Adversarial Examples: Unsupervised Training through Deep Convolutional Generative Adversarial Networks

Seung Lee
CSC 4810
Georgia State University
Atlanta GA
slee185@student.gsu.edu

***Abstract*—Adversarial examples are inputs given to a machine learning model that causes it to intentionally make a mistake through perturbing data. So far, the most effective solution has been adversarial training. However, it has its major drawbacks such as lack of adaptability and learning from finite training data. In this paper I proposed the solution of using unsupervised adversarial training though deep convolutional generative adversarial networks (DCGAN) for adversarial training. The implementation was failed to run. Therefore, implementation of the model by others were observed. Unsupervised use of DCGANs appear to be a promising solution to adversarial examples to the decrease in limitation from finite training data, stability, and accuracy. More research should be done to better investigate its effectiveness.**

Keywords—generative adversarial networks, adversarial training, adversarial examples

I. INTRODUCTION

Despite continuous progress in AI technology and machine learning research, problems continue to emerge, especially on computer security. One of the most complicated but important issues is adversarial examples. Adversarial examples are inputs given to a machine learning model that are designed to trick them model into make a mistake [1].

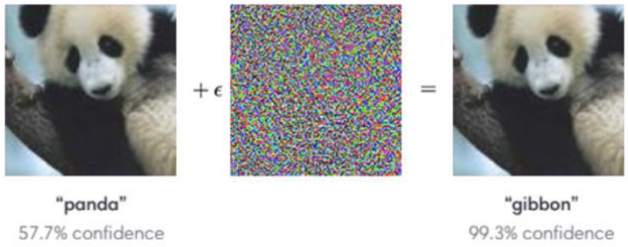

Figure 1: An attacker adds small perturbation to model to misclassify a panda image as a gibbon [1].

While a lot of solutions have been proposed, most of have failed completely, such as weight decay and dropout, and none have proven to be without major vulnerabilities. So far only two being somewhat effective: adversarial training and defensive distillation[2]. Adversarial training is creating a lot adversarial examples for machine learning model data so that the model can learn to recognize them. Defensive distillation is a training method where one machine learning model is trained to predict the output probabilities of another model. So far, adversarial training has shown the most success [1].

Despite it being the most accurate defense known, current adversarial training methods share the challenges of all current adversarial defenses.

- Lack of sufficient adaptability is an issue. Once an attacker knows about the defense used, the model is vulnerable to unrecognized attacks. An example is a "black-box attack", which is when an attacker can observe the label the model assigns to inputs and make its own imitation model to find those points to create effective adversarial examples [3].

- Dependence on limited training data. An optimal machine learning model would produce to good output for every input. Because there are so possible adversarial examples, our current training methods cannot defend against every perturbation [4].

- Defending against adversarial examples is an optimization problem that is non-linear and non-convex (multiple optimization points). We don't have the theoretical tools to describe such a problem [3].

Overcoming all of these challenges will require a long time of research and development. With adversarial training being the most successful method that we have, a practical approach is to optimize the traditional method and decrease vulnerabilities as much as possible. In this research, we try to develop an effective solution that bypasses the limitation having to manually inputting perturbed samples.

II. Generative Adaptive Networks

One of the biggest contributers to research on adversarial training is Ian Goodfellow. He is not only at the forefront of research on adversarial examples, but is also known as the father **Generative Adversarial Networks (GAN)**. This is a framework of two competing neural networks, a **generative network** and **discriminative network** . The significance is that it is able to generate new false inputs automatically and the two competing networks mimicks an adversarial attacker vs model.

A. Generative Adversarial Network Architecture

- The generative network creates instances out of **random noise**. It consists of a **generator** that uses the noise to synthesize inputs that is meant to look like training data.
- At the same time, we also have actual training data inputs.
- Both generated inputs and real inputs lead to the **discriminator** which determines if the input it is receive is real or fake (synthesized) [12].

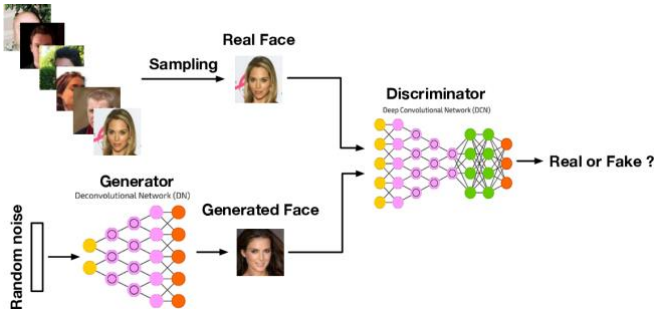

Figure 2: A visual example of the GAN architecture explained so far. for image inputs. Here is the discriminator tries to determine if the incoming image is real or fake [11].

- The discriminator is part of the **discriminative network** which continues by using the discriminator output, calculating a loss, and with that loss backpropagates to obtain gradients of the discriminator and generator to adjusts their weights with that loss [12].

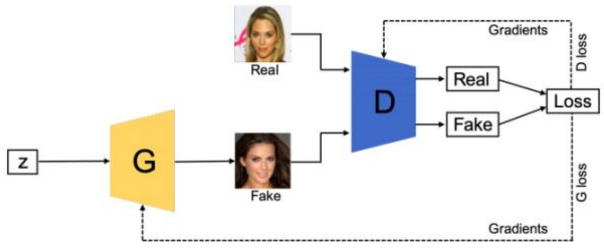

Figure 3: Here we see the paths to the discriminator, the output, the loss, and the path that leads to back to the generator and discriminator to adjust weights using gradients [11].

.

III. Deep Convoluted Generative Adversarial Networks

Since the founding GANs, numerous different types have been proposed and implemented. For image classification, DCGAN (Deep Convoluted Generative Adversarial Networks) is ideal because **convoluted neural networks** require less pre-processing for training images than regular neural networks and have architectural constraints that make them more stable than traditional GANs. This stabilizes learning and is critical in preventing one of the major weakness in traditional GANs which is "mode collapse" [10]. A DCGAN uses the same general architecture of the original GAN but uses convolutional neural networks, which are similar to regular neural networks in having weights and bias but the layers have 3 dimensions: weight, height, and depth. The illustration below shows multiple convolutions the input goes through [11].

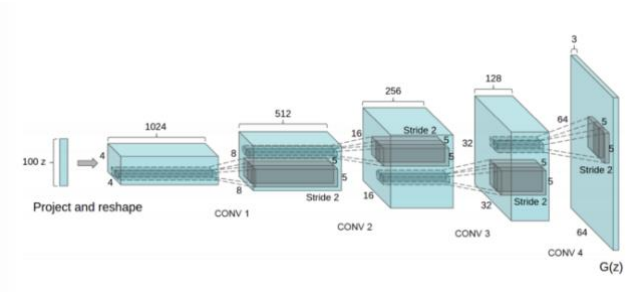

Figure 4: Generator of a DCGAN. Rather than fully connected layers, the generator has convolutions that reshape and flatten out. It begins with a uniform noise distribution as input that travels down until it is reshaped into a 4-dimensional tensor and used at the start of the convolutional stack [7].

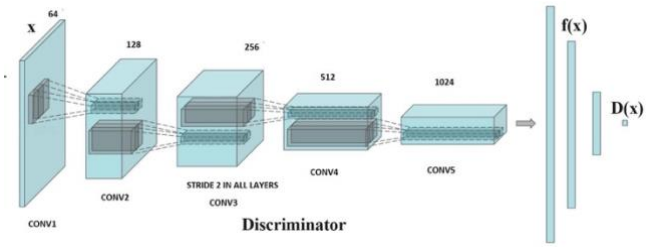

Figure 5: The discriminator behaves in reverse and fed into a single sigmoid output [7]

Unlike regular GANs, no fully connected or pooling layers are used. The architectural constraints in creating a DCGAN is shown below.

1. Convert max-pooling layers to convolution layers with larger or fractional strides
2. Convert fully connected layers to global average pooling layers in the discriminator
3. Use batch normalization layers in the generator and the discriminator.
4. Use leaky ReLU activation functions in the discriminator [7]

IV. Unsupervised Learning

One of the major strengths of GANs is ability for unsupervised learning. Unsupervised learning looks for patterns in a data set with no labels and minimum human supervision. While we still do have a finite amount of training data, our DCGAN is able to constantly create new inputs from the generator without having to manually add adversarial samples. In unsupervised learning, the outputs do not correspond to the inputs. This improves on the issue with linearity in traditional adversarial training models [12].

V. Implementation using Python via PyTorch and TensorFlow

PYTORCH and TensorFlow was installed along with Anaconda3 on MacOSX. More details of this is shown in the user manual.

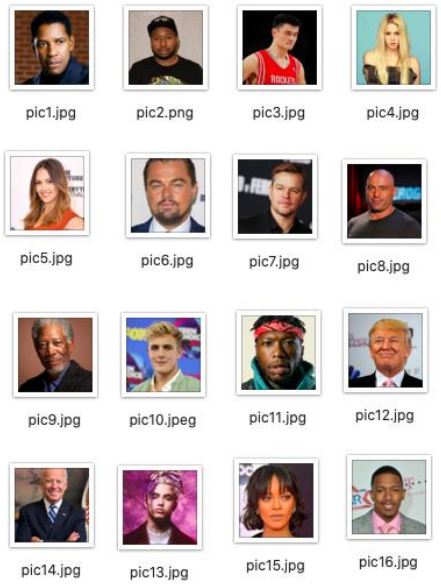

Figure 4: Training Set consisted of selected celebrity images from Google Images.

The first thing that was done is to create an image data set used for our training set. I chose celebrity pictures that show their faces clearly. 16 images were used and then the sizes were cropped to be the same size, using *BIRME*.

A. Set-Up of Discriminator and Generator

Using Python, the discriminator and generator classes were developed with neural networks through the PyTorch library, while using PyCharm as the IDE. The code was designed to matched the same DCGAN architecture proposed by Radford [7].

```
import torch
import torch.nn as nn


class Discriminator(nn.Module):
    def __init__(self, channels_img, features_d):
        super(Discriminator, self).__init__()
        self.disc = nn.Sequential(
            # input: N x channels_img x 64 x 64
            nn.Conv2d(
                channels_img, features_d, kernel_size=4, stride=2, padding=1
            ),
            nn.LeakyReLU(0.2),
            # _block(in_channels, out_channels, kernel_size, stride, padding)
            self._block(features_d, features_d * 2, 4, 2, 1),
            self._block(features_d * 2, features_d * 4, 4, 2, 1),
            self._block(features_d * 4, features_d * 8, 4, 2, 1),
            # After all _block img output is 4x4 (Conv2d below makes into 1x1)
            nn.Conv2d(features_d * 8, 1, kernel_size=4, stride=2, padding=0),
            nn.Sigmoid(),
```

Figure 5: Screenshot of discriminator function in Python. Here the convolutions reshape back to how the first convolution looked from the generator model

B. Weight initialization and test model

The function for weight and testing were created. For weight, the mean was set to 0 and the standard deviation was set to 0.2.

C. Training Set Up

We then moved onto our training file where the hyperparameters, optimizers (using AdamOptimizer), and loss functions were initialized. We also considered training an MNIST data set as a testing procedure. We set the labels for discriminator outputs as “real” or “fake”. The training loop was then set up that represented the DCGAN process with calculation of gradients for backpropagation and constant updating.

D. Troubleshooting/Results

Unfortunately, my actual code was not able to function due to my IDE not detecting the torch modules from PyTorch, with the error shown below.

```
    import torch
  File "/Applications/PyCharm.app/Contents/plugins/python
/pydev_import_hook.py", line 21, in do_import
    module = self._system_import(name, *args, **kwargs)
ModuleNotFoundError: No module named 'torch'
```

FIGURE 6: Despite having installed PyTorch and verifying the modules being on my computer, this error was unsolved after numerous attempts.

It was likely due to directories of the installed modules not being the same directory used by PyCharm. This demonstrates that understanding directories well is critical to implementing machine learning techniques.

As a result, I tried using the written code for a web browser version of TensorBoard, which did exist through Google Collab as a notebook extension, but was unable to incorporate my data set into a browser environment.

E. RESEARCHED RESULTS

Despite my inability to train my data set, many other people have also tried to this procedure of DCGAN for image recognition.

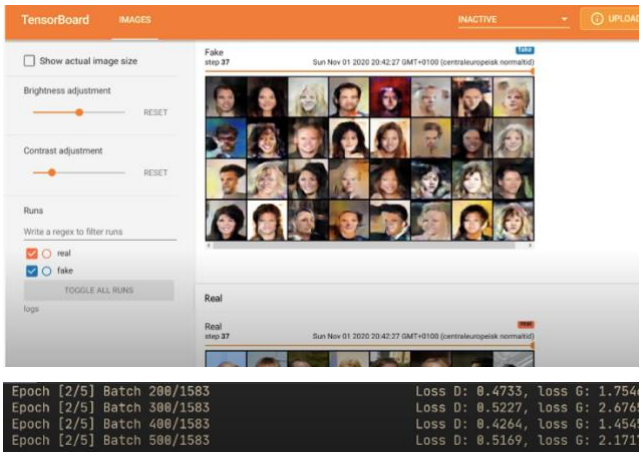

Figure 7: A screenshot of Aladdin Persson's demonstration TensorBoard running the DCGAN program [8]. https://www.youtube.com/watch?v=IZtv9s_Wx9I

In the figure 7, the program does seem effective at separating at real from fake images, although far from optimal as the 2nd screenshot below shows the penalty for errors. Persson stated that more training time will likely improve accuracy, since the gradient adjustments from the losses is how the discriminator and generator learn. In fact, this is shown below.

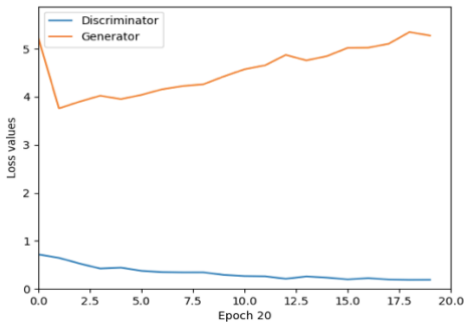

Figure 8: A screenshot of the graph of Minjae Kim's implementation of the procedure. The inverse relationship in losses for generator vs discriminator is noticeable. As the generator gets penalized in larger values, the loss for the discriminator drops, which is a sign of a gradual decrease in error rate [9].

CONCLUSION

Adversarial examples remain one of the most difficult and important challenges to cybersecurity for machine learning models. While adversarial training remains the most successful defense against adversarial examples, the standard model has weakness that must be improved upon and generative adversarial examples offer a more efficient solution to the traditional adversarial training model. Successor models of the GAN framework, such as Deep Convoluted Generative Adversarial Network can provide a more accurate, stable architecture, especially for specific tasks. Because adversarial training is such a difficult problem, more research on using GAN for adversarial defense to progress in finding an optimal solution.

ACKNOWLEDGEMENTS

I would have to thank Ian Goodfellow for the important work he has done in providing interesting research on this subject. It is possible I would not know about this topic at all without his contribution to the field. He has not only allowed the information of adversarial examples to more well-known and widely available through countless lectures, articles, and research papers, but has laid foundational work that other researchers that I have cited build off of.

REFERENCES

[1] Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In International Conference on Learning Representations, 2015.

[2] Zhang J, Li C. Adversarial Examples: Opportunities and Challenges. *IEEE Trans Neural Netw Learn Syst*. 2020

[3] 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 (ASIA CCS '17). 2017.

[4] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik and A. Swami, "The Limitations of Deep Learning in Adversarial Settings," *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, Saarbrucken, 2016

[5] SZS13] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. "Intriguing properties of neural networks". arXiv preprint arXiv:1312.6199. 2013

[6] Papernot, Nicolas, P. McDaniel, Arunesh Sinha and Michael P. Wellman. "Towards the Science of Security and Privacy in Machine Learning." *ArXiv* abs/1611.03814 (2016)

[7] Radford, A., Luke Metz and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." *CoRR* abs/1511.06434 (2016)

[8] Persson, A. PyTorch DCGAN Tutorial - Improving the architecture with CNNs. *YouTube*. 2020.

[9] Kim, Minjae. PyTorch implementation of Deep Convolutional Generative Adversarial Networks (DCGAN). *Github*. 2017

[10] I. J. Goodfellow. Advances in Neural Information Processing Systems 2016 tutorial: Generative adversarial networks. arXiv preprint arXiv:1701.00160, 2017.

[11] Wang, Zhengwei, Qi She and T. Ward. "Generative Adversarial Networks: A Survey and Taxonomy." *ArXiv* abs/1906.01529 (2019)

[12] Brownlee, Jason. "Master Machine Learning Algorithms: discover how they work and implement them from scratch". *Machine Learning Mastery*, 2016