figure 8 the iter. l.l FGSM method remains the accuracy around 0.2. The result is quite different from the result in Figure 7. In the paper's result, the iter. l.l performs the best, and basic iter. adversarial attack performs the best in our experiment.

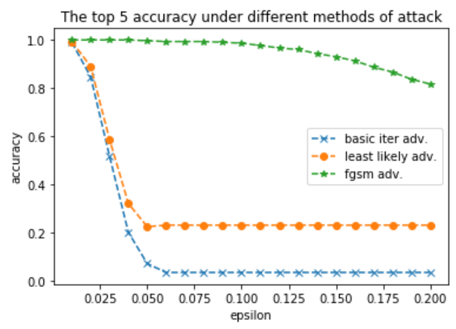


Figure 8: *The top-5 accuracy of classification model under attacks by different adversarial methods and different ϵ values in paper[7].*

V. Discussion

In our project, the experimental results are different from the paper's result, especially for the comparison on top-5 accuracy. There are many potential reasons for the differences.

First, the complexity of our model and the size of our dataset may contribute to our result. Our trained model is relatively simple and the MNIST dataset is much smaller than the ImageNet dataset used by the paper.

Second, some labels in ImageNet dataset are very similar and there may be some correlation between the related labels. For example, the ImageNet dataset labels various kinds of fish and it is hard to tell their differences. Even our attack successfully decrease the probability of one of these labels, some related labels will also be affected. As a result, the possibility of all the relative labels will decrease, the true label will still remain in the top-5 prediction. This might explain the reason for basic iterative attack not performing well in the paper. On the contrary, our training data only has ten different labels and each label is independent, attacking to one of the labels will not affect other labels. That may be the reason for the outstanding performance of basic iter. adversarial attack on MNIST dataset.

Third, it is more challenging for the attack to make the prediction model predict the least likely class, so it may requires more iterations and larger epsilon number. It is understandable that the iter. l.l adversarial attack doesn't have satisfying performance.

These are our assumptions on the possible reasons that lead our result to be dissimilar with the paper's result. We need further experiments to justify our assumptions and try more datasets to discover if the properties of datasets have influence on the result.

VI. Conclusion and Future Work

Our project is mainly focused on the FGSM-based adversarial attacks. We learned the definitions of different kinds of attacks and compared their differences.

We accomplished experiments to prove the label leaking property and compare the efficiency of various attacks. Also, we discussed the potential reasons to make our result be inconsistent with the papers' results.

Our further plan is to implement more adversarial attacks to discover the weaknesses of these gradient-base attacks and test the effect of different input data on the trained model. Based on the evaluation, we will gain a deeper understanding of the data security problem on neural networks.

References

[1] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

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

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

[4] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[5] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[6] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

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

[8] Hvass-Labs. Tensorflow-tutorials. `https://github.com/Hvass-Labs/TensorFlow-Tutorials/./master/02_Convolutional_Neural_Network.ipynb`, 2018.

[9] Alexey Kurakin, Ian Goodfellow, Samy Bengio, Yinpeng Dong, Fangzhou Liao, Ming Liang, Tianyu Pang, Jun Zhu, Xiaolin Hu, Cihang Xie, et al. Adversarial attacks and defences competition. *arXiv preprint arXiv:1804.00097*, 2018.