IFT6261 - The implications of adversarial attacks on deep learning methods

Eric Pfleiderer
Jeremy Rabes

April 19^{th} 2022

1 Introduction

In the last decade, deep learning has been touted as a catch all solution to traditionally difficult computer tasks like computer vision and natural language processing. The popularization of neural networks has indeed provided a powerful framework for function estimation. However, these methods are known to be obscure and the reasoning behind their decision-making is difficult to analyze. As a consequence, users often treat deep learning models as black boxes. In this paper, we argue that such a methodology is susceptible to leave the user unaware of adversarial attacks, as it would successfully fool the model without raising significant suspicions. In order to do so, we train a simple classifier on the MNIST dataset and explore four different methods to create adversarial samples: the Fast Gradient Sign Method (FGSM), the Projected Gradient Descent (PGD), Genetic Algorithms (GAs) and Generative Adversarial Networks (GANs). We offer an overview of each method, compare their performance and discuss the possible implications of our results.

2 Methodology

As a simple but rigorously defined context, we chose to use the very popular MNIST dataset, which consists of 80000 grayscale images of digits ranging from 0 to 9. Images are encoded in $\mathbf{R}^{28\times 28}$ with a single color channel. No transformations were applied to the data.

We then trained a simple classifier to predict the probability of an example $\mathbf{X}_i$ to be part of the correct class $\mathbf{Y}_i$. The architecture of the classifier is largely irrelevant for the purposes of our demonstration, so we chose a simple CNN [2] that regularly attained an accuracy of over 90% on our test set of 10000 images. The classifier's parameters remain frozen during the rest of the experiment.

To roughly estimate the difficulty of the task itself (which is fooling the classifier), we chose to compute the number of potential attack targets. A vulnerable attack target is a sample $\mathbf{X_i}$ from our dataset $\mathbf{D}$ for which the certainty (or density) of the model's prediction is spread a cross two classes, instead of being concentrated in a single one. This most often is the case with classes that are semantically similar. For example, an image of a 4 may be easily misinterpreted for a 9, but it will very rarely be misinterpreted as 3. This classification difficulty is not due to random noise or egregious samples, but is rather due to the semantic structure that some classes share. More rigorously, if our model predicts $\tilde{\mathbf{Y}}_\mathbf{i} = f_c(\mathbf{X_i})$, then the sample $\mathbf{X}_i$ is considered a candidate if

$rank_1(\tilde{\mathbf{Y}}_\mathbf{i}) - rank_2(\tilde{\mathbf{Y}}_\mathbf{i}) \leq t$, where $f_c : \mathbf{X}^{28\times28} \rightarrow \tilde{\mathbf{Y}}^{10}$ is our classifier, $rank_i$ is the *i'th* highest element of $\mathbf{Y}_i$ and t is an arbitrary threshold between 0 and 1. Figure 1 shows the potential number of candidates as a function of t and is discussed in further details in section 3.

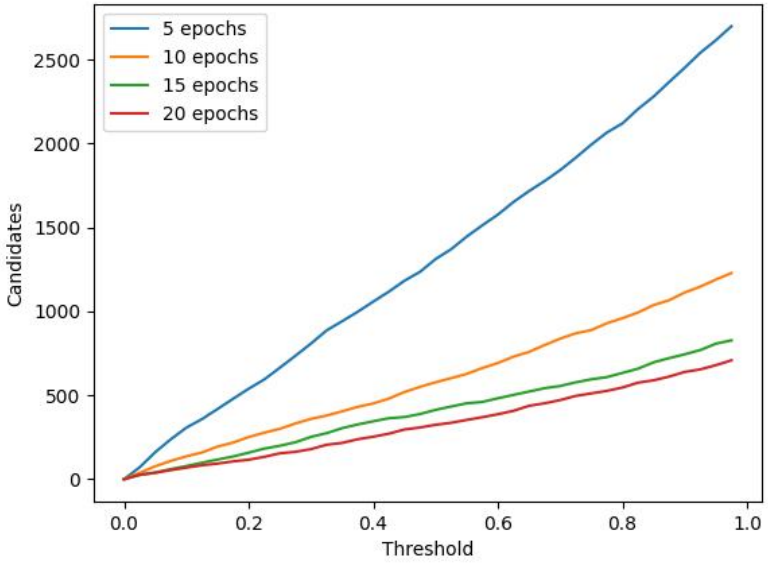


Figure 1: Number of potential candidates for adversarial attacks as a function of threshold t

Finally, we proceed with the generation of adversarial samples $\mathbf{X}_a$ using four methods: FGSM, PGD, GAs and GANs. To evaluate the quality of our adversarial samples, we use the following loss

$$\mathcal{L} = \underbrace{P_{f_c}(Y|X_a)}_{\mathcal{L}_u} + \epsilon \underbrace{(X - X_a)^s}_{\mathcal{L}_s}$$

which we subdivide in two terms, the uncertainty loss $\mathcal{L}_u$ and the sameness loss $\mathcal{L}_s$. The uncertainty loss corresponds directly to the model's probability to predict the correct class, while the sameness loss is the euclidean distance set to an arbitrary power. Minimizing the first term pushes the model towards misclassification, while minimizing the second punishes the adversarial sample $\mathbf{X}_a$ for diverging from the original sample $\mathbf{X}$. The parameter ϵ regulates the total pressure applied by each term.

Symbol	Description
$\mathbf{X}_i$	MNIST sample example
Y_i	MNIST sample label
$\tilde{\mathbf{Y}}_i$	classifier prediction distribution
f_c	classifier function
p_{f_c}	probability density under the classifier
$\mathbf{X}_a$	adversarial MNIST sample
f_a^i	attack function i
$\mathcal{L}_u$	uncertainty loss
$\mathcal{L}_s$	sameness loss
s	sameness power
ϵ	loss coefficient

2.1 FGSM

The FGSM creates adversarial samples by leveraging its access to the classifiers parameters. Given a sample $\mathbf{X}_i$, we use our classifier f_c to obtain a probability distribution $\tilde{\mathbf{Y}}_\mathbf{i}$ over the predicted classes. We can then compute the gradient of the loss with respect to each of the classifiers parameters to know how each pixel should be modified in order. The equation for creating an adversarial sample with the FGSM is as follow :

$$\mathbf{X}_a = \mathbf{X}_i + \gamma * sign(\nabla_\theta J(\theta, \mathbf{X}_i, \mathbf{Y}_\mathbf{i}))$$

The size of the perturbations is parameterized by a learning rate γ. This method is significantly limited but can still produce decent results. The significant perturbations needed to fool the classifier usually require a large γ, which increases the approximation error made by the method when performing a linear step. This problem is somewhat solved by using this method iteratively, which leads us to our next method.

2.2 PGD

The PGD method is a generalization of the gradient descent algorithm. It can create much finer solutions by iterating the FGSM with smaller learning rates γ. This alleviates the high approximation error from the FGSM.

2.3 Genetic Algorithm

Genetic algorithms (GAs) are a class of **heuristics** that are inspired by the process of genetic evolution. Given a current solution $\mathbf{X}_i$, we first create a population by cloning. Then, we iteratively evaluate the population's quality, make its most fit members reproduce and then apply random mutations. Over time, the population's fitness will generally increase and we will be left with an optimal solution. The advantages and disadvantages of GAs are discussed in further details in section 3. We offer the following pseudocode as a high level abstraction of our implementation:

$X_i : example_i$
$Y_i : target_i$
$population \leftarrow copy(X_i)$
for *epochs* **do**
  $quality \leftarrow evaluate\_quality(population)$
  $rank \leftarrow argsort(quality)$
  $parents \leftarrow select\_parents(population, rank)$
  $children \leftarrow generate\_children(parents)$
  $children \leftarrow mutate\_population(children)$
  $population \leftarrow children$
end

2.4 GAN

This section's work is based on the paper of Xiaosen Wan et al.(2019). We first train a GAN on our MNIST dataset and then retrieve its generator. We then fine tune the generator by training it a

second time on an dataset of adversarial samples (transfer-learning) created by our other methods. Once the generator is retrained, we can sample its latent space and create many more adversarial samples. The method is shown in more detail in the figure 2.

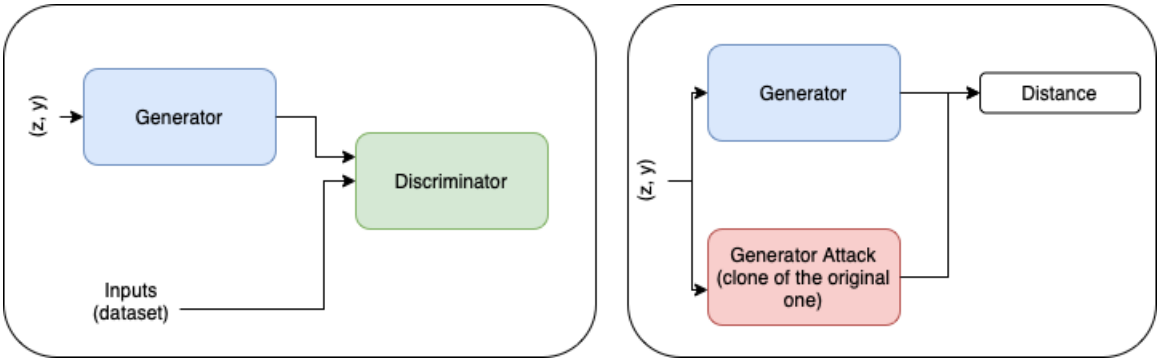


Figure 2: Architecture of the GAN method to create adversarial samples.

3 Results

For easier comparison, we applied each method on the same MNIST sample, which is shown in figure 3. The classifier originally had high confidence ($\geq 95\%$) in the predicted class "9", which is the correct label for this sample. However, this example was selected using the insight derived from figure 1. Figure 4 shows the result for the FGSM while figure 5 shows the adversarial sample created by PGD.

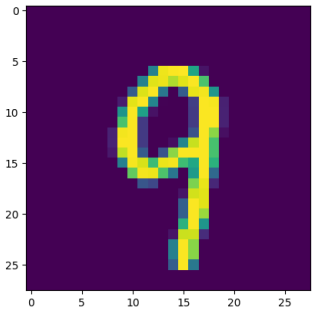
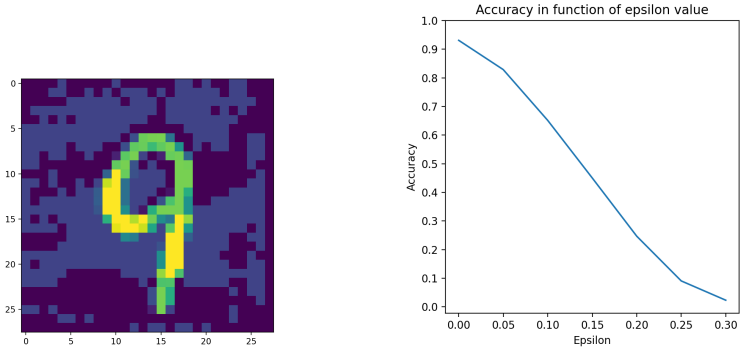
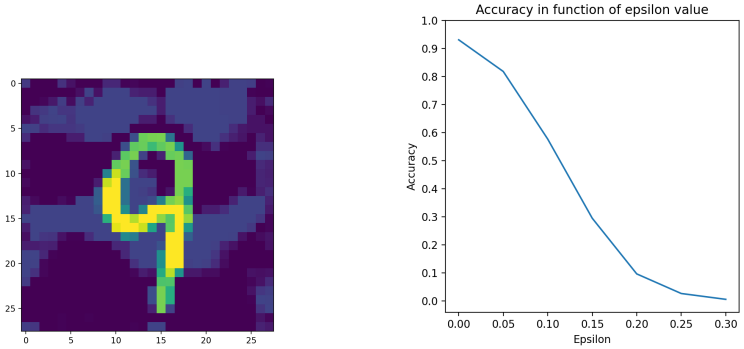


Figure 3: Original adversarial attack target



(a) Adversarial sample (b) Accuracy

Figure 4: FGSM

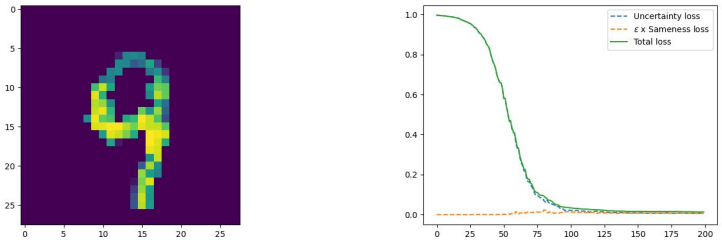


(a) Adversarial sample (b) Accuracy

Figure 5: PGD

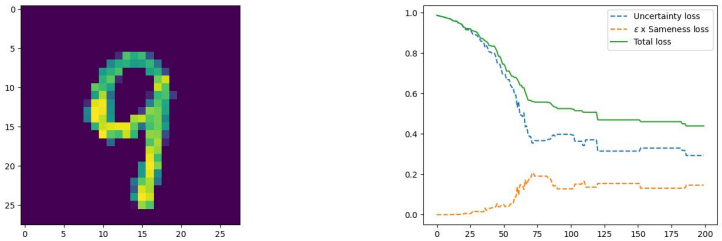
Figures 4 and 5 show that PGD delivers slightly finer solutions than the FGSM. Distortions are smoother and the the models predictive certainty towards the correct class drops faster than with PGD. Perturbations are not restricted to non-zero pixels and this creates a noticeable transformation of the data. However, the pixels creating the structure of the image seem relatively unaffected in comparison. Although we obtained an acceptable performance, these types of simple adversarial attacks may be easily noticeable as they significantly change the image's statistics (mainly its mean).

We then applied the genetic algorithm and this time decided to limit the perturbations to pixels that already have a non-zero values. In so doing, background pixels will never be altered. However, this restrictions also implies that dead pixels will never be able to be reborn. In other words, a pixel that reaches a value of 0 will be stuck with that value. In figures 6, 7 and 8, we present three experiments: one where the parameter ϵ of the loss function was set too low, one where it was set to high, and one that was selected to be adequate.



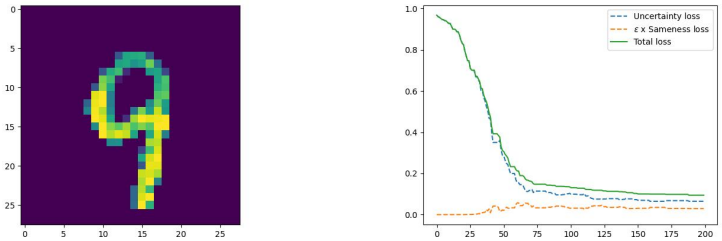
(a) Adversarial sample (b) Loss

Figure 6: Experiment with low epsilon



(a) Adversarial sample (b) Loss

Figure 7: Experiment with high epsilon



(a) Adversarial sample (b) Loss

Figure 8: Experiment with appropriate hyperparameters

In figure 7, we see that setting the parameter ϵ too low allows us to drive the overall loss towards 0. However, the resulting adversarial sample is significantly distorted. Although it is now easily misinterpreted for a 4, the new image is so far from the original one that it might as well be considered completely different. In figure 7, we have the inverse case: a high epsilon results in an adversarial image that is close to the original, but the loss remains high and may even fail to converge entirely. The ideal scenario is the one presented in figure 8, where a compromise is made between the attained performance and the distance between the adversarial sample and the original. In this case, we see that the image still represents a 9, but its semantic structure is somewhat ambiguous and could be interpreted as a 4. This is indeed the case, as the classifier model ended the simulation with a confidence of over 90% that this image should be classified as a 4.

Another relevant parameter of the loss function is its parameter s, which is the exponent of the sameness loss. This parameter imposes a prior on the mutations. A high sameness power will

harshly punish pixels that deviate marginally from their original value, while a low sameness power will be more lenient. Imposing a high sameness power will often result in many pixels slightly changing their values, while a low sameness power may yield more diverse results (many pixels changing slightly or a few changing drastically). In figure 8, we raise the sameness loss to the fourth power, which resulted in relatively diffuse perturbations that are less noticeable but still allows the genetic algorithm to completely fool the classifier.

While FGSM and PGD have the advantage of being simple to implement and may offer acceptable performance, both are white box attacks, meaning that both of these methods rely on having access to the classifiers parameters to be able to compute a gradient. This makes them less applicable in real world scenarios, where the attacker does not usually have access to the model itself, but rather only has access to its output. This is where the genetic algorithm truly shines, as all it needs to find a solution is the certainty of the classifier in each of the predicted classes (its normal output). In this context, genetic algorithms are then considered black box attacks, as the assumptions made about the information at our disposal are much softer. They are more realistically applied to real-world scenarios and offer similar if not better performance than FGSM and PGD.

One of the most significant disadvantages of genetic algorithms is that they must evaluate the quality of a population at every step. This causes important problems when this evaluation is time consuming, as it often is with more complicated simulations. However, deep learning models are notorious for taking a long time to train but a short time to perform inference. This guarantees that our genetic algorithm will evaluate populations quickly and allows us to ignore one of the important considerations of GAs.

Figure 9: GAN classic samples

In figure 8, we show the outputs of our GAN which was trained to reproduce the original MNIST distribution. Unfortunately, we were unable to fine tune it on an adversarial dataset due to time constraints. GANS are notoriously difficult to train and require high variance in their training set. Although our methods were implemented efficiently, creating enough adversarial samples to train our GAN would take many hours. It is however a relatively straight forward task, and would reward us with an even more efficient way of creating adversarial images.

4 Conclusion

We have shown that different types of adversarial attacks, including FGSM, PGD and GAs, can produce similar results and fool a well trained image classifier. However, some attacks are more

representative of the real world; while the FGSM and PGD methods require access to the classifiers parameters, the genetic algorithm only requires its output and is therefore more practical. Although MNIST is a popular dataset, creating adversarial images that are indiscernible to the human eye is very difficult because of its inherent simple semantic structure. We suspect that applying these methods on richer images with multiple color channels would increase the number of degrees of freedom and realistically make our task even easier, which would be an interesting pathway to investigate in further works,

5 References

References

[1] Xiaosen Wang, Kun He and John E. Hopcroft, "AT-GAN: A Generative Model for Adversarial Transferring on Generative Adversarial Net", 2019.

[2] https://github.com/EricPfleiderer/IFT6261/settings