,

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

FAKULTÄT FÜR
ELEKTROTECHNIK,
INFORMATIK UND
MATHEMATIK

Institut für Elektrotechnik und Informationstechnik
Universität Paderborn
Fachgebiet Nachrichtentechnik
Prof. Dr.-Ing. Reinhold Häb-Umbach

Technical Report

Adversarial Attacks on Machine Learning Models

by

Nikhil Kumar Jha
Matr.-Nr.: 6823597

Supervisor: Jahn Heymann
Abgabetermin: March 13, 2019
Nummer: EIM/NT - 012010

Statutory Declaration

I hereby declare that I have done the present work on my own. The ideas and knowledge used directly or indirectly from external sources are identified as such in the citations. This work has not yet been submitted to any other examination authority and has not yet been published.

Paderborn, on _____ _____

(Datum) (Unterschrift)

Abstract

The machine learning model can be easily fooled to incorrectly classify an input sample which was structurally and intentionally modified. These perturbed samples created from the original data set by making the worst case changes are called adversarial examples, and this act of fooling the model is called adversarial attacks. This vulnerability of the machine learning models to force misclassification is a major security concern since such models are deployed at various locations for various tasks. So depending on the application, if proper measures are not taken to secure them, these models can easily be deluded by the attacker and the results can be critical.

This technical report is structured as follows. Section 1 provides a detailed introduction to this domain. Section 2 presents the various ways an adversarial example can be created. Section 3 describes the existing approaches or defense against such attacks. Section 4 explains the experiments performed to create adversarial examples and also creating a defense against them.

Contents

1 Introduction

We are very well aware of how powerful machine learning is since many of the tasks which were earlier not solvable are possible now with the help of the various machine learning algorithms. The recent advances in the past decade have seen the development of these machine learning techniques to solve problems in domains like image, video and text classification. Deep neural networks caught up to the human level performance in tasks like image and object recognition during the ImageNet competition. With such great performance, it is nearly an exception that a trained model could make a mistake on some classification task. This inspired the area of research where people started to study ways of breaking the machine learning model.

There was a prodigious discovery made by [Sze+13] in 2014, that most of the machine learning models are prone to attacks, even the ones which are deemed state of the art. An adversarial example in machine learning means making a slight, intentional change to the reasonable input data, which results in a machine learning model to behave in a way different than expected. In other terms, adversarial examples are used to fool the machine learning model, in order to make an incorrect prediction. The changes are in themselves so small that most of the times it gets undetected, even from the human eyesight and the altered input data can hardly be distinguished from its original counterpart. This exposed a blind spot in the way machine learning models are trained. The reason why exactly an adversarial example works is still receiving speculative explanations. The most popular hypotheses being the linear behavior of machine learning models, while others suggest its caused by over-fitting of the model.

The development of machine learning models is originally done for the benignant and static environment, where the distribution for training and test data is present from a stationary source. But just by changing the input data by a small amount in the "worst" direction, making it malicious and still indistinguishable from the original sample, will cause the model to give wrong predictions. This change is not just some random noise but carefully computed as a function of the network parameters, after a series of optimization iterations. Goodfellow [KGB16] also has shown that these examples will still remain misclassified in case of an image recognition task, for a physical world adversarial examples observed via a camera.

This realm of adversarial examples can be considered as a by-product of machine learning algorithms, carrying no practical relevance as such, but the fact that these examples make the machine learning models vulnerable to attacks is an interesting and important domain to explore, from a security point of view as well. For example, a classifier used to identify a product as malign or benign based on its features. But, if we tweak the features of the malign product to make it an adversary, the classifier will falsely classify it as a good product. Another very common example is putting up a picture over the stop road sign, which to human eyes will look like dirt or just a spot, but the autonomous vehicle might identify it as something else and can cause it to crash into some other vehicle. A security scanner at the airport which utilizes machine learning methods to identify between a weapon and other items can be fooled to identify a gun as a mobile phone. There are many such security concerns, which makes it interesting and necessary for more research in the open areas of creating a defense against these adversaries.

2 Creating adversarial examples

The existence of adversarial examples suggest that the model is not actually learning the true data distribution of the train samples, which is the main objective when we train the model to predict output label. This means the model will perform well on the data samples which are naturally occurring and are present in the train data set, but will misclassify the samples which have not so high probability in the data distribution [GSS14]. This is true even for the state of the art models, not just the deep learning models but also the linear models suffer from similar problems.

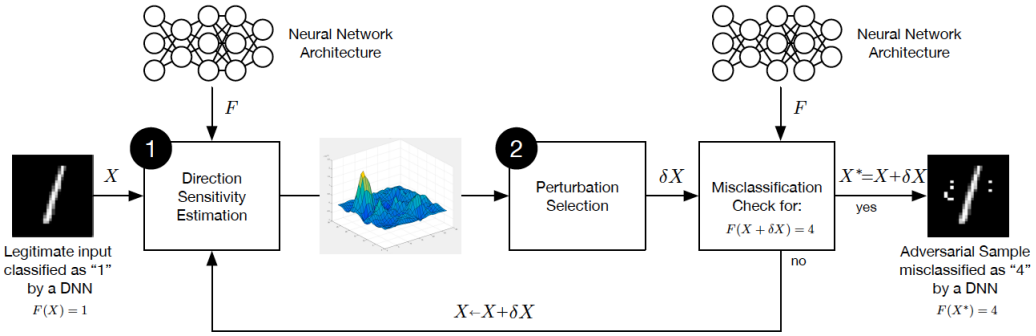

Figure 2.1: Adversarial example creation is a two-step process: 1) Estimating direction sensitivity 2) Perturbation selection

Adversarial attacks can be crafted in various ways [Pap+16b], and many such different methods have been published in the last few years, which we will discuss briefly in this section. But the underlying crafting process in most of the methods is composed on the two steps shown in the Figure 2.1 [Pap+16a].

2.1 Fast Gradient Sign method

Goodfellow[GSS14] showed that we can have a linear view of the adversarial examples. A infinitesimal change in the input data gets unnoticed by the model owing to the limited precision of the features. But with sufficient dimensionality, these small changes at the input add up and grow linearly with the number of dimensions, and add up to one large output . So with enough number of dimensions, a linear model will also have adversarial examples.

This linear view makes it easy to create the adversaries. Let x and y be the inputs and targets to and from the model, and θ represents the parameters of the model. Also J(θ, x, y) is the cost function used while training the model, which is linearized across the current value of the θ, and we get a constrained perturbation of η given by the formula,

$$\eta = \epsilon sign(\nabla J(\theta, x, y)) \tag{2.1}$$

where ϵ is small perturbation to not be detected by human eye but it has enough magnitude to change the classification decision by the classifier. So by adding this perturbation of η to the original data set, we get the adversarial samples. It was found that this technique could reliably cause wide variety of models to give incorrect predictions. There are also other methods based on similar underlying concept of using gradients. For example, adversarial examples can also be produced by rotating the x by a small angle in the direction of the gradient.

The activation functions in neural network makes the output of the layers piece-wise linear. Goodfellow[GSS14] hypothesized that these neural networks are just too linear to not be affected by adversarial examples created by the Fast Gradient Sign method (FGS) method. So, to corrupt the neural networks, these low cost computed adversaries should also be good enough.

2.2 L-BFGS method

Christian Szegedy et al.[Sze+13] were the first to discover the existence of adversarial examples, the samples which represent very low probability in the manifold which are hard to explore just by randomly sampling the input around a given example. For example, even the various augmentation techniques used before training a computer vision model won't be able to provide good robustness against these adversaries, since these augmentations are highly correlated to the original data distribution and are statistically inefficient to create a good defense.

Let $x \in [0, 1]$ be the input example and f be the classifier, mapping the inputs to a discrete label set l, where $l \in 1 \ldots$ k. Also, let r be the minimum distance such that $x + r$ is the closest example classified incorrectly as l such that $l \neq f(x)$. In short, the optimization problem to solve is:

$$Minimize \ \|r\|_2 \ s.\ t.\ f(x + r) = l \ and \ x + r \in [0, 1] \tag{2.2}$$

They[Sze+13] used the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm to solve this optimization problem. The calculated value of r might not be

unique, and also this value might be hard to find. The L-BFGS algorithm will however give an approximated value of r. So, of the different solutions we can get, we have to select the one where the classifier loss is also minimum when the the input is $x + r$ and the output label should be l. So we end up calculating the minimum value of c by doing a line search to solve the following optimization problem:

$$Minimize\ c\|r\| + loss_{classifier} f(x + r, l)\ s.\ t.\ x{+}r \in [0,\ 1] \tag{2.3}$$

The solution from this optimization problem will be the minimum value of r that we should add to the original image to make it an adversary. In case if the function above is actually a convex function, then we will not get the exact value but an approximation of the value r.

2.3 Carlini-Wagner method

Carlini et al. [CW17] introduced this method, which to some extent is inspired by the last method we discussed. Here also, the adversaries are calculated by looking at it as an optimization problem. So, the minimum distance which should be added to the original example is calculated, such that the resultant output is still in the original range but the classifier output is changed to some other label.

The distance metric here is instantiated as an L-norm of 3 type viz. L_0, L_2 and L_∞ norm. Further a function f is defined such that after adding the perturbation δ, only if the model misclassifies the perturbed example, the value of $f(x{+}\delta)$ will be less than zero. The objective function in this case becomes:

$$Minimize\ \|\delta\|_p + c * f(x{+}\delta)\ s.\ t.\ x{+}\delta \in [0, 1] \tag{2.4}$$

The p decides which norm is used and c is the trade-off constant, which is chosen by a binary search. Unlike in previous method, here we use Adam optimizer instead of L-BFGS to solve the box constraint. This experiments by Carlini et al.suggests that this attack is much more powerful than the ones discussed till this point, and L_2 being the one giving best rates in misclassification. Figure 2.2 gives an overview of how the original and adversarial examples look similar.

However, owing to the number of computations and the complexity of the method, this technique is quite slow and that turns out to be the trade-off against the powerful examples received. One way to speed it up a bit is to take only one value of c into consideration, instead of doing the binary search.

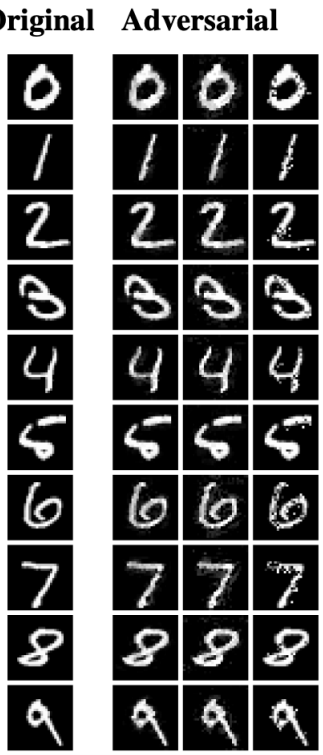

Figure 2.2: First column shows the original image while the other three columns show the adversarial images created using L_0, L_1 and L_∞ distance metric.

2.4 DeepFool

Moosavi-Dezfooli et al. [MDFF16] suggested a method to find the adversarial examples by iterative linearization of the constraints of the classifier, giving the corresponding hyper-planes which separate the various classes. Any small perturbation that makes the value of the example to cross this hyper-plane will result in misclassification by the model. The author first performed the experiments on a binary classifier and later generalized it for multi-class problems. This method, however, is only for non-targeted case, which means the network can be fooled to misclassify the example as some other class, but we don't have the control over the prediction of a particular class.

The optimization strategies used for the binary case can be seen either as the Newton's iterative algorithm or a gradient descent algorithm with an adaptive step size automatically chosen at each iteration. The algorithm is based on the greedy approach and that makes it to settle for non-optimal solutions. Still, the perturbations received as output are small enough to cause the least distortion in terms of euclidean distance, and causes the example to cross the nearest separating plane. The maths and equation for this method is out of scope for this technical report, so it is not mentioned here.

2.5 Saliency Map Approach

Papernot et al. [Pap+16c] presented this Jacobian-based Saliency Map Approach (JSMA), which is a deviation from the gradient based approaches discussed recently. Here the first step is to compute the Jacobian of the model, from which we get a matrix of values $\frac{\partial f_j}{\partial x_i}$, means the derivative of a particular class j with respect to a input feature i. Using this matrix, we create the saliency map in the following way:

$$S(x,l)[i] = \begin{cases} 0, & \text{if } \frac{\delta f_l(x)}{\delta x} < 0 \text{ or } \Sigma_{j \neq l} \frac{\delta f_j(x)}{\delta x_i} > 0 \\ \frac{\delta f_l(x)}{\delta x} * \Sigma_{j \neq l} \frac{\delta f_j(x)}{\delta x_i}, & \text{otherwise} \end{cases} \tag{2.5}$$

The idea is to perturb the features corresponding to high adversarial saliency values in the map, since these are the features which will affect the classification decision of the model to the most extent. So, if the adversary wants to assign the machine learning model a class l, then it selects the feature i which has the highest value in the saliency map and changes its value. This feature's value can either be increased or decreased, given by the two different strategies suggested in the paper to achieve misclassification.

Once the input feature which is to be perturbed has been identified by looking into the saliency map, it has to be perturbed in each iteration by some amount which is problem dependent. There is another parameter used called γ, which tells the maximum number of iterations and equivalently the maximum perturbance that can be done to a input sample. This is an important parameter in the sense that if the distortion is large enough, it is easily detectable by the humans, and adversary loses its purpose.

2.6 Elastic Net method

Elastic Net method (EAD) is proposed by Chen et al. [Che+18] to attack deep neural networks and can be seen as a generalization to the work done in Carlini-Wagner attack (CW) who took L_2 as distance metric. This method, as the name suggests, uses both L_1 and L_2 distance metric and controls the trade-off between the two using a β variable. The work is very much inspired from the CW, and the same loss function has been used here. For the value of $\beta = 0$, this becomes the CW due to presence of only the L_2 metric.

The previous work done by Carlini et al.[CW17] did not consider the L_1 distance metric, but only L_0 , L_2 and L_∞. Chen et al.[Che+18] leveraged the use of L_1 metric since it accounts for total variation and also encourages sparsity in the distortion. For a given example x_0 and its label l_0, we define $x^{`}$ as the adversarial of x_0 with label $l \neq l_0$, then the loss function *f(x)* is given by:

$$f(x,l) = max\{max_{j \neq l}[Logit(x)]_j - [Logit(x)]_{l,-\kappa} \tag{2.6}$$

where Logit(x) = $[Logit(x)]_1, ...[Logit()x]_k \in R^k$ are the logits i.e. the values at the pen-layer before the softmax output layer, k is the number of classes, and κ is a non negative value which gives the confidence parameter that gives a constant gap between the values of the logits corresponding to j and l such that $j \neq l$. Considered as a robust version of CW, the algorithm uses the Iterative shrinkage Thresholding Algorithm (ITSA) [BT09], which is essentially a first order optimization algorithm with additional shrinkage -thresholding step at each iteration.

The aim of the loss function is to make l as the most probable class for x and κ controls the gap between the most probable class and next most probable class. This way the output class can be manipulated and on top of it, elastic net regularization is added with the variable β.

$$minimize \qquad c * f(x, l) + \beta \|x - x_0\|_1 + \|x - x_0\|_2^2 \quad s.t. x \in [0, 1]^p \tag{2.7}$$

Considering $\delta = x - x_0$, EAD finds the adversarial example x while minimizing the perturbation δ in terms of elastic net loss given by $\beta * \|\delta\|_1 + \|\delta\|_2^2$.

2.7 Basic Iterative Method

Goodfellow et al. [KGB16] suggested a straight forward extension to the popular FGS method by applying the same algorithm multiple times with a smaller step size, and after each step clipping the intermediate values to make sure the pixel values are still in the max perturbation ϵ-neighbourhood.

$$X_0^{adv} = X \qquad X_{N+1}^{adv} = Clip_{X,\epsilon}\{X_N^{adv} + \alpha * sign(\nabla_x J(X_n^{adv}, y_{true})) \tag{2.8}$$

This extension helps minimize the computation costs of the experiments while calculating adversaries.

3 Defense strategies

In the previous section we saw how the different methods can be used to exploit the vulnerabilities of a machine learning model, and limit their usage in security critical areas. Defenses against such attacks is still an open area, since there are a many options available to secure the models, but none of these are completely invincible against all of the possible attacks. One method might be able to provide defense against a particular type of vulnerability but does not provide protection against other. If the underlying defense mechanism for a model is known, the attacker can leverage this knowledge to carefully create an adversary which can bypass this security mechanism.

The reason why it is difficult to provide concrete protection against all such vulnerabilities is that we cannot create a theoretical model of adversary creation process [Cha+18]. Since calculating an adversary is an optimization problem most of the times, if we give the system enough number of iterations, we will get an adversarial example which can break the protection provided by the state of the art defense mechanisms. Also, providing defense comes with an overhead on performance, and in some cases the accuracy of the model also deteriorates. In this section, the available counter-measures against such attacks are discussed.

3.1 Adversarial Training

The idea proposed by Szegedy et al. [GSS14] [Sze+13] is fairly simple - train the model on the original data as well as the adversarial examples to make the model more robust against such attacks, by exposing the flaws in the way the model is learning the decision function while training. This way, after the training the model will have some idea about the hidden pockets present, which earlier could have been used as a vulnerability. This training is different from the one generally done on the augmented data set, in the sense that adversarial training uses inputs which have very low probability of ever occurring, while the augmented data set has samples from the original distribution. One can create as many adversarial samples and perceiving it as a form of augmentation, train the model in this rugged approach. Another approach is instead of using the original loss function $J(\theta, x, y)$, Goodfellow [GSS14] suggested using an alternative adversarial objective function based on the FGS method, given by:

$$\tilde{J}(\theta, x, y) = \alpha * J(\theta, x, y) + (1 - \alpha) * J(\theta, x + \epsilon * sign(\nabla_x J(\theta, x, y)), y) \tag{3.1}$$

where the value of α can be assumed anything to start with and later explore to find the most optimal value by performing binary search. The above formulation suggests that during the training, we update the adversarial examples on the go to make the model resistant to its current version's adversaries. The worst case error rate observed when the data is perturbed by an adversary, was reported to reduce from 89.4% to 17.9% in the experiments. Two other perspectives [GSS14] on this form of learning are:

- The new training method can be seen as a form of active learning, since model gets new points based on the current data points, with the benefit that there is no human intervention needed in generating those new data points.
- The FGS component in the new loss function also works as a regularizer to prevent over fitting in the network. Due to training the model on data points in the ϵ bounded region, the model is now insensitive to the small changes in that region.

The input data can also be augmented by adding random or calculated noise with a max norm value of ϵ, but the dot product of such a noise vector with the reference data vector is so small (usually 0), that the output is unaffected by that change in the input and essentially this distortion is of no importance. So adversarial training can also be seen as an operation of finding the hard examples among the noisy inputs which are misclassifed by the model, and training the model on those specific examples.

Another important point is that the adversarial training will develop protection against only those adversaries which were created on the original model, i.e. where the model architecture was completely known. However, if the model is a black box, then attacker will generate the malignant adversaries on a similar local model, trained on the same data set and can then use those adversaries against the original black box model. This property, that the adversaries for one model can be easily ported to another model, is called transferabilty [Sze+13].

3.2 Defensive Distillation

Distillation [HVD15] is a technique used to train an artificial neural network using knowledge gained from some other neural network. The intuition behind distillation is to reduce the computational complexity of the neural networks by training a large network and transferring that knowledge to a small network (often called Teacher - Student method). This small network can now be deployed on devices with very limited resources, which can not afford that huge amount of computations.

Papernot et al. [Pap+16a] suggested using a variant of above distillation method to improve network's ability to resist the adversarial attacks. The idea is to reduce the variations of output around the input, which can be achieved by reducing the magnitude of adversarial gradients. If this value is small, then we will not have problems due to large distortions at the output caused by very small changes at input. During an adversarial attack, the model trained with defensive distillation will be less sensitive to the little variations caused by the adversaries. In other words, defensive distillation will smoothen out the model.

Distillation extracts out the knowledge from the data in the form or probability vectors, which are then fed to the smaller *student* [HVD15] network to be trained on. This makes sure that the accuracy of the smaller network is comparable to the one we got in the large network. This also helps in improving the generalization capability of the *student* network.. In defensive distillation, both *teacher* and *student* network have the same architecture, i.e. the *student* is not a smaller network. The idea is not to achieve compression but to extract out the knowledge from the network and feed it back to the same network, in order to achieve pliancy against adversaries.

From the first network. we get the probability *f(x)* also called soft labels, corresponding to input labels x∈X. The soft labels carry extra knowledge compared to the original hard labels, for example, the entropy information will also be encoded. So, we will know that the two classes are somewhat similar if they have comparable probability value at the output. This prevents the model against over fitting to the hard labels, but instead provide good generalization capability.

Distillation has a parameter T called the temperature, shared across the softmax layer. The constraint imposed is that the temperature value while training the first neural network, should be large (>1) in the softmax layer. The high temperature forces the model to give ambiguous probability values, all converging to 1/N as t $\rightarrow \infty$, and vice versa. So if the T is low, we will get discrete values where one of the classes will have the highest probability value ($\simeq$ 1), which defeats the purpose of generating the soft labels. The second network will also be trained on a high temperature, while during the test time, temperature T is set back to 1. For an input X, output of softmax layer for a class $i \in 0$ ... N-1 is given by:

$$F(X) = e^{z_i(x)/T} / \Sigma_{l=0}^{N-1} e^{z_l(X)/T} \qquad for \quad i \in 0...N-1 \tag{3.2}$$

3.3 Gradient Hiding

Since most of the attacks like , Basic Iterative Method (BIM), etc are based on the gradient's value and the direction, most intuitive solution can be to simply hide the gradients [Tra+17]. Models like KNN classifier or a decision tree, which are non-differentiable,

are protected against gradient based attacks. However, these defenses can also be bypassed by training a local black-box model and transferring the adversaries across models[Pap+17].

Gu et al. [GR14] in order to force neural network to produce near-zero gradient, added a penalty term in objective function, which was a layer by layer F-norm of the Jacobian matrix. This makes the model more robust against adversaries but it takes a toll on the accuracy of the model since its capacity is reduced.

3.4 Feature Squeezing

Weilin Xu et al. [XEQ17] suggested a different approach against adversarial attacks by detecting the adversarial inputs itself. A classifier which can detect such input can directly inform the user to take necessary action. This method is also important in the sense that with more optimization, one will always be able to craft better adversaries and break the protection. But since this method can detect an adversarial input itself, it might promise effective protection.

The approach is called feature squeezing and it suggests that the dimensionality of the input is generally large and it is easy then to find an adversary. So the dimensionality of the input data set is reduced by squeezing the non-critical input features. Next we compare the output prediction on both the original and squeezed data set, and if the values differ above a threshold, then the input is most probably an adversarial example. Figure 3.1 gives an overview of how the framework looks like.

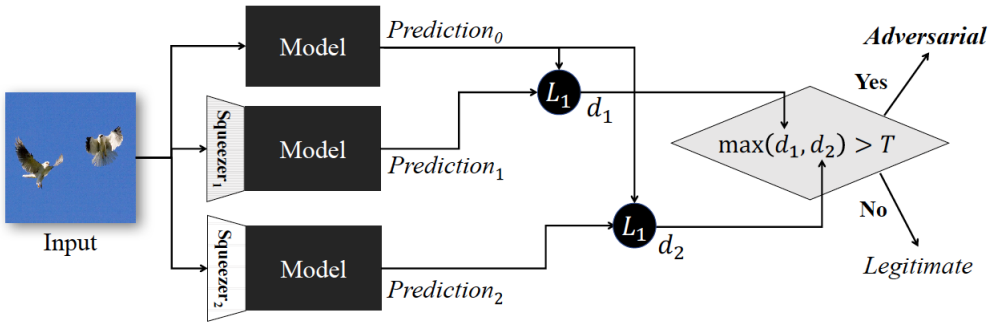

Figure 3.1: Feature squeezing framework

3.5 Defense GANs

[Samangouei] Pouya Samangoueui et al. [SKC18] proposed to use Generative Adversarial Networks (GAN) (originally introduced by Goodfellow et al. [Goo+14]) against the adversarial attacks and the framework was called Defense-GAN. The method is effective against black box as well as white box attacks. Idea is to utilize the representative power of GAN by projecting the input data onto the GAN's range, before feeding the it to the classifier.

Here, two models are trained simultaneously, a generative model G and a discriminative model D, in adversarial setting. The generative model G is fed input data samples x and a random vector z which has an Gaussian prior. G learns the underlying data distribution and generates the samples *G(z)* which resembles the original samples x. Discriminative model D learns to differentiate between real input examples x and the fake generated examples *G(z)*. Figure 3.2 gives an overview of the algorithm. The two networks are trained in alternating fashion and the following loss function is minimized:

$$min_G max_D V(D,G) = \mathbb{E}_{x\ p_{data}}[logD(x)] + \mathbb{E}_{z\ p_z(z)}[log(1 - D(G(z)))] \tag{3.3}$$

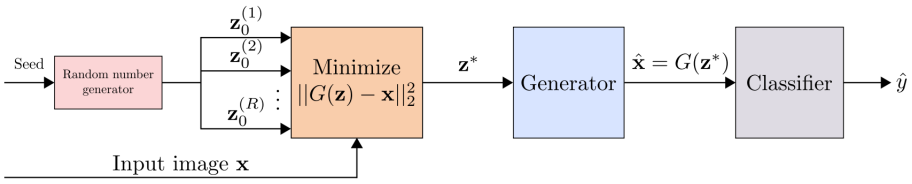

Figure 3.2: An overview of the Defense GAN algorithm

This framework employs a variant of normal GAN called WGAN, proposed by Gulrajani et al. [Gul+17] which uses Wassertein distance, with a slightly modified loss function. WGAN is trained on legitimate input training samples to eliminate adversarial samples, and before feeding data x to the classifier, it is projected on G's range while minimizing the reconstruction error.

$$\|G(z) - x\|_2^2 \tag{3.4}$$

This way, the logically correct samples are close to the range of the generative model, while adversarial samples will be away from the range of the generative model. At inference time, given input x, we find z^* which minimizes the reconstruction error mentioned earlier and then feed this latent variable to the classifier. However in practice, GANs are difficult to train and if not done so properly, fails poorly in operation.

3.6 MagNet

Meng et al.[MC17] proposed this framework which utilizes detector networks for detecting the adversaries and a reformer network to create a legitimate input from the adversarial examples using auto-encoders. The detector network exploits the information learned, to differentiate between original and adversarial samples by exploring the manifold of the data distribution. It doesn't need any knowledge of the process generating the adverasries, so it generalises well over different input samples. Since it learns the features of training set with hidden representation of certain properties, if a new example if provided from the same data generation process, we expect a small reconstruction error and vice versa. An autoencoder [HS06] is used a a detector here which uses the reconstruction error to approximate the distance between the given test sample and the manifold of the original unperturbed samples.

The reformer network strives to find an example x^* close to the manifold for given example x, such that x^* is a close approximation of x, and give this approximated sample to the classifier. In other words, it tries the reconstruct the test input data. This is only used during inference and not during the training. An ideal reformer is expected to not change the classification output for normal data samples, while change the adversarial examples just enough so that they are close to the original examples.

The technique works well with black box attacks, but if the attacker knows the internal architecture (parameters) of the detector or reformer, MagNet is prone to attack. The reason is intuitive since the MagNet transforms the classifier f to $f`^*$ and if the architecture of $f`^*$ is already known, the attacker can easily create adversarial examples for the new classifier. The approach to combat such an attack is training n number of diverse autoencoders, all with different initialization and randomly pick one for every session or every test set. The attacker can only predict which architecture was selected, and this makes it hard to perform an attack. The defense can be made even more stronger by diversifying the the autoencoders employed.

4 Simulations

For the purpose of having some hands on exercise on generation of adversarial examples and creating a defense against it, this section summarizes the approach that was followed and the simulation results received. The code for these experiments were referred from here.

A very basic feed forward neural network was used for the experiments, having only a single hidden layer with 50 units. The data set used is the Modified National Institute of Standards and Technology database (MNIST) data set (Figure 4.1), which contains handwritten digits and each digit is a 28 x 28 pixel image. This vanilla feed forward network trained on the MNIST data set gave around 97% accuracy, which is pretty decent for such a small network, given MNIST is not a very challenging data set to learn.

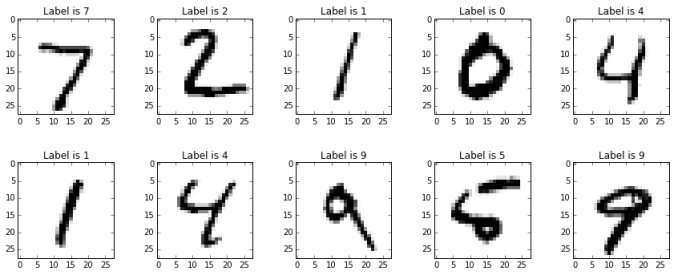

Figure 4.1: MNIST Dataset - Collection of handwritten digits

In the Figure 4.2, the network is giving the predictions against a randomly selected data input. Since, the output of the softmax layer of the neural network is a probability value, the label corresponding to the output node with highest value is the actual prediction by the network.

4.1 Creating attacks

The objective is to fool this trained network into misclassifying the labels of the input data. Two different types of approaches are shown here viz. non-targeted and targeted approach. In a non-targeted attack, the objective is to find an input data sample (here image) that makes the neural network predict a particular output label of our

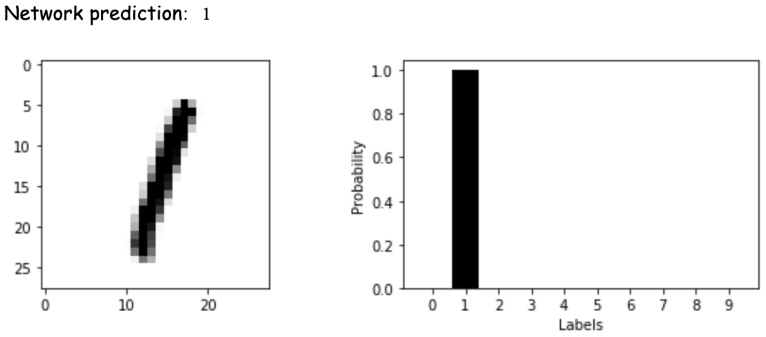

Figure 4.2: Prediction by the network on a random sample

choice. As we had seen in the previous section, this is an optimization problem, which can be solved using different algorithms. Here we have used the popular and simple to implement FGS algorithm. Since this is an non-targeted approach, we start with any random input image, which is a 784-dimensional vector we get by flattening the original image of size 28 x 28 pixels. The cost function that we have to optimize is given by:

$$C = 1/2\|y_{target} - y_{network}(x)\|_2^2 \tag{4.1}$$

where y_{target} is the target label that we wish to achieve and $y_{network}(x)$ is the output of the network for the data input x. The value of this cost function will be high if the two labels are far apart, and we have minimize this distance between the two. We do so by using the gradient descent algorithm, in a way similar to how the normal neural networks work, i.e. by using back-propagation, the values of biases and weights are changed to reduce the error. Here we do something similar, except the fact that, we update the the input values x, while the weights and bias are held constant. We can find the derivative of the cost function with respect to the input, and then update x in order to minimize this cost. After achieving the convergence, i.e. where the two labels are similar, we will have a input image x, which the network classifies as y_{target}.

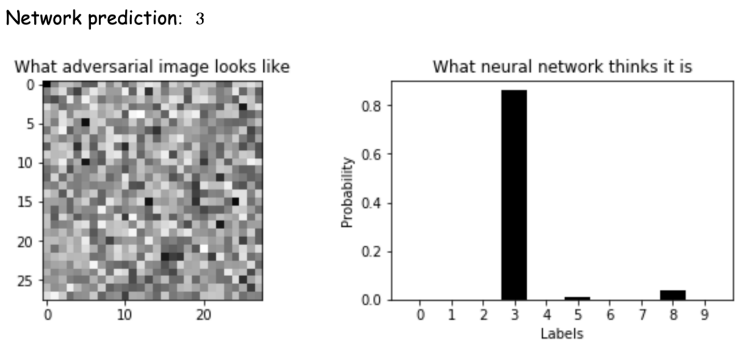

Figure 4.3: Non-targeted attack: Image misclassified by neural network as label 3

In Figure 4.3, we can see the image we got after the convergence. This image got misclassified as the digit 3 by the neural network since the probability of prediction for the digit is the highest among all labels. Note that this image will not be unique in the different iterations, and also this resultant image might not even look like an actual digit, but some random pixel values which get classified as y_{target}. The point to note is that only a fraction of total input images classified as y_{target} will look like digits to human eyes, while in the eyes of neural network, all those images correspond to that label.

In the targeted attack, the objective is to make changes to an existing data example in order to misclassify its label. The adversarial example we get from non-targeted approach looks like noise to human eye, and to actually fool the model, we want the image to look something meaningful and preferably similar to the original image, but causing misclassification by the network. The approach will be the same as before, except the part that we add one more component in the objective function. The new cost function becomes:

$$C = 1/2\|y_{target} - y_{network}(x)\|_2^2 + \lambda\|x - x_{target}\|_2^2 \tag{4.2}$$

where x_{target} is the image what we wish our adversarial example to look like. So here we two components to minimize. Minimizing the first component means to make the output label of the network to look like our target label, while minimizing the second component means making the input image look like the target input image. While we use the λ term to annotate the importance of the two terms. The resultant image from this optimization will most of the times be able to trick the neural network. In Figure 4.4, we can see that for the original image with label 1, we created an adversarial example which looks a lot similar to the original image, but gets misclassified as label 3.

4.2 Creating defense

For the purpose of creating a defense against the adversarial examples we saw in the previous section, we try two techniques here, viz. binary thresholding and adversarial training.

In the adversarial images, which are the outputs from the targeted approach, we can visually observe some slight gray noise in the background, which actually are the perturbations that we introduced. If we get rid of those, then the network might not get fooled by such samples. After using the binary threshold technique, which makes all the pixels value below 0.5 as 0 and above 0.5 as 1, we get a black and white image as output. The value of threshold can be between [0, 1] depending on the application.

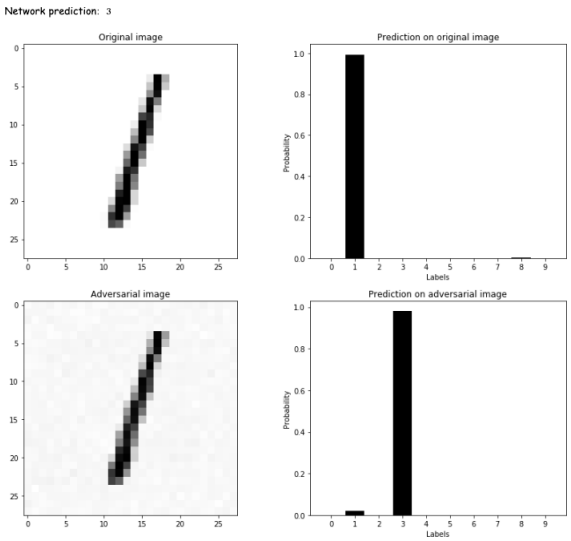

Figure 4.4: Targeted attack: Image misclassified by neural network as label 3

When given this new image, the neural network gave correct outputs most of the times, so we were able to resist the adversarial attack.

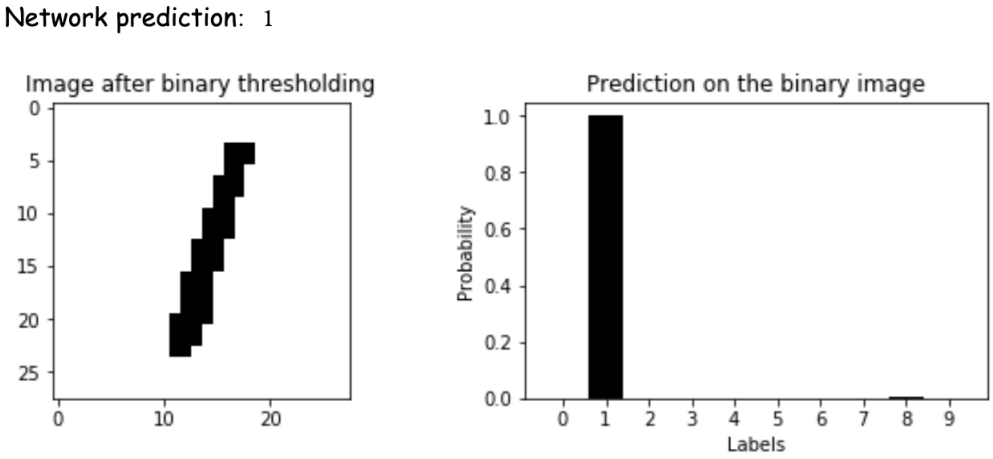

Figure 4.5: Image classified correctly after binary thresholding

Figure 4.5 shows the correct classification of the new transformed image. However, this technique won't work in cases where the background is not completely white and carry useful information. If we apply binary threshold there, we lose a lot of information, making it difficult for the network to give correct predictions.

One of the most robust ways against these adversarial attacks is to train a new network on these adversarial samples as well as the original training data set. If the network has knowledge of the various types of adversaries it might see during inference phase, it will be able to generalize better and give less test error. For this, we create a large number of adversarial samples for each of the digits, and add it to the training data set of the

new neural network. Comparing the two neural networks - with and without adversarial training can give a good idea of the performance measure. The network without the information of the adversaries performs poorly when it is given adversarial examples to classify during inference and the accuracy is only 41%. But with adversarial training, we get an accuracy of 95% on the test set including the adversaries as well.

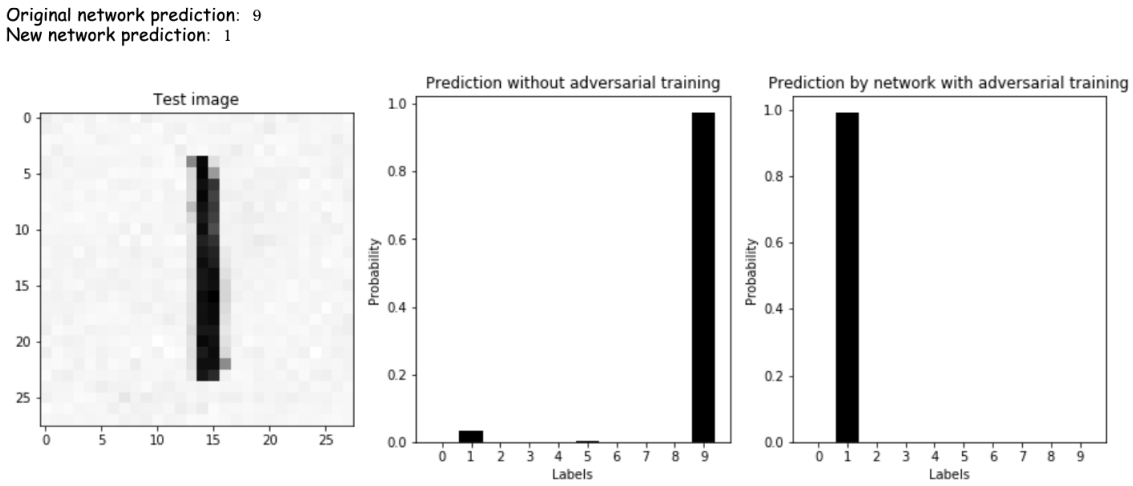

Figure 4.6: Image correctly classified in the new network with adversarial training

Figure 4.6 shows the effect of adversarial training. An example is picked at random from the adversarial test set, and compared the performance of the two different neural networks - with and without adversarial training. We can see the new network with adversarial training is able to correctly identify the adversarial sample. However, this method will also fail in some of the situations, since one cannot actually know what sorts of attacks can be developed by the attacker, since we have only worked on one example employing FGS based adversaries.

5 Summary

Despite the fact that the machine learning algorithms are so powerful, and have near super human capabilities while solving tasks which were earlier unthinkable of due to their complex nature, we observed how it is so easy to fool these models. This paper discussed how these models are vulnerable to small, specially crafted perturbations (called adversaries), which goes undetected by humans. The models fails so catastrophically and easily, and can be used in security critical areas but with multiple limitations, since they are prone to such adversarial attacks.

Since finding such "blind pockets" is just a matter of how good is the optimization process, there is a dire need of more effective defense mechanisms. The good results obtained on some of the defense mechanisms can be due to a possibility that the adversarial examples were not powerful enough. This is a motivation to do further research on finding better adversarial examples and more powerful defense strategies.

This also suggests that we might not have completely understood these algorithms and need to put in more research efforts to have a sound understanding. We should not rely on our human intuition to explain how things work in machine learning models, since even though some models are biologically inspired but these models don't think and operate the way humans do. The fragile outputs from these models and there exploitation by the adversarial examples, are a good example that even though we have come a long way ahead but still there are a lot of things that we do not understand clearly.

Appendix

The code for all the experiments are present here on this gitlab link.

List of Figures

Acronyms

BIM Basic Iterative Method.

CW Carlini-Wagner attack.

EAD Elastic Net method.

FGS Fast Gradient Sign method.

ITSA Iterative shrinkage Thresholding Algorithm.

JSMA Jacobian-based Saliency Map Approach.

L-BFGS Limited-memory Broyden–Fletcher–Goldfarb–Shanno.

MNIST Modified National Institute of Standards and Technology database.

Bibliography

[BT09] A. Beck and M. Teboulle. **?**A fast iterative shrinkage-thresholding algorithm for linear inverse problems**?** In: *SIAM journal on imaging sciences* 2.1 (2009), pp. 183–202.

[Cha+18] A. Chakraborty et al. **?**Adversarial Attacks and Defences: A Survey**?** In: *arXiv preprint arXiv:1810.00069* (2018).

[Che+18] P.-Y. Chen et al. **?**EAD: elastic-net attacks to deep neural networks via adversarial examples**?** In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[CW17] N. Carlini and D. Wagner. **?**Towards evaluating the robustness of neural networks**?** In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, pp. 39–57.

[Goo+14] I. Goodfellow et al. **?**Generative adversarial nets**?** In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[GR14] S. Gu and L. Rigazio. **?**Towards deep neural network architectures robust to adversarial examples**?** In: *arXiv preprint arXiv:1412.5068* (2014).

[GSS14] I. J. Goodfellow, J. Shlens, and C. Szegedy. **?**Explaining and harnessing adversarial examples**?** In: *arXiv preprint arXiv:1412.6572* (2014).

[Gul+17] I. Gulrajani et al. **?**Improved training of wasserstein gans**?** In: *Advances in Neural Information Processing Systems*. 2017, pp. 5767–5777.

[HS06] G. E. Hinton and R. R. Salakhutdinov. **?**Reducing the dimensionality of data with neural networks**?** In: *science* 313.5786 (2006), pp. 504–507.

[HVD15] G. Hinton, O. Vinyals, and J. Dean. **?**Distilling the knowledge in a neural network**?** In: *arXiv preprint arXiv:1503.02531* (2015).

[KGB16] A. Kurakin, I. Goodfellow, and S. Bengio. **?**Adversarial examples in the physical world**?** In: *arXiv preprint arXiv:1607.02533* (2016).

[MC17] D. Meng and H. Chen. **?**Magnet: a two-pronged defense against adversarial examples**?** In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 135–147.

[MDFF16] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. **?**Deepfool: a simple and accurate method to fool deep neural networks**?** In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2574–2582.

[Pap+16a] N. Papernot et al. **?**Distillation as a defense to adversarial perturbations against deep neural networks**?** In: *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2016, pp. 582–597.

[Pap+16b] N. Papernot et al. **?**Technical report on the cleverhans v2. 1.0 adversarial examples library**?** In: *arXiv preprint arXiv:1610.00768* (2016).

[Pap+16c] N. Papernot et al. **?**The limitations of deep learning in adversarial settings**?** In: *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2016, pp. 372–387.

[Pap+17] N. Papernot et al. **?**Practical black-box attacks against machine learning**?** In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM. 2017, pp. 506–519.

[SKC18] P. Samangouei, M. Kabkab, and R. Chellappa. **?**Defense-GAN: Protecting classifiers against adversarial attacks using generative models**?** In: *arXiv preprint arXiv:1805.06605* (2018).

[Sze+13] C. Szegedy et al. **?**Intriguing properties of neural networks**?** In: *arXiv preprint arXiv:1312.6199* (2013).

[Tra+17] F. Tramèr et al. **?**Ensemble adversarial training: Attacks and defenses**?** In: *arXiv preprint arXiv:1705.07204* (2017).

[XEQ17] W. Xu, D. Evans, and Y. Qi. **?**Feature squeezing: Detecting adversarial examples in deep neural networks**?** In: *arXiv preprint arXiv:1704.01155* (2017).