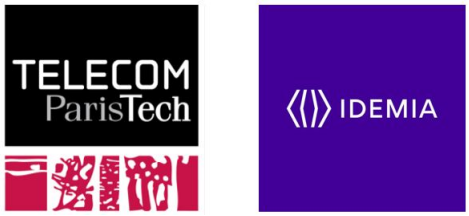


Adversarial Examples

Analysis of state-of-the-art attack/defense strategies and underlying causes of adversarial examples

Alexandre MARTIN
MSc Machine Learning - Telecom ParisTech
alexandre.martin@telecom-paristech.com

July 9th 2019

1 Abstract

This project has been carried out as part of the MSc in Machine Learning at Telecom ParisTech in partnership with IDEMIA. Formerly known as Morpho S.A.S, IDEMIA is a global leading company specialized in augmented identity solutions, biometrics identification and digital security. By being at the forefront of computer vision technologies, IDEMIA is particularly concerned about adversarial samples which cause Deep Neural Network (DNN) models to make predictions different from ground truth with high confidence. The project's objective was to investigate the effectiveness of state-of-the-art attack/defense techniques on MNIST dataset while defining ways to evaluate DNN models' robustness.

2 Introduction

Deep learning is at the heart of the current rise of artificial intelligence. In the field of computer vision, it has been used with great success for applications ranging from biometric authentification, facial detection to autonomous driving. While deep neural networks have demonstrated impressive results (often beyond human capabilities) in solving complex problems, recent studies have shown that they would be vulnerable to so-called *adversarial examples* which represent subtle pertubations that when added to an input image would drive state-of-the-art models to predict output different from ground truth with high confidence. The accuracy of deep neural networks can dramatically deteriorate in the face of adversarial examples (Biggio et al., 2013; Szegedy et al., 2013; Goodfellow et al., 2014).

For images, such pertubation are often too small to be perceptible to the human eye, yet they could completely fooled deep learning models. The vulnerability to adversarial examples becomes one of the major risks for applying deep neural networks in safety-critical environments. As presented in a subsequent section, we show that none of these deep learning models are fully bullet proof against adversarial examples. In fact, we posit that in general the more sophisticated a model is the more likely it is vulnerable to adversarial attacks. This could have profound implications for safety-critical environments where deep learning techniques is being applied.

3 Definition of an adversarial example

An adversarial example is a signal to which small carefully crafted noise are applied which push a model of machine learning to make false predictions.

It is already well known that linear models extrapolate and their behavior can be quite pathological outside the region where training data is concentrated. This extrapolation is due to the fact that, according to the definition of a linear model, each entity has the same partial slope, regardless of its value or the value of the other entities. If we manage to move our input away from the learning data region in a direction perfectly aligned with the decision limit, we will access a part of the decision space for which the model is almost certain that it corresponds to a different category.

4 Project Presentation

We have used the MNIST dataset which consist of 60K hand written digits images in the training set and 10K in the test set. MNIST images have a 28X28 pixel resolution and a channel of 1 (gray scale). This low resolution was particularily convient for running our attacks and subsequently our defenses across the whole dataset with a reasonable execution speed and calculation capacity in GPUs via Colab and Kaggle.

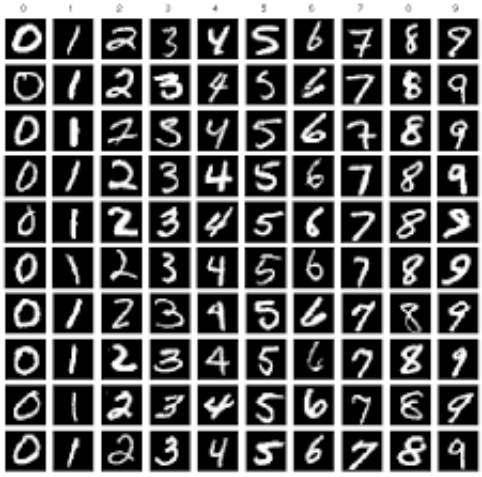


Figure 1: MNIST dataset

This project is divided into three axes of work:

- The attack section where we define how to run an attack on a given network, follow by an introduction to some of most common attacks. Next, we present a comparison on the effectivness of the attacks implemented during this project. Finally, we discuss models robustness and their implications in today's deep learning era.
- The defense section where we introduce state-of-the-art techniques along with their respectives strenghts and weaknesses. We also present in greater details two cutting edge defense strategies which have shown to be particularily efficient.
- In the last section, we try to dig deeper into the underlying causes that make deep learning models so vulnerable to adversarials.

5 Taxonomy of adversarial attacks

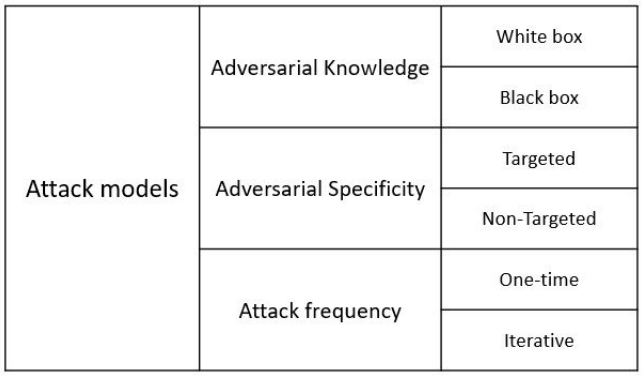


Figure 2: Taxonomy of attacks

5.1 Adversarial Knowledge

5.1.1 White box

White box attacks assume that the opponent has perfect knowledge of the target network architecture. Thus, the hyper-parameters, the number of layers, the activation functions, the weights and gradients of the model are known to adversary. It is then pretty straight forward for the adversary to set up an attack with a crafted pertubation specifically designed using the gradients'knowledge to fool the target model.

5.1.2 Black box

With "Black Box" attack, the opponent has limited information on the architecture of the target network. In fact, only the input images of the network and the predictions associated with the output are known. All parameters of the target network are therefore unknown to the adversary. Under these conditions, the adversary will define a local "substitute" or distilled network which aims to replicate as much as possible the target network. Once the locally constituted network is deemed to be close enough to the target network, the adversary can then (indirectly) have the parameters necessary for the proper functioning of his attack. For this, it will rely on the distilled network parameters as proxy for the actual target network.

5.2 Adversarial Specificity

5.2.1 Targeted attack

The purpose of targeted attacks is to influence the prediction of the target model to a class predefined prior to the attack taking place. For example, the adversary will be interested in influencing the network so that an input image corresponding to the label 3 is classified as a class 5 by the target network.

5.2.2 Non-targeted attack

The particuliarity of non-targeted attacks (also called universal attacks) is simply to deceive the network by forcing the latter to predict a class that is different from ground truth (misclassification). We want the network to classifies an adversarial input in a different class than that of the clean input image. We therefore seek to maximize the cost function related to a correct prediction of the ground truth label.

5.3 Attack Frequency

5.3.1 One-time

"One time" attacks to optimize the adversarial example.

5.3.2 Iteratives

These attacks require multiple steps to optimise the adversarial example.

6 Adversarial attacks (Gradient based)

In this section, our intend is not so much to provide an exhaustive description of all the gradient-based attacks we have implemented during this project but rather to give readers insights around the intricaties that makes neural nets particularily sensitive to adversarial attacks.

There are plenty of dedicated litteratures describing in great details the nuts and bolts of the many existing gradient-based attacks (some of which could be find in the reference section) and we felt, for the sake of conciveness, we would only present attacks related subjects that helped us carried out this research report. Overall, we have implemented 9 differents state-of-the-art attacks on MNIST dataset.

Fast Gradient Sign Method (FGSM)

In 2014, [Goodfellow et al.] introduced the Fast Gradient Sign Method (FGSM) as a mean to attack at deep neural network and subsequently as a regularization technique. FGSM has a solid theorical ground and has been used by many deep learning practitioners for years now. The attack is relatively straight forward and many subsequent gradient-attacks have been built upon the initial FGSM structure.

The FGSM attack works as follow:

$$x' = x + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y) \tag{1}$$

Intuively, the attack consist of calculating the model's gradient $J(\theta, x, y)$ w.r.t to x. We then take the gradient *sign*, multiply it by a scalar ϵ, which represents the magnitude of the pertubation, and add this resulting tensor back to our original input image x. Here we are, we have just crafted an adversarial examples using the gradient of our model. This is the essence of the FGSM attack but also that of the many other gradient based attacks. An illustration of the FGSM attack is presented in figure 3 below.

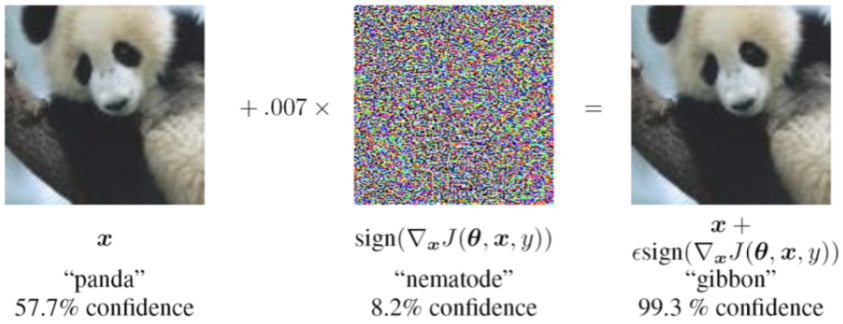


Figure 3: representation of an FGSM attack

Basic Iterative Method (BIM)

Basic Iterative Method is an iterative variant of the FGSM attack and can be defined as below:

$$x_m = clip_\epsilon(x_{m-1} + \frac{\epsilon}{i} \cdot sign(\nabla_{x_{m-1}}(J(x_{m-1}, y))) \tag{2}$$

Projected Gradient Descent (PGD)

Projected Gradient Descent is similar to BIM, and starts from a random position in the clean image neighborhood $U(x, \epsilon)$. This method applies FGSM for m iterations with a step size of γ as:

$$x_m = x_{m-1} + \gamma \cdot sign(\nabla_{x_{m-1}}(J(x_{m-1}, y))) \tag{3}$$

We then clip x_m so as to project it the same space as the original input.

$$x_m = clip(x_m, x_m - \epsilon, x_m + \epsilon) \tag{4}$$

Deepfool attack

Next, we introduce another interesting, slightly more sophisticated attack which not only uses the gradient method but also caracterizes the attack as an optimisation problem under constraints. This is the Deepfool attack and was initially introduced by Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi and Pascal Frossard in July 2016.

The Deepfool attack is a powerful method to calculate an optimal adversarial perturbation. Within a multiclass framework and given an input image x, the attack will first define the closest decision boundary within the outcome space representation with respect to $f(x)$. The attack will then consist of taking gradient steps in the direction that is orthogonal to the closest decision boundary until it eventually crosses it.

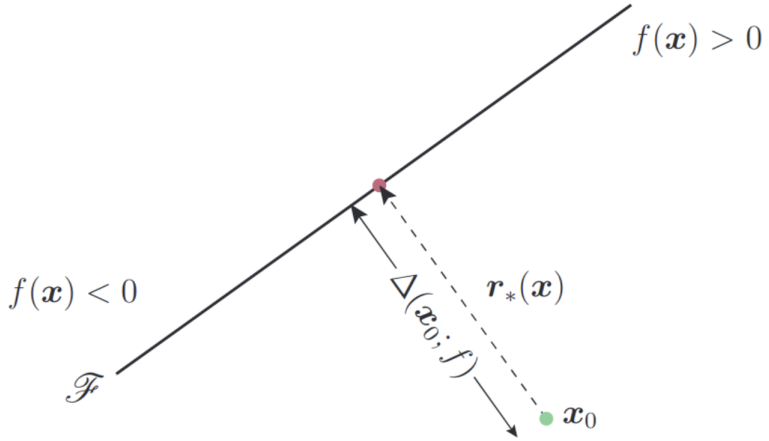


Figure 4: Adversarial examples for a linear binary classifier.

Figure 4 illustrate the deepfool attack for a binary classifier. Here, we want x_0 to be classified in the $f(x) < 0$ output space. We are given the gradient $\nabla(x_0; f)$ and we want to find $\boldsymbol{r_*(x)}$ which is the direction orthogonal to the decision boundary ($f(x) = 0$). Our goal is therefore to move in this direction until our classifier classifies x as negative ($f(x) < 0$).

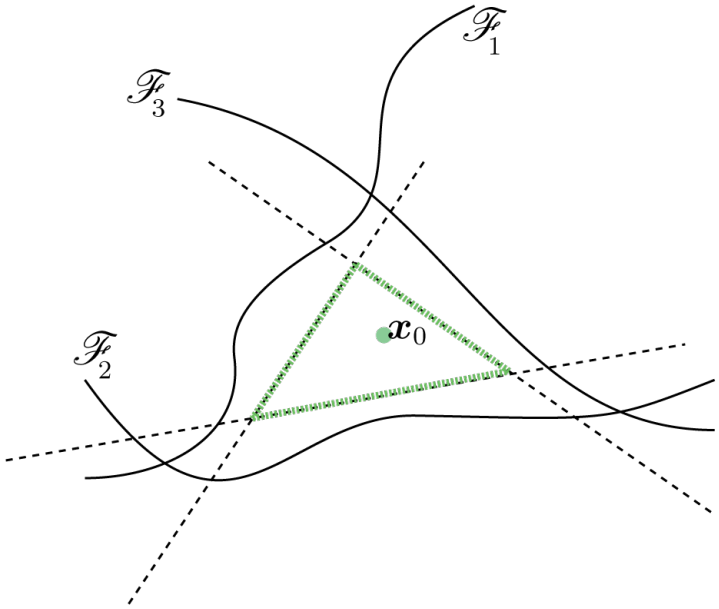


Figure 5: The boundaries of the polyhedron.

Figure 5 illustrate a multiclass decision boundaries. Decision boundaries are represented by F where F_1, F_2, F_3 are labels 1, 2 and 3 of MNIST dataset. x_0 is positionned within a polyhedron representing the output space of class 4. Because the true decision boundaries are non-linear we use a polyhedron to approximate them as indicated by the green triangle in figure 5.

The attack will now consist of finding the direction orthogonal to the closest of the three decision boundaries. It turns out F_3 is the closest. We will take a gradient step in that direction until we eventually cross F_3. At that point, our model will classify x_0 as class 3 leading to a successful attack.

The equation for $\hat{l}(x_0)$ below is designed to find the closest boundary w.r.t x_0 within a multi class setting.

$$\hat{l}(\boldsymbol{x}_0) = \underset{k \neq \hat{k}(\boldsymbol{x}_0)}{\arg\min} \frac{\left| f_k(\boldsymbol{x}_0) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_0) \right|}{\|\boldsymbol{w}_k - \boldsymbol{w}_{\hat{k}(\boldsymbol{x}_0)}\|_2}.$$

Once the closest decision boundary is known we will calculate the optimal pertubation to fool the classifier with $\boldsymbol{r_*(x_0)}$ defined as:

$$\boldsymbol{r}_*(\boldsymbol{x}_0) = \frac{\left|f_{\hat{l}(\boldsymbol{x}_0)}(\boldsymbol{x}_0) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_0)\right|}{\|\boldsymbol{w}_{\hat{l}(\boldsymbol{x}_0)} - \boldsymbol{w}_{\hat{k}(\boldsymbol{x}_0)}\|_2^2}(\boldsymbol{w}_{\hat{l}(\boldsymbol{x}_0)} - \boldsymbol{w}_{\hat{k}(\boldsymbol{x}_0)}).$$

For reference, the Deepfool attack algorithm will proceed as follow:

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 6: Deepfool algorithm for multiclass

Carlini-Wagner attack (C&A)

Over the course of this project we managed to implement a pretty effective defense strategy which is based on the statistical detection of adversarial examples in the logits layer. This defense algorithm is introduced in the subsequent defence section of this report called "Adversarial logits detection and correction".

Figure 7 below displays the effectiveness of this defense strategy against one of the best attack, namely the NewtonFool attack. The x-axis represents the L2 distance (basically the square norm of the difference between the

adversarial image and its clean equivalent) and y-axis is the model accuracy. The blue dot line represent the classifier with the logits defense while the red dot one is the same classifier but with no defense.

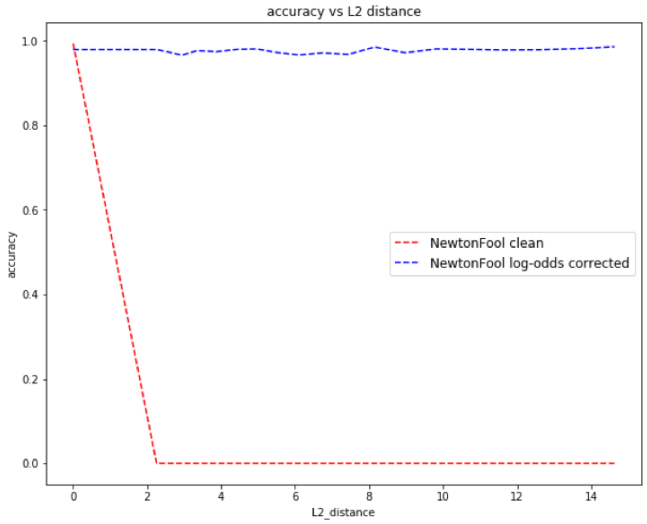


Figure 7: Logits detection and correction defense on NewtonFool attack

The reason why we want to point that logits defense strategy out is to introduce an attack that specifically aims at manipulating the logits score layer and thus counteract any logits-based defense. This is the Carlini Wagner attack.

The main objectives of the Carlini Wagner attack is presented below:

$$\begin{gathered} \text{minimize } D(x, x+\nabla) \\ \text{such that } C(x+\nabla) = t \\ \text{with } x+\nabla \in [0,1] \end{gathered}$$

Where D is a distance vector, x is the original input, ∇ is the pertubation, C is our classifier and t is a target label which we want to be different from ground truth.

The objective function $C(x+\nabla) = t$ is highly non-linear and we want to express it in a different form that is better suited for optimization. So Carlini and Wagner came up with the following optimization function:

$$f(x) = (\underset{i \neq t}{max}(Z(x)_i) - Z(x)_t)^+ \tag{5}$$

Where Z represent the logit score for each class and t is our target label (different from ground truth).
Finally, integrating equation 5 as the objective function we have:

$$\begin{gathered} \text{minimize } \|\nabla\| + c \cdot f(x + \nabla) \\ \text{such that } x + \nabla \in [0, 1] \end{gathered}$$

Where c is a constant scalar use to control the trade-off between the magnitude of the pertubation $\|\nabla\|$ and the likelihood of the attack being successful.

As a result, the optimization function of the Carlini Wagner attack directly manipulates the logits scores as a way to fool the classifier.

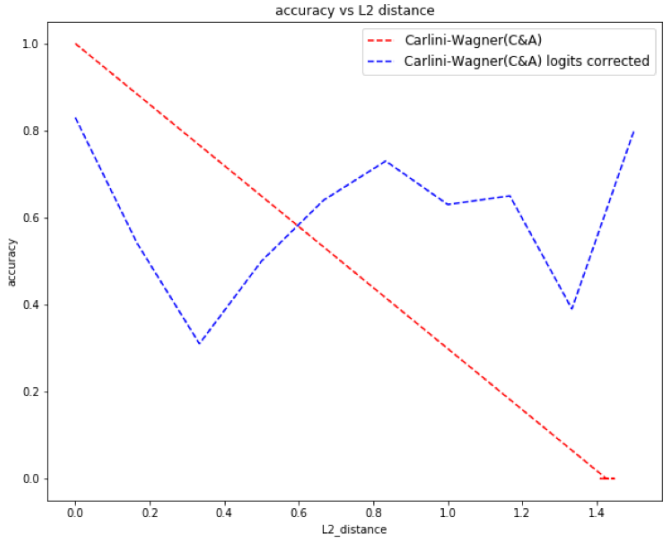


Figure 8: Logits detection and correction defense on Carlini Wagner (CA) attack

Figure 8 above shows the curve of the Carlini Wagner attack while we defended our model with the adversarial logits detection and correction technique we discussed previously. The red dot line is the C&A attack with no

defense while the blue dot line is the same attack but with the defense. It is pretty clear that the logits defense is not working anymore when attacked by C&A. The reason is that because the logits of the models are under the influence of C&A attack (by equation 5), the defense is now unable to properly detect adversarial examples. This can be seen by the very high variance of the C&A with defense curve which highlights the unreliabity of logits detection techniques. Also note that the attack with no defense brings the classifier's accuracy down to 0 with an L2 distance of only 1.4 making the C&A attack the most effective attack we have came across during this project.

7 Attacks Summary

7.1 One model versus multiple attacks

In this section, we will review the caracteristics associated with the gradient-based attacks covered in the previous subsections.

At the onset of this project, we have decided to use accuracy versus L2 distance as the most appropriate metric to assess attack's effectivess. For a given L2 distance we'd like to see the extend with which a particular attack could fool our classifier. That is, bringing the classifier's accuracy to its lowest level. That's exactly what Figure 10 intend to represent. The x-axis represent the L2 distance and the y-axis is the model's accuracy.

How did we obtain these curves ?

Each attack implemented during this project has an hyperparameter which directly or indireclty influence the degree of pertubation added to a clean input image. For example, FGSM has an explicit scalar value called epsilon ϵ which controls the magnitude of the pertubation. We then proceeded by incrementing the espilon value for each FGSM attack and obtained the dot blue curve shown in Figure 10. We have done likewise for all the attacks mentionned so as to compare them on a consistent basis.

Here are some of the key highlights:

- Iteratives attacks perform best. These attacks move with relatively small step size in the direction of the gradient. At each iteration the attack is subtlety moving closer to the decision boundary until it eventually crosses it and makes the classifer's prediction different from ground truth.

- NewtonFool is one the best performing attack while also being one of the fastest. It only require a pertubation with an L2 distance of 2 to completely fool the classifier.

- Jacobian Saliency Map (JSMA) performed moderately well as it requires a large L2 distance of about 9 to bring the classifier's accuracy

down to 0. JSMA has also the longest execution time among all of the attacks with approx 2h for 500 input images.

- Momentum iterative attack seems to exhibit a similar curve shape as JSMA and thus performed likewise. The main difference between the two lies in the execution speed where Momentum iterative is faster by a factor of 10.

- FGSM is the worst attack among all others. It is nonetheless, one of the fastest with approx 5 min execution time for 1000 input images.

- Adv-GAN attack curve seems to be off the track compared to other attacks. The reason is that we generated pertubations specifically designed to attack a different model than the one used for the other attacks. That is, we ran a black box attack for which gradients of our original model were unknown to us. We therefore attacked a "distilled" model that was supposedly replicating our original model in the hope that we could approximate our original model's paramaters and find the right direction for our crafted pertubations. To improve the Adv-GAN attack effectivness we would need to fine tune our distilled model so as to make it as close as possible to our original model. This however was behond the scope of this project and we decided to present the attack as is.

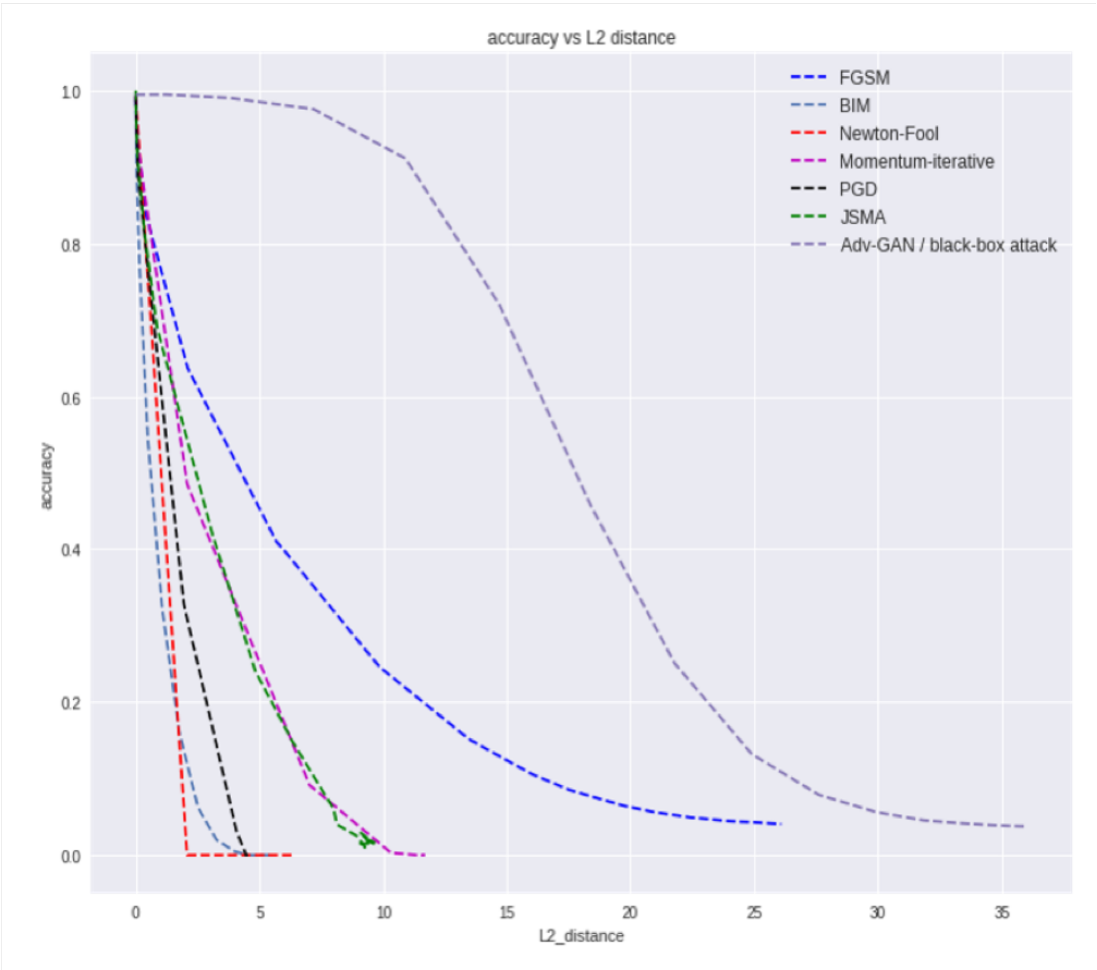


Figure 10: Attack effectiveness comparison.

8 Models robustness

8.1 One attack versus multiple models

In this section, we will look at the model's robustness from another perspective. Previously we studied the effectivness of several attacks on one model (our custom base model) and now we will focus on the impact that one attack has on multiple deep learning models. Figure 11 represents the robustness curve of six different deep learning models when attacked by Projected Gradient Descent (PGD). The axis are the same as Figure 10. We chose the PGD attack for its effectivness and execution speed.

There are important elements to note from Figure 11

- EffecientNet-B3 and MobileNetV2 are the most vulnerable models. It is evident from their respective sharpe downward curves that the PGD attack is doing a lot of damage to these two classifiers for a very little L2 distance. Interestingly, these models are also ones with the top 1 percent accuracy on Imagenet dataset. We posit that this excess of vulnerability must come from the dephtwise convolution layers which are key components of these two model's architecture. EfficientNet is a relatively recent architecture (June 2019) and thus finding out whether depthwise convolution layer has anything to do with adversarial vulnerability is a very new open question. We haven't had time to investigate this subject matter and leave that to further research.

- Densenet121 appears to be the most robust model among all others. One could hypothetise from a network architecture stand point that this could be due to the concatenation of dense blocks layers as well as the skip connections which help improve model robustness. Again, this is just assumption and this needs to be further investigated.

- Our custom base model (solid grey line) has only 4 convolution layers and 594,922 parameters yet it has relatively high accuracy on MNIST and seems to offer a moderate level of robustness.

These deep learning models offer various degrees of robustness some which could be more easily fooled by adversarial examples than others. However the key point to remember is that none of these models are fully bullet proof. Yet they are all used on a daily basis by a lot of industries ranging from biometric identification, facial detection to automous driving. In a near future, it is expected that these deep learning pre-trained models will become increasingly prevalent in many computer vision tasks. At that point, adversarial examples could become a major threat and could potentially lead to severe consequences for many industries.

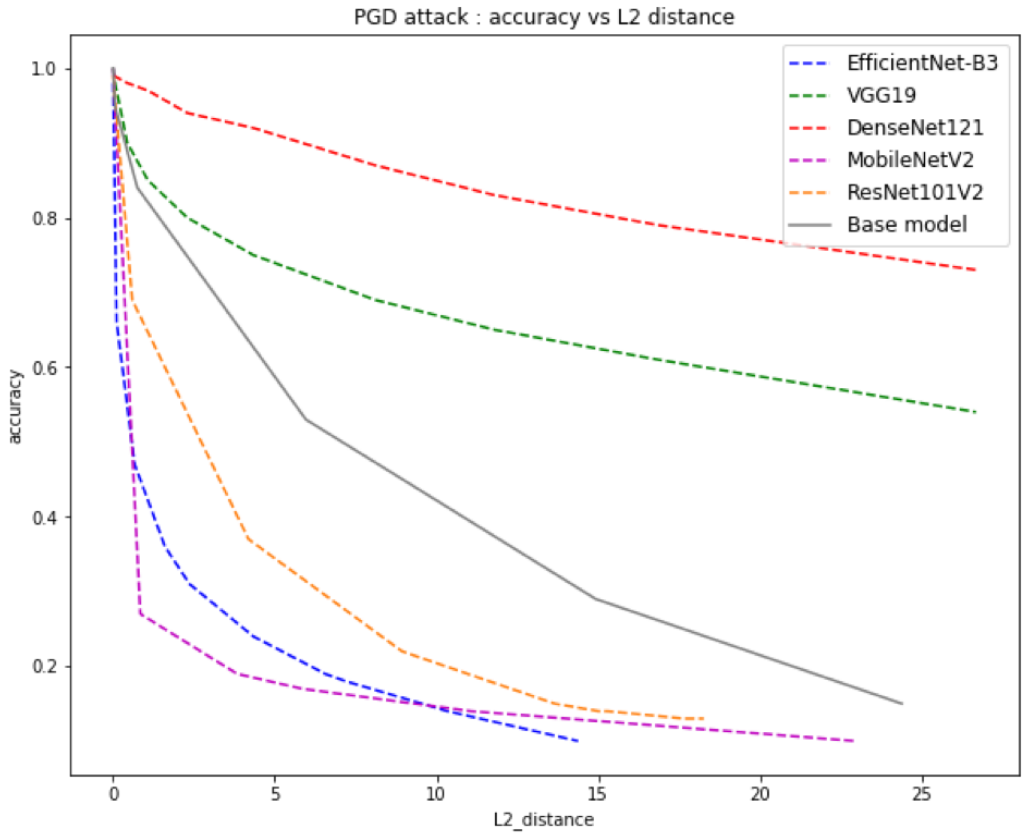


Figure 11: Models robustness comparison on Projected Gradient Descent attack (PGD).

9 Defense

Defense strategy is wide subject among academic literatures. As mentionned in introduction, if a slight perturbation added to a benign input drastically changes the deep network's output with a high-confidence, it reflects that our current models are not distinctively learning the fundamental visual concepts. Therefore, the design of robust deep networks goes a long way towards developing reliable and trustworthy artificial intelligence systems. To mitigate adversarial attacks, various defense methods have recently been proposed. These can be broadly classified into two categories: (a) Reactive defenses that modify the inputs during testing time, using image transformations to counter the effect of adversarial perturbation (i.e. Logits defense presented in subsequent section) and (b) Proactive defenses that alter the underlying architecture or learning procedure e.g. by adding more layers, ensemble/adversarial training or changing the loss/activation functions. Proactive defenses are generally more valued, as they pro- vide relatively better robustness against white-box attacks. Nevertheless, both proactive and reactive defenses are easily circumvented by the iterative white-box adversaries.

9.1 Auto-encoder denoiser

The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal "noise". Along with the reduction side, a reconstructing side is learnt, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input, hence its name. The autoencoder is similar to PCA, except that it exploits the non-linearity of the input. Within the adversarial defense context, the autoencoder has denoising function whereby the pertubated image will go through a reduced latent space (encoder), then the decoder will reconstruct this latent representation called the bottleneck by projecting it back into the same dimensions of that of the input space. This process is illustrated in figure 12 below. Autoencoder are supposed to remove the adversarial pertubation (denoising) perpetrated by an attack and therefore are used to reconstruct the original clean image. The denoised image is then given to the classifier which would supposedly, be able to classifies it properly. From our experiment it happen sometimes that the denoiser removes too much information and the classifier's score decreases not because of the adversarials per se but because of

the inappropriate denoising process.

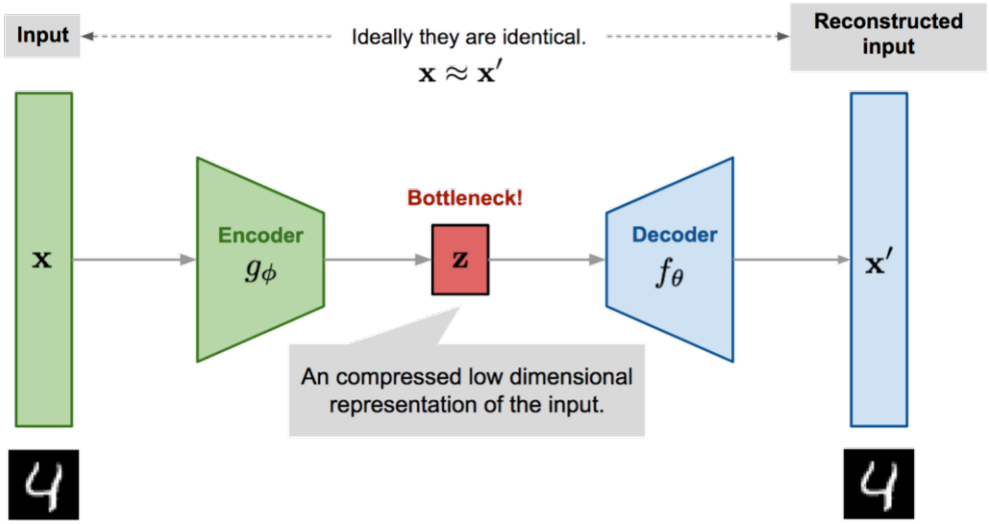


Figure 12: Auto-encoder denoiser architecture.

9.2 Adversarial logits detection and correction

Adversarial logits detection and correction is a universal defense strategy. This procedure require to defend a machine learning model against adversarial attacks by detecting whether or not the input has been perturbed, either by detecting characteristic regularities in the adversarial perturbations themselves or in the network activations they induce (Roth, Kilcher and Hofmann, 2019). Notably, Grosse et al. (2017) argue that adversarial examples are not drawn from the same distribution as the natural data and can thus be detected using statistical tests. Our work has been largely inspired by the research paper "The Odds are Odd: A Statistical Test for Detecting Adversarial Examples" by Kevin Roth, Yannic Kilcher and Thomas Hofmann (also referenced at this end of this report).
To implement this defense we will focus exclusively on the logit layer of scores as this gives us access to continuous values. Therefore we will refers to $f_y(x)$ as the y output from the logit layer of our target model when given an input image x. Note that typically $F(x) = argmax_y f_y(x)$.

Implicitely what we are implying is that the statistic within the logit layer will behave differently based on whether we gave a clean input image or an adversarial image to our classifier. Because of the variety of potential attacks we want to somehow standartized this logit statistic behavior by performing a Z-score.

Additionally, the log-odds may behave differently for different class pairs, as they reflect class confusion probabilities that are task-specific and that cannot be anticipated a priori. This could also be adress by performing a Z-score standardization across data points x and perturbations ∇. For each fixed class pair (y, z) we will define:

Logits differences for clean examples:

$$\underset{y \neq z}{f_{y,z}}(x) = f_z(x) - \overbrace{f_y(x)}^{max\ logit} \qquad with\ y = y^* = ground\ truth\ label \tag{6}$$

Note that equation 6 should always gives negative result. Intuitively

$f_y(x)$ with y corresponding to the ground true label should have the max score within the logits score layer.

Now, let's take a look at the same equation but with an adversarial input image $(x + \nabla)$ instead of clean input x.

Logits differences for adversarial examples:

$$\underset{y \neq z}{f_{y,z}}(x + \nabla) = f_z(x + \nabla) - \overbrace{f_y(x + \nabla)}^{not\ max\ logit} \qquad with\ y = y^* = ground\ truth\ label \tag{7}$$

In equation 7 notice how the two components of the equation will push the result toward positive territory. $f_y(x+\nabla)$ should have relatively low value because otherwise the attack would be worthless. In fact, this is precisely what an effective attack should do - pushing the logit score of the ground truth label toward the minimum logit score value.

It is important to notice that during test time the ground truth is unknown and thus we could only rely on our related statistic logits detection procedure to define whether we are dealing with a clean example x or an adversarial example $x + \nabla$.

Based on what we have defined previously we will now implement a Z-score standardization on $f_{y,z}(x)$ such that:

$$\underset{y \neq z}{g_{y,z}}(x, \nabla) = zscore(f_{y,z}(x + \nabla) - f_{y,z}(x)) \tag{8}$$

Following the recommendation presented in the research paper mentionned earlier we define equation 8 as the key metric for our detection algorithm. In this case whenener $\nabla \neq 0$ we would expect $g_{y,z}(x, \nabla)$ to exhibit differents statisticals properties than when $\nabla = 0$.

Finally our detection algorithm will take the form of equation 9 below,

$$\max_{y \neq z} \{g_{y,z}(x) - \tau_{y,z}\} \geq 0 \tag{9}$$

In equation 9 we define a threshold $\tau_{y,z}$ which guarantee a maximal false detection rate (of say 1 percent), yet maximize the true positive rate of identifying adversarial examples. Through trial and error we have find that $\tau_{y,z}$ at 1.28 gives the best detection rate with about 95 percent of adversarials example detected on MNIST dataset.

Now that we have defined a very effective detection mechanism we are interested in correcting these pertubated logits on the spot (at test time). To that end, we have decided to use an SVM classifier specifically trained on adversarials logits. We have thus generated adversarials logits using a Fast Gradient Sign Method (FGSM) attack and subsequentially gave these manipulated logits for training to our SVM classifier with the ground truth labels.

Then, at test time the prediction of the overall architecture will go as follow:

Whenever an input's logit statistics are flagged as adversarials by our detection algorithm, the prediction part will be handled by our SVM classifier (trained only on adversarials logits). Otherwise, the input will be flagged as clean example and prediction will be given by the regular softmax activation of our initial model.
Figure 13 provides a concise representation of that architecture.

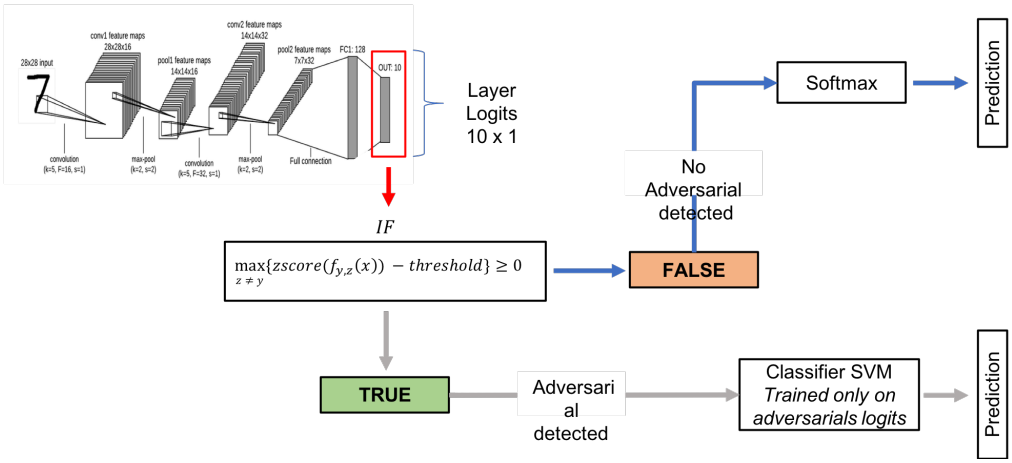


Figure 13: Adversarial logits detection and correction architecture.

The outcome of this defense strategy is pretty impressive. We have tested it multiple times on different attacks and we were surprise to see how our model accuracy stayed in relatively high terrority across the whole L2 distance spectrum. In fact, for most cases what we have observed is that the accuracy of the logits corrected model tend to decrease slightly before jumping back up whenever our detection algorithm is having hard time figuring out which example is an adversarial and which is a clean one. This happen precisely when the attack is subtile enough and the L2 distance is very small. At that point, the logits manipulation perpetrated by the attack is not obvious and the detection procedure fails to capture this adversarials example. As the L2 distance increase the detection rate improves considerably.

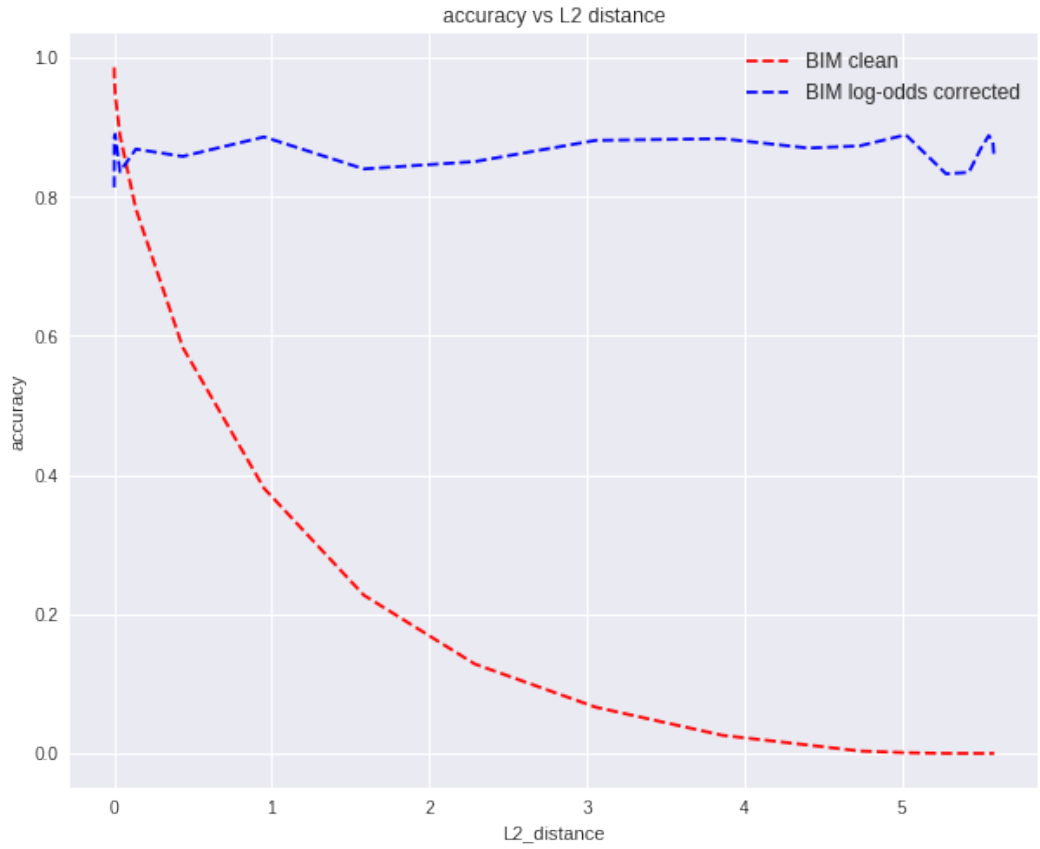


Figure 14: Logits defense on BIM attack.

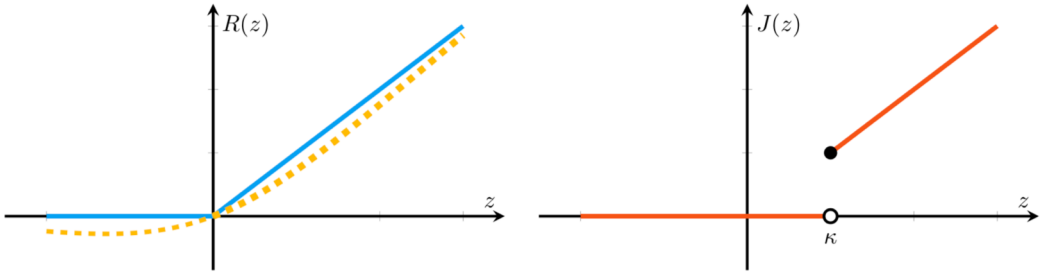
Figure 14 above shows how our logits detection and correction defense add robustness to our initial model when attacked by BIM. The accuracy of our corrected model (dot blue line) stayed within a 99.1 and 85.5 percent range across the whole L2 distance spectrum. Comparatively the same model without defense (dot red line) has an accuracy at 0 with an L2 distance of 5.

The defense provide similar robustness benefit against other attacks. The important aspect is that the detection and the correction are universal. That is, there is no need to redifined our detection algorithm for different attacks. Likewise, there is no need to retrain an SVM classifier on adversarial logits generated by other attacks.

Implicitely, it is assumed that every adversarial attacks will influence the logits in a way that is statitically detectable since their logits will behave differently than those from clean examples, regardless of the attack being perpetrated. That being said, we presented in a previous the Carlini Wagner attack which has the particularity of covering its own tracks when manipulating logits. This would render the defense technique completlety inefficient since adversarial logits would become undetectable.

9.3 JumpReLU defense strategy

In this section we introduce an activation function which aims at improving models robustness against adversarial examples. This new activation function relates to work done by N. Benjamin Erichson, Zhewei Yao,Michael W. Mahoneyis on their research paper "JumpReLU: A Retrofit Defense Strategy for Adversarial Attacks" published in April 7 2019.



(A) ReLU and Swish (dashed). (B) JumpReLU activation function.

Figure 15: JumpReLU activation function.

Figure 15: The rectified linear unit is the most widely studied activation function in context of adversarial attacks, illustrated in (a). In addition its smooth approximation (Swish) is shown, with = 0.5. The JumpReLU activation function (b) introduces robustness and an additional amount of sparsity, controlled via the jump value (threshold value) . In other words, JumpReLU suppresses small positive signals.

The JumpReLU activation function could then be defined as follow:

$$\mathrm{J(z)} = \begin{cases} 0 \; if \; z \leq k \\ z \; if \; z > k \end{cases}$$

This activation function introduces a jump discontinuity, yielding piecewise continuous functions. As a consequence small positive signals are ignored by the network. In practice, one has to decide which threshold value k to use for the model to have both high accuracy and significant robustness to adversarials. There is a trade-off between accuracy and robustness that needs to be balanced properly.

We replaced all the ReLU activation functions of our based model by JumpReLU functions as illustrated in figure 16 below.

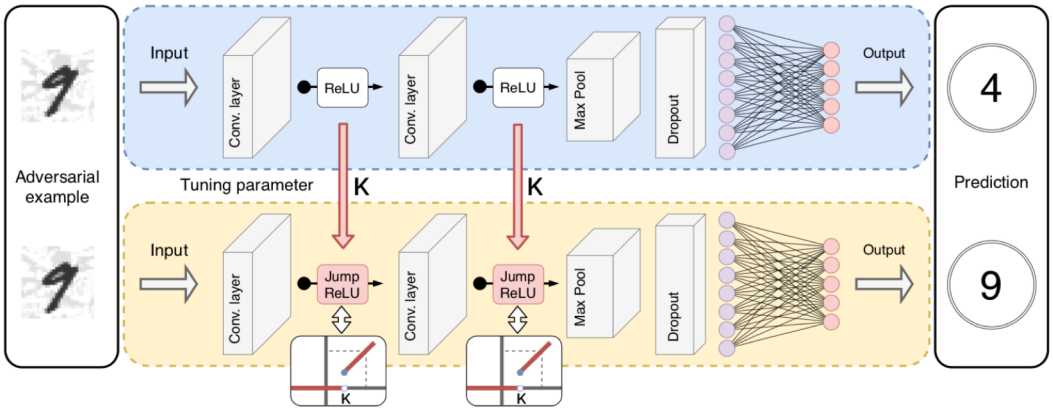


Figure 16: JumpReLU Network Architecturen.

We then chose to select k empirically by trial and error in order to find the threshold that maximise our model accuracy for given level of robustness. We ended up using $k = 0.0645$.

However, the research paper also suggest another technique to find k. This technique consist of using some randomness by letting k be a random scalar within a predefined range. The theorical assumption behind this rational is that by introducing some randomness in the model we are in fact improving the robustness. Randomness as a resource to improve model robustness has been demonstrated before within the defense literature. More concretely, the randomized JumpReLU selects a random in a specified range for every forward pass. The underlying idea is that this approach leads to obfuscated gradients.

Figures 17 and 18 display the model accuracy benefit brings about by the jumpReLU defense strategy (dot line) vs the accuracy of our classifier with no defense (solid line).

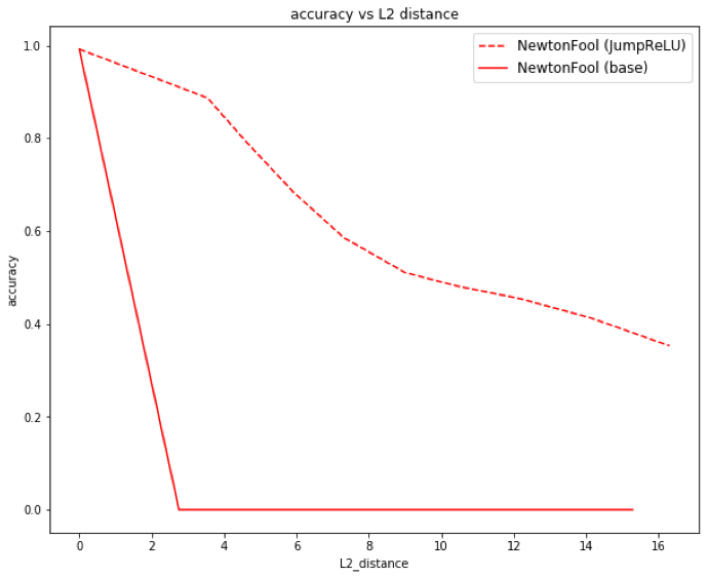


Figure 17: JumpReLU defense vs NewtonFool attack.

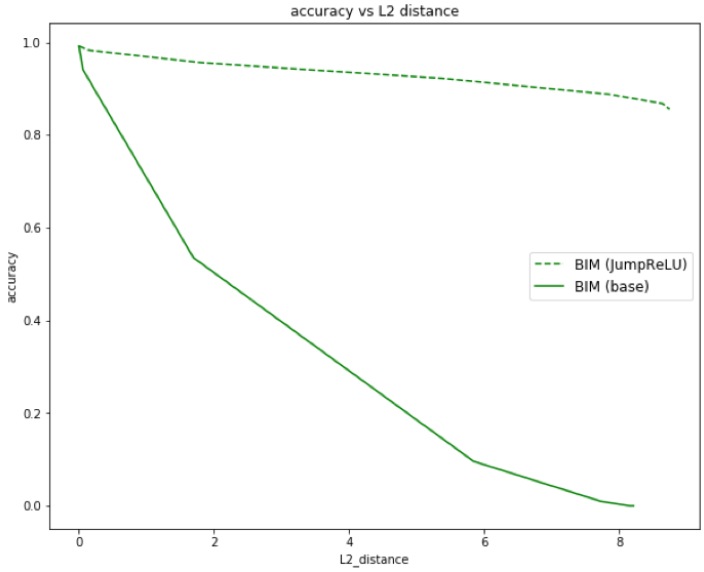


Figure 18: JumpReLU defense vs BIM attack.

From Figures 15 and 16 it is pretty clear that the JumpReLU add robustness to our model. It is even more pronounced with the BIM attack where the model's accuracy stayed within a narrow range close to 95 percent.

We have tried other attacks and obtained similar results. Our experimental results show that this simple and inexpensive strategy improves the resilience to adversarial attacks of previously-trained networks. The benefit of the defence lies in its simplicity. It is backed by sound theorical explanations and could be very easily implemented to any models pre-trained or not. Besides, the random selection of the k threshold is still a widely open academic research subject.

10 Study of the weaknesses of the classifiers

The goal of this section is to propose an explanation on why adversarial examples exist. Adversarial examples not only on neural networks but on any classifier.

When we talk about adversarial examples, it is directly associated with neural networks because the existence of the former was releaved when processed by the latter. That's exactly what Ian Goodfellow find out when he first introduced adversarial examples. Of course adversarial examples also exist for other classifiers. Goodfellow was surprised to discover that only a minimal perturbation was necessary to make the neural network changes its class prediction. The car was still a car but it was now classified as a plane by the model. This is a misclassification. Because even if the neural network miclassifies it, human brain is not fooled as it can still tells the difference between a pitcure of a car and the picture of a plane.

10.1 Understanding the problem

Classifiers work well and better with big data because they can capture nearly the entire input distribution. If a test input come from the same distribution as the training input, there is a high probability for the new prediction output to lie inside the existing data manifold. If the image space was fully supervised, every points had a label, obviously there will be no adversarial attack possible. Now this will never be the case but it can help to understand why adversarials exist.

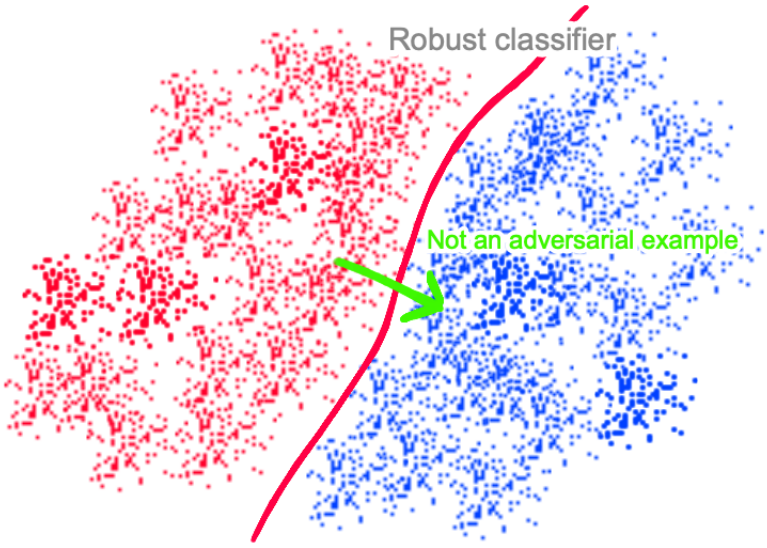


Figure 19: A perfect decision boundary for a binary class problem

The problem can be illustrate by a 2D example, for a binary class problem. There are 2 cases: If the two classes manifolds are very close to each other, the decision boundary is easily defined. Points from different classes are very close. You just have to give enough complexity to your neural network to draw a decision boundary which will classify perfectly the existing points. In this case, no adversarial attacks will succeed. Because if a point cross the decision boundary, it will automatically be close to points of the new class, so it has to be similar. To understand it, let's take an theorical example with MNIST, if the 3 and 7 class manifolds were very close to each other, and every points of the two classes were defined by a clearly defined decision boundary, the attack from misclassifying a 3 to a 7 will result in a point very close to other 7 points. It will thus looks like a 7 and therefore will not be an attack because the classifier will rightfully classifies it as a 7.
That is, there will be no space between the two manifolds for an attack to be perpretated. In essence, we posit that adversarials attacks exist because they can exploit some unoccupied regions of space that lie between class manifolds. By doing so, adversarials force the network to re-evaluate its decision boundary and make it wrongly consider than what was previously a vacant space is now

an extended class manifold. Figure 19 above illustrate this theorical perfect decision boundary where no space is available for an adversarial attack to take place.

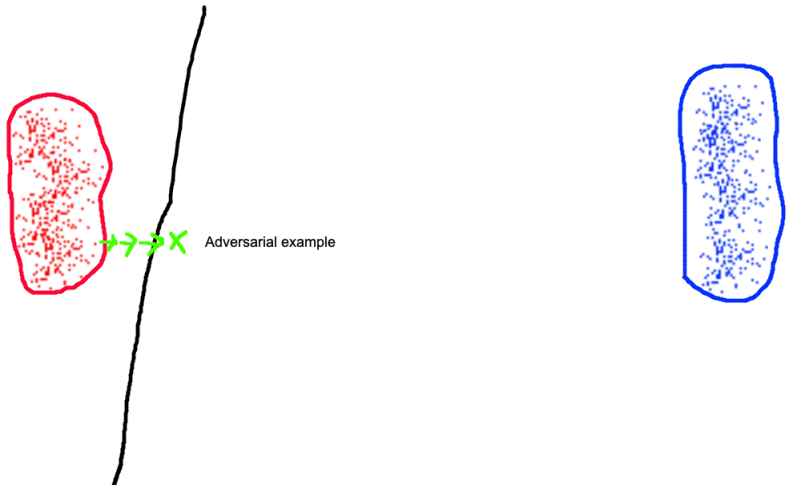


Figure 20: Decision boundary in practice for a binary class problem

Now, let's take a realistic problem, with class manifolds (very) far from each other as illustrated in figure 20 above. The neural network will just draw an arbitrary decision boundary, among many ones separating the two classes. There are potentially infinite unlabeled points between the 2 manifolds and we don't know what they are or what their labels are. If the decision boundary is very close to the class 3 manifold it will be easy to attack and force the network to misclassifies a 3 to a 7. Taking the gradient of the loss will give you the direction to approach the decision boundary. Crossing this boundary, the classifier will think it is a 7 yet the point will still be closer to the class 3 manifold. It will indeed still look like a 3 to human eyes. This is what we call an adversarial example.

10.1.1 Adversarial training

Continuing with our example of class 3 and 7; The real problem here is that the supervision did not capture the whole class 3 distribution. "The whole

class 3 distribution" means all (infinite) images which would be classified as a 3 by a human. It includes images with different scales, translated, or rotated, blurry... Had we labeled all possible images looking like a 3 in the 28x28 image space then any attacks will be completely worthless because all possible existing 3 will be properly labeled as such. This is exactly what we try to do in adversarial training. It's a form of data augmentation. Remember, adversarial training consist of attacking a class, relabel the adversarial example with the right class, and retrain the classifier. By doing this we just guide the network toward drawing better decision boundaries. Of course we cannot label the whole class 3 distribution in practice.

10.2 The loss function

The first part was explaining why adversarial examples exist, now the focus is on the weaknesses of the classifiers we know. The objective function is very important as in reinforcement learning. The loss function in classifiers (reward function in RL) is a very important function, because it defines what the classifier tries to do, what is its objective. If the loss function is not well defined, the result will not be satisfying.

For classification we often use the binary/categorical cross entropy which is great to achieve one goal: classify well the data. But the way we implement it does not take into account how well the classifier has to design the decision boundary.

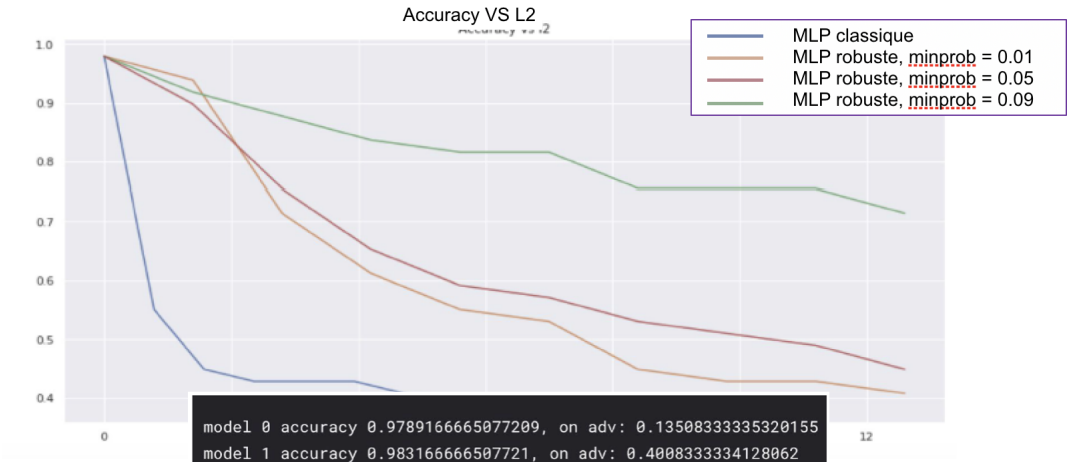
$$\mathcal{L}(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(p_i) + (1-y_i)\log(1-p_i)\right] = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(p_{ij}) \tag{10}$$

The loss function understands well that it has to increase the probability of the target class by penalizing the gap between the predicted vector and the output vector for the target class. In that sense, this loss function is good to classifies well data but not so much in drawing efficient decison boundaries.

11 Label embedding

The problem is that the target vector is only zero except for one class, the ground truth label. Therefore the target one-hot-encoded vector representation leads to 2 problems: In the logit layer (prior to softmax) the neural network just has to output a big number to get the target class probability to one, because of the softmax/sigmoid function asymptotic properties. In our view, the loss function is intrincically biased toward the target class only because it fail to consider the relative proximity of the target class label to that of other classes. That is, the loss function only consider the target label due to the sparcity of the target one-hot vector with 1 for ground truth and 0 otherwise. For exemple, in MNIST, if an 8 looks more like a 0 than any other classes, then I want my model to integrate this similary and thus I would adjust the target representation for this class. Which means I won't be using a one-hot-encoded vector but a vector that attribute small probability to other classes as well and not be zeros. In this example, I'm interested in having target probabilities that reflect the similarity between a 0 and a 8. For Imagenet, if I have an image of a cat I want the probability of this image being a dog higher than that of being a plane for example. And this for any cat. This is related to semantic in word embedding. The target vector has to understand this semantic which is not the case with the one hot encoded vectors. This could also be referred to as labels embedding. This form of label representation, we are convinced, could helps drastically increase deep neural networks robustness. Labels would then be represented as semantic vectors and would eventually allow neural networks to model class similarities into their predictions.

We tested this idea with simple dense network on MNIST to see what it was like. We changed the one hot encoded vector to (0.91,0.01,0.01,0.01,0.01, 0.01,0.01,0.01,0.01,0.01) for the class 0. We used a classical softmax as the last activation function. This tiny change increase the robustness, with accuracy on an FGSM attack jumping back up from 0.04 to 0.61 for the same L2 distance. The blue line is the reference model, trained with one hot encoded labels.




```
model 0 accuracy 0.9789166665077209, on adv: 0.13508333335320155
model 1 accuracy 0.983166666507721, on adv: 0.4008333334128062
model 2 accuracy 0.9806666665077209, on adv: 0.6106666668256124
model 3 accuracy 0.9810833331743876, on adv: 0.7242500001589457
```

12 References

[1] Towards Evaluating the Robustness of Neural Networks, Nicholas Carlini, David Wagner.

[2] JumpReLU: A Retrofit Defense Strategy for Adversarial Attacks, N. Benjamin Erichson, Zhewei Yao, Michael W. Mahoney.

[3] DeepFool: a simple and accurate method to fool deep neural networks, Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Pascal Frossard.

[4] The Odds are Odd: A Statistical Test for Detecting Adversarial Examples, Kevin Roth, Yannic Kilcher, Thomas Hofmann.

[5] Towards Deep Learning Models Resistant to Adversarial Attacks, Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu.

[6] Generating Adversarial Examples with Adversarial Networks,Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, Dawn Song

[7] Adversarial Defense by Restricting the Hidden Space of Deep Neural Networks, Aamir Mustafa, Salman Khan, Munawar Hayat, Roland Goecke, Jianbing Shen, Ling Shao