Generating Adversarial Attack Examples using GANs

Pankti Hitesh Parekh, Kedar Prasad Karpe, Vaibhav Sahu
{pankti81, karpenet, vaibhavs}@seas.upenn.edu

December 2022

1 Introduction

The use of Machine Learning, and Deep Learning in particular is on the rise being to applied to diverse fields such as scientific research and data science, to everyday algorithms in our smartphones and gadgets. Nowadays, the applications have expanded into realms of much higher stakes where scope of error itself comes at much cost, let alone the threat of adversaries! Adversarial Machine Learning, hence often not only focuses on new ways adversaries can find to attack models but also defenses against them.

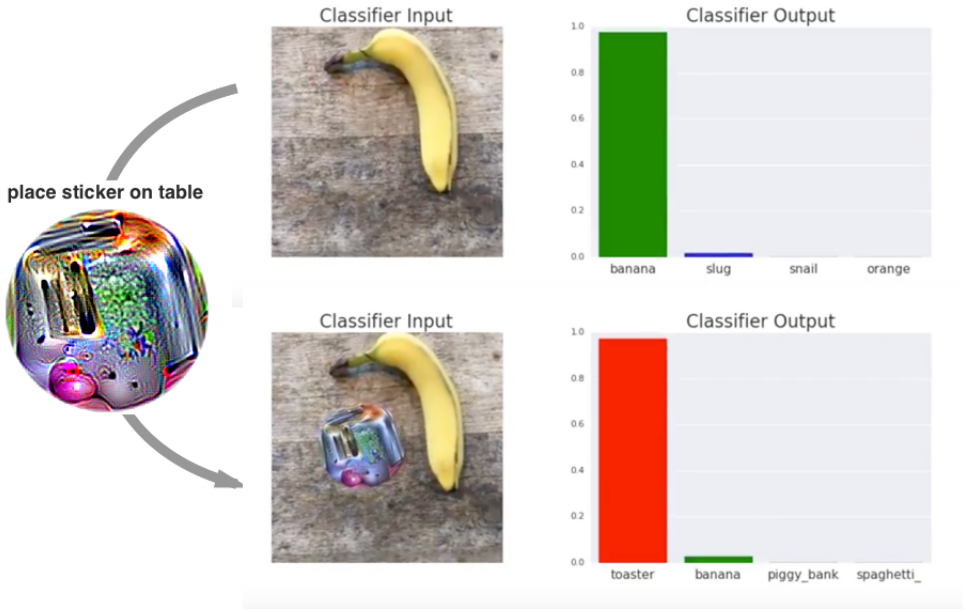


Figure 1: Adv-patch [1]: Example of an adversarial attack on a Vision Model

At the same time, Generative Adversarial Networks (GANs) have also been gaining more and more attention on how realistic examples they have been able to create.

We plan to fuse these two ideas to actually be able to train GANs to do adversarial attacks on trained models.

2 Abstract

We will be attacking the all-cnn network trained on the CIFAR-10 dataset, that we have spent a fair bit of time analysing in our homework. The network in its original essence is no way close to being robust to even the weakest of optimisation based attacks such as L-BFGS that we saw in one of our assignments. We however, will be attacking this model in a semi-

whitebox setting instead so we will not be accessing the parameters or even the structure of our model at all. This is a much more realistic assumption as an adversary. Our model is motivated almost completely from the work of Xiao et. al. [3] and their model called Adv-GAN. We modify this to the context of CIFAR-10 and train our model and make examples on this dataset instead of the MNIST dataset to generate new results. Finally, we will try how Madry Lab's PGD-training [2] on all-cnn fares against the adversarial examples generated by our GAN.

3 Generating Adversarial Images with Adversarial Networks

We will be summarizing our approach and implementation motivated from the Xiao et. al [3] paper. While, the models we use are relatively simpler, the training framework is not. Training even a small-scale GAN in this context was no less than a challenge.

3.1 GAN model architecture

We will be using Deep Convolution GANs as implemented in [4]for generating perturbations for each of these images. The architecture of our GAN is fairly simple. The discriminator is made up for convolutional and batch normalisation layer blocks with max pooling to reduce dimensions. The generator takes in a random variable and from this noise and upsamples it using transposed convolutions followed by Instance Normalisation layers for stability, to generate perturbations for our images.

3.2 Training framework

The GAN loss ($\mathcal{L}_{GAN}$) is given by the BCE loss. However, we have other loss terms that actually are responsible for generating adversarial examples:

$$\mathcal{L}_{adv}^{f} = l_f(x + G(x), t) \tag{1}$$

where f is our targeted model. In our classification task, we will take this loss to be Cross-Entropy loss. The other term is a hinge-loss term to ensure our perturbations are actually adversarial and hence, are very small in magnitude. This is given by:

$$\mathcal{L}_{pert} = \max(0, \|G(x)\| - C) \tag{2}$$

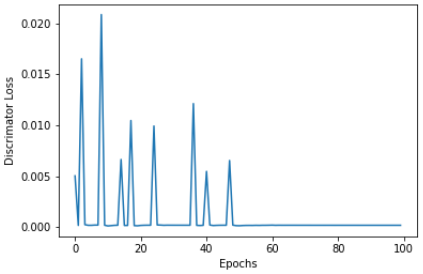
where C is set to a small number to clip perturbations whenever their magnitude increases C.

3.3 Training and Hyperparameter tuning

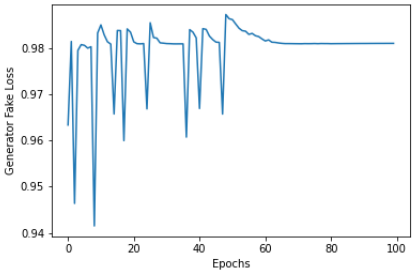
Training GANs is hard enough. DC-GANs particularly have a problem. When the transpose convolution kernel sizes are not chosen carefully, we get checkerboard patterned noise in the images. This was the case for us and we will get to see some checkerboard patter noise in our images. However, the real challenge was to actually balance the hyperparameters that scale the individual loss components. We need the images to look as real as possible, yet fool the network and hence tweak the λ accordingly at the same time keeping C small!

$$\mathcal{L} = \lambda_G \mathcal{L}_{GAN} + \lambda_{pert} \mathcal{L}_{pert} + \lambda_{adv} \mathcal{L}_{adv}$$

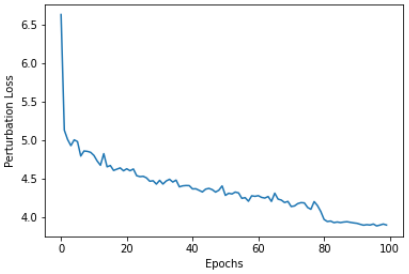
Effectively, we only need to scale two of them but in order to make the generator learn faster, we scaled up the its loss instead of changing the learning rate for its optimizer. We used ADAM optimizer for our implementation. Here are the loss profiles for our training.



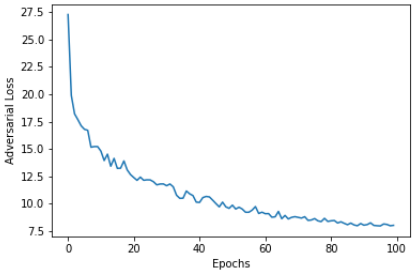
(a) Discriminator-loss vs Epochs



(b) Generator-loss vs Epochs



(c) Perturbation-loss vs Epochs



(d) Adversarial-loss vs Epochs

The generator and discriminator loss do not drop significantly as the network learns. However, it definitely learns to reduce the perturbations as well as fool the network as it trains.

3.4 Results

The accuracy after attack on training and test images was 5% and 13% respectively. This is still good considering we are not accessing the weights of our model and the perturbation noise is not too much. Here are some visual results from our model:

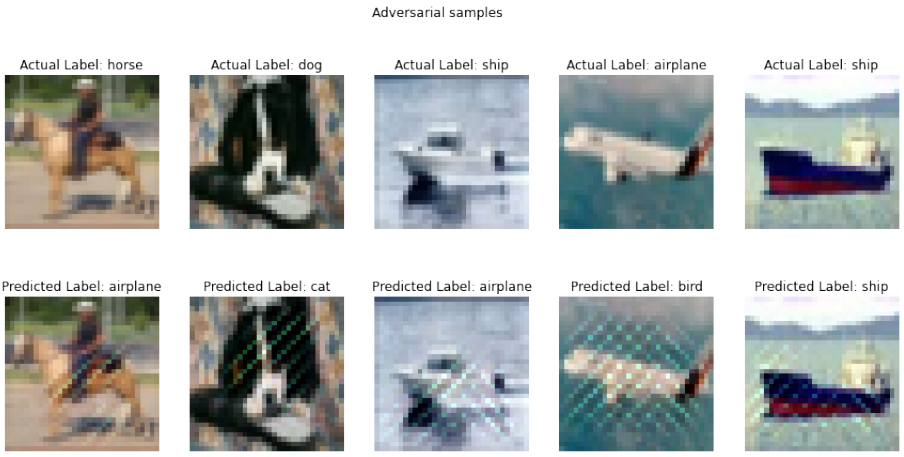


Figure 3: Results: Adv-GAN

4 PGD Defenses vs AdvGAN

4.1 Project Gradient Descent

Projected gradient descent (PGD) is a universal "first-order adversary", i.e., the strongest attack utilizing the local first order information about the network. PGD attempts to find the perturbation that maximises the loss of a model on a particular input while keeping the size of the perturbation smaller than a specified amount referred to as epsilon. This constraint is usually expressed as the L^2 or L^∞ norm of the perturbation and it is added so the content of the adversarial example.

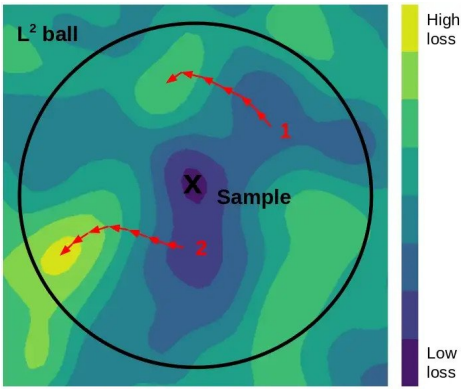


Figure 4: Projected gradient descent finds a high loss adversarial example within the L^2 ball.

To perform generate a PGD adversary we iteratively update the perturbation δ to the original images according to the following steps:

1. Compute the gradient of $\mathcal{L}_{target}$ with respect to δ: $\nabla\mathcal{L}_{target}$
2. Take a step in the direction of the gradient: $\delta = \delta + \alpha\nabla\mathcal{L}_{target}$
3. Project δ onto the feasible region defined by the constraints on δ: $\delta = P(\delta)$
4. Here, α is the step size or learning rate, and $P(\delta)$ is the projection operator that maps δ onto the feasible region. The iterative process is repeated until the model produces the desired output y', or until some other stopping criterion is met.

4.2 PGD Adversarial Samples

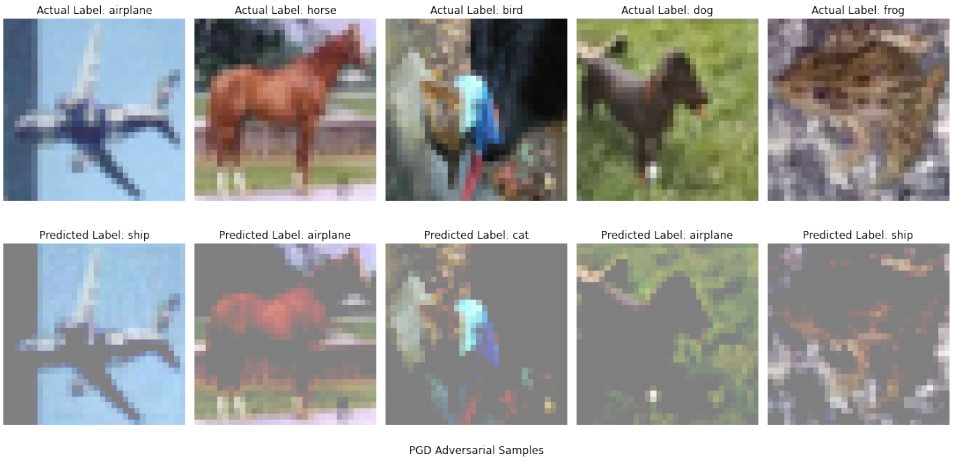


Figure 5: Adversarial Samples Generated with Projected Gradient Descent

5 Conclusion and Potential Improvements

The method of generating Adversarial examples using GANs is actually on of the strongest black box attacks. The attack is also easily generalisable but this requires distillation of model on a dataset in order to be able to target it. However, in they are still very weak compared to optimisation based White-box attacks.

A more generalisable attacks recently have actually made use of attacking images in the latent space of GANs.

Apart from this, the image quality could definitely have been improved with more careful upsampling in the generator. It seems like a much better practice to use conventional interpolation techniques for upsampling followed by convolutional blocks instead of transposed convolutions directly.

Training GANs on adversarially trained model was not in the scope of our project, however, it would be interesting to observe the effects of training the Adversarial GAN network on a model already trained with adversarial samples.

References

[1] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch.

[2] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2017.

[3] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks.

[4] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2017.