Evaluating the Landscape of Adversarial Attacks and Defenses

Jacob Steen, Kevin Chen, Chien-Te Lee, Shivang Patel

December 2018

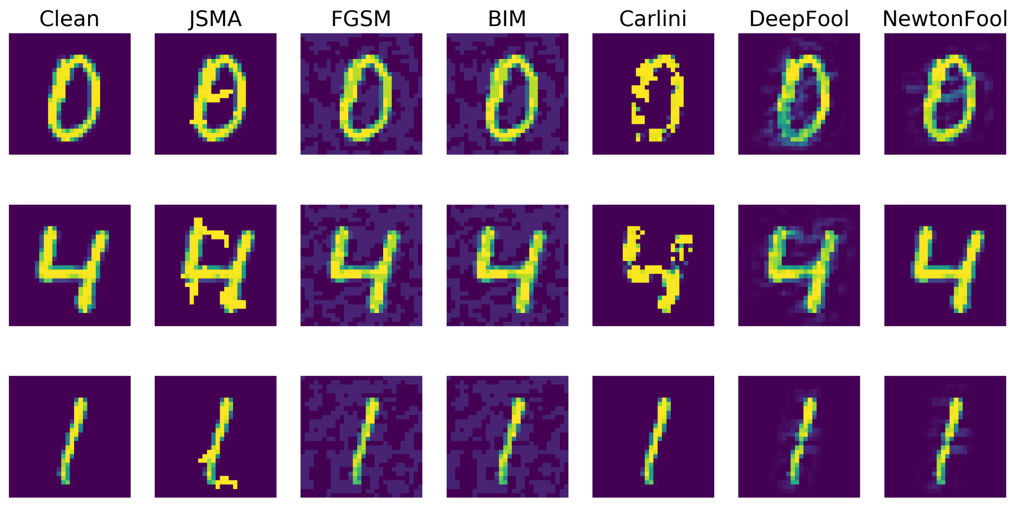

Figure 1: Images generated from several adversarial attacks on MNIST.

1 Introduction

Improvements in hardware and computer vision have led to the development of larger, more complex neural network (NN) architectures such as Deep Learning [14] and convolutional neural nets [11]. These methods allow for the successful deployment of machine learners for advanced tasks ranging from AlphaGo [20] and its successors dominating the world's best Go players, to language translation in applications like Google Translate. Increasingly, though, these learners are being used for tasks such as self-driving cars [5] [23], bank account access [15], or cyber security [1] where security is a top priority.

In such applications it is not enough to simply show that a learner achieves a certain accuracy; some errors (mistaking a pedestrian for a speed bump, for example) are obviously more problematic than others. Furthermore, many of these domains include parties with incentives to cause the learner to make certain errors. This changes the learning environment from a neutral environment to an adversarial one. Learners using DL or CNN's used for image recognition, for example, have recently been shown to be vulnerable to adversarial examples of images that are designed to cause misclassication of the image examples [22]. The learners misclassify these examples despite the fact that humans perceive no difference between the adversarial images and the true images they are crafted from. Similar results have been shown for automatic speech recognition [2], and virus/malware detection [17].

2 Background

In this section give a brief description of the attacks and defenses implemented in this study.

2.1 Attacks

2.1.1 Inference & Test Time Attacks

The set of attacks considered in this project, falls within a class of attacks known as inference/test time attacks. Test time attacks are designed to attack pre-trained networks (i.e., a model whose parameter, θ, are fixed). While the range of test time attacks currently in existence is itself quite diverse, all work by manipulating examples, x, by some factor, r. The aim is that the manipulated example $x^* = x+r$ receives a different predicted label from the classifier than x. However, it would defeat the point of the attack if x^* is so different from x as to more properly be thought of as actually belonging to a different class than x. For example, one way to get a well-trained classifier to label an example which is a '1' as a '7' is to simply change the '1' into a '7.' This kind of manipulation would not demonstrate a weakness of the network. Attackers want to manipulate the example in a way that is not so drastic.

For image classification, the goal is usually thought of as manipulating the image in such a way that the classifier changes its predicted label, but a human would not. What the standard should be for non-sensory classification tasks is less clear, but attack methods implement the idea by ensuring that x and x^* are sufficiently close to one another according to some chosen distance metric with ℓ_1, ℓ_2, and ℓ_{infty} norms being popular choices. Formally, the goal of test time attacks can be expressed by the formula

$$\arg\min_r f(x+r) = y \text{ s.t. } y \neq t \text{ and } x+r \in D$$

where y is the classifier's predicted label for $x^* = x+r$, and t is the classifier's (presumably correct) label for the original example, x. The choice of distance metric will affects which values of r minimize the equation above, and the constraint of $x^* \in D$ is meant to ensure that attackers don't shift x outside of the actual domain of inputs when creating x^*. When the classifier is not a convex function from example to label, there are a number of ways to approximate a solution for r. Different test time attacks differ in the approximation technique or heuristic they use for this purpose.

Before providing an overview of the specific attacks tested in this paper, there are two distinctions amongst the various test time attacks that are worth noting. First is the difference between what are known as white-box and black-box attacks. White-box attacks have access to various features of the model being attacked such as the type of model used, architecture if the model is a network, and parameters of the trained model. Black-box attacks do not have access to these features of the model, and are only assumed to have the ability to query the model by presenting inputs and getting the model's prediction in return.

2.1.2 FGSM

Proposed by Goodfellow et al. [7], the Fast Gradient Sign Method (FGSM) find adversarial examples $x^* = x+r$ by calculating

$$r = \epsilon \text{ sign}(\nabla_x C_\theta(x,t)$$

where J_θ is the error/cost function of training the classifier with parameters θ, and t is the true label of example x. The FGSM attacker, thus, finds the gradient of this cost function with respect to each feature of x, but uses only the direction of the gradient with respect to the features, not the magnitude of the gradient. ϵ controls how much each feature of x is perturbed along the gradient. If ϵ is too small, the resulting adversarial example, x^*, might not fool the classifier. If ϵ is too large, changes might be detected or even amount to changing the true classification of x^* from that of x. There are two important feature of FGSM worth noticing. First, is that it relies on the idea that making small changes to each and every feature of x will lead to a x^* that fools the classifier without being noticed. The value of this approach will be challenged by many other attacks is explored. Second, FGSM only calculates the $\nabla_x C_\theta(x,t)$, once and manipulates x a single time to yield the resulting x^*. This has the advantage of making FGSM a fast method of attack, but it has the disadvantage of making the resulting adversarial examples less likely to fool classifiers, on average, than those generated by methods that recalculate the gradient after each change made to x and use this to make further changes x. These are referred to as iterative methods/attacks.

2.1.3 JSMA

The Jacobian-based Saliency Map Approach (JSMA) differs from FGSM by only manipulating a small number of input features on a given example by a larger amount. Papernot et al. [18] designed this attack to generate adversarial examples via a (repeated) three-step process.

First, the algorithm calculates the Jacobian matrix for the unmanipulated example, x for the trained network, f,

$$J_f(x) = \frac{\partial f(x)}{\partial x} = \left[\frac{\partial f_j(x)}{\partial x_i}\right]_{(i\in M)\times(j\in N)}$$

where M and N are the input size and number of class labels, respectively. Next it uses this matrix to create an adversarial saliency map indicating which features of x have the most impact on f's prediction, if perturbed. Finally, the attacker manipulates the feature that is most likely to make $f(x^*) \neq t$ by a predefined, constant amount, ϵ. This process is then repeated using the newly manipulated sample for each iteration until either the classifier's prediction has changed or the predefined, maximum distortion value, Υ, has been reached. The values used for both ϵ and Υ are task-specific and need to be chosen so as to avoid the final, manipulated example, x^*, being detected as fraudulent.

2.1.4 DeepFool

Developed by Moosavi-Dezfooli et al. in [16], the DeepFool attack is grounded in how one would find an adversarial modification of an example x for a classifier when the classifier, f, is both binary and linear. In this simple case, the goal would be to first find the decision boundary (an affine hyperplane), $\mathcal{F} = \{x : w^\top x + b = 0\}$. Then DeepFool projects the example x onto $\mathcal{F}$, and nudges it just over this boundary so as to change f's prediction by multiplying the manipulation vector, r, by $1+\eta$, where

η is very small. The resulting manipulated example, x^* can be expressed by the closed-form formula

$$x^* = (1+\eta)x(\arg\min \|r\|_2)$$
$$\text{s.t. } sign(f((x+r))) \neq sign(f(x))$$

which the authors show gives us $x^* = -(1+\eta)x\dfrac{f(x)}{\|w\|_2^2 w}$

To handle general (non-affine) classifiers they use a linear approximation of $\mathcal{F}$ around x, manipulate x just as in the affine case (but without the final nudge) to obtain $x_i = -x\frac{f(x)}{\|w\|_2^2 w}$, and repeat the process by approximating $\mathcal{F}$ around x_i. This process is iterated until one of the stages yields a manipulated example, x_n, that lies on F, at which point the final nudge is given to get the adversarial example, x^*. When the classification task is not binary, but has M different classes, the approach is to find which of the $M-1$ hyperplanes is closest (either directly for affine classifiers, or via linear approximation for non-affine), and proceed as normal. When f is non-affine though, which hyperplane is closest, and, hence, which class Deepfool is aiming to get f to mislabel x^* as can change with each iteration of this process. The authors also show how to extend DeepFool from the ℓ_2 distance norm to any other ℓ_p norm ($p \in [0, \infty]$).

The main difference between DeepFool and both FGSM and JSMA is its reliance on linear approximations of the separating hyperplanes when finding x^*, and how it manipulates individual features of x to obtain x^*.

FGSM manipulates each feature by a small but uniform amount at each step until a decision boundary is crossed, whereas JSMA only manipulates a single feature at each step until the same.

DeepFool (potentially) manipulates each feature at each step like FGSM, but weights the change of each feature individually. The authors' empirical studies show that this yields smaller perturbations on individual features than JSMA and less overall manipulation of features than FGSM. Both of these advantages could be useful for avoiding detection.

2.1.5 NewtonFool

NewtonFool [9] is a recent proposed improvement on DeepFool. The authors make use of an aggressive assumption: that around the original data point x, there exists some neighboring data point where the confidence in the true class is significantly lower. Starting with this assumption, they choose some threshold p and apply Newton's method to solve the equation: $f(x) - p = 0$, where f is the classifier described in DeepFool.

2.1.6 C&W Attack

The Carlini and Wagner (C&W) Attack developed by its namesakes in [4] generates adversarial examples by reformulating

$$\arg\min_r f(x+r) = y \text{ s.t. } y \neq t \text{ and } x+r \in D$$

into an optimization problem that can be solved by existing methods. The constraint that $f(x+r) \neq t$ makes the problem as stated highly non-linear and difficult to even approximate. The authors get around this difficulty by defining an objective function g such that $g(x+r) \leq 0$ <u>iff</u> $f(x+r) \neq t$. They consider seven candidates for g with varying success, but settle on

$$g(x) = \max(\max_{i \neq t}(Z(x)_i) - Z(x)_t, -k)$$

where k is a hyperparameter (often set to 0) meant to ensure that g respects its defining inequality, and $Z(x)_n$ denotes the Softmax output for class n of x. Depending on the distance norm used, the C&W Attack then generates an adversarial example in a single step (for ℓ_2 version), or through an iterative process (for both ℓ_0 and ℓ_{infty} versions).

2.1.7 BIM

Kurakin et al. [12] developed the Basic Iterative Method (BIM) as a way to improve FGSM. This improvement is is due to two main alterations in BIM over FGSM. First, BIM is an iterative method that uses the same formula

$$r = \epsilon \text{ sign}(\nabla_x C_\theta(x,t))$$

to perturb x repeatedly, it uses a smaller perturbation at each step (smaller value for ϵ), and recalculates $\text{sign}(\nabla_x C_\theta(x,t)$ at each step based on the example output by the previous step. BIM also tries to limit how much any given feature of x gets altered in this process by clipping values of individual features after each step when necessary via the rule

$$Clip_{x,\xi}\{x'\} = min\{255, x+\xi, \max\{0, x-\epsilon, x'\}\}$$

for grey-scale images with pixels as features. This means that BIM actually generates a series of adversarial examples based on x according to the process

$$x_0 = x$$
$$x_{i+1} = Clip_{x,\xi}\{x_i + \epsilon \text{ sign}(\nabla_x C_\theta(x_i,t)\}$$

with each successive x_i more likely to be misclassified by f than the earlier examples. The process stops when either a generated example is sufficiently likely to be misclassified by f, or when the *max-iter* hyperparameter is reached.

2.2 Defenses Against Test Time Attacks

While attacks on ML methods can be fit into a fairly thorough taxonomy by distinguishing between what different attack methods know, or are aiming to do, etc., the wealth of proposed defenses to these attacks cannot be grouped together quite so well. But there are commonalities between some of the existing defenses, including those considered here, as we will point out below.

2.2.1 Adversarial Training

First proposed in [22], Adversarial Training is perhaps the most straight-forward approach to defending against attacks. This approach involves generating adversarial examples designed to target the classifier, and retraining the classifier with these adversarial examples (along with their correct labels) mixed into the new training data. The adversarial examples used for training the classifier can be generated from any attack method or from several methods.

The goal with adversarial training is not just to get the classifier to correctly label those adversarial examples identical with (or very close to) those used in the training set. An actual attacker will generate adversarial examples independent of those attackers used for training, and so, this would be unlikely to help classifiers against new adversaries. The goal of adversarial training is instead to smooth the classifier's likelihood outputs for different labels as input features vary. This is done by making the classifier predict the same labels for unseen (adversarial) examples that are only a small perturbation away from already seen examples.

Adversarial Training does have a few pressing disadvantages though. First, it can be taxing on resources both in terms of time: the network has to be retrained on an even larger training set than the original, and if the attack methods used are computationally expensive, adversarial training that needs to use these attack methods will also be computationally expensive. Second, classifiers have to be trained using adversarial examples that were generated based on that same classifiers parameters, which is not always possible. Last, as Zantedeschi et al. [26] note, Adversarial Training is not as successful at making classifiers robust against new adversarial examples that have perturbations of similar magnitude but different direction than those used during training as one would hope.

2.2.2 GDA

The basic idea that motivates Gaussian Data Augmentation (GDA) as a defense against adversarial examples [26] is the same as that behind adversarial training: augment the training data with slight perturbations of those in the original training set to smooth a classifier's predictions and increase robustness to future adversaries.

GDA differs from Adversarial Training as it only presents the classifier with perturbations of the original examples in the directions that attackers choose. Attack methods generally aim for a perturbation that will approach or cross a decision boundary in the shortest distance, and so adversarial examples generated from some true example, x, will be highly similar to one another, even if generated from different attacks. This has the effect of only smoothing the classifier's predictions in the direction of the generated adversarial examples and leaving the classifier vulnerable to perturbations in other directions, and these perturbations may be only slightly larger than those of the adversarial training examples. At test time, new attacks can take advantage of this vulnerability.

GDA avoids this problem by generating the additional training examples through perturbations of the original examples in random directions. Further, the magnitude of each generated example is both capped by the hyperparameter σ, and sampled from a Gaussian distribution centered around the example. The fact that the direction chosen for each new training example is random avoids the issue of classifiers being only robust to perturbations in certain directions. The fact the magnitude is sampled from a Gaussian distribution rather than a uniform distribution up to σ means that the classifier's confidence of predictions is highest for new samples closer to the originals. This has the desired smoothing effect on the classifier's labeling. Additionally, the authors showed that unlike standard Adversarial Training, GDA does not require that classifiers be completely retrained. The classifier only needs additional training on the newly generated examples.

2.2.3 Label Smoothing

Warde-Farley and Goodfellow's Label Smoothing defense [24] is based on the goal of a separate defense method, defensive distillation [19] that was shown to be 'breakable' by Carlini and Wagner [3] and is not considered in this paper. Like Adversarial Training and GDA, Label Smoothing alters a given classifier's training procedure. However, whereas those two defenses altered training by augmenting the training set with new, perturbed examples, Label Smoothing alters the goal of training slightly and produces no new examples.

The last layer of a neural network classifier is (almost always) a Softmax layer, which produces likelihood estimates for a given example being labeled as each of the available classes. The classifier minimizes loss when $Z(x)_t = 1$ and $Z(x)_{i \neq t} = 0$. This has been noticed by some [21] as a strange assumption for many domains where an image of one class (e.g., a house cat) may be fairly close to those of another class (e.g., a lion) but very far from examples of other classes (e.g., seals, trucks, etc.). This has the effect of forcing the classifier to shift its Softmax outputs from favoring one class to another rapidly as it moves from across the input space.

Label Smoothing fits this overconfidence of the classifier on ambiguous or edge-case examples and the lack of robustness this leads to by softening the target output of the classifier's Softmax layer. In their empirical tests the authors adjusted the targets to $Z(x)_t = .9$ and $Z(x)_{i \neq t} = \frac{1}{10k}$ where k is the number of classes.

2.2.4 Feature Squeezing

First proposed by Xu et al. [25], the Feature Squeezing defense method takes a slightly different approach to handling adversarial examples than the defenses considered so far. Whether it was augmenting the training data, as with Adversarial Training and GDA, or altering the classifier's objective function, as with Label Smoothing, these defenses seek to make the classifier more resistant to attack by changing the model itself. Feature Squeezing, on the other hand, leaves the classifier unchanged after its employment. Instead, it creates a detector that exists independent of the classifier. This detector helps identify whether incoming test examples are true examples or adversarial examples. This allows the classifier to fully utilize its training to predict true examples as normal, while either ignoring adversarial examples or finding a way to project them back over the decision boundary towards their true classification.

The detector works by comparing the classifier's Softmax outputs for a given text example, x, with its Softmax outputs on an altered version of x, x'. It compares these outputs using the ℓ_1 norm, which just returns the maximum difference of any class between $Z(x)$ and $Z(x')$. If this difference is sufficiently large, the detector flags x as an adversarial example.

The name Feature Squeezing is derived the way in which this defense alters x to produce x'. The authors noticed that many problems involve feature spaces that are unnecessarily large and fine-grained for the learning task at hand. For example, the classic MNIST dataset [13] consists of 28x28 pixel images, which results in 784 separate pixels/features. Each of these features can take one of 256 values. Feature Squeezing makes each of these features more course-grained by lumping all of these values into one of 5 boxes ([0, 51], [52, 102], [103, 153], [154, 204], [205, 255]), for example.

The intuition is that most attacks generate adversarial examples by attempting making alterations to true examples that are small, but cross a decision boundary from the true class' region to that of another class. When the classifier makes its predictions on the more course-grained versions of the examples, many of the small perturbations to each feature will be washed out. For most true examples, the fine-grained and course-grained predictions will be very similar, but for adversarial examples, the elimination of many perturbations to individual features via this lumping procedure will alter the predictions. The label predicted by the classifier on the course-grained version of an adversarial example is actually more likely to be the label of the true example that was used to generate the adversarial example in the first place, and, hence, would be what the classifier should predict for the adversarial example, though the authors got weaker results to support this claim.

2.2.5 Spatial Smoothing

Also developed in [25], Spatial Smoothing is actually proposed as a another method of feature squeezing, and is suggested to be used in combination with the Feature Squeezing defense just described. Spatial Smoothing also generates an altered version of a given test example x, x' and compares its Softmax outputs to that of x in the same manner described for Feature Squeezing. Where the two methods diverge is in how they alter x to produce x'.

The idea behind the way in which Spatial Smoothing alter test examples is based on the technique of the same name used in image processing to reduce the noise of images and reduce file sizes. It uses a filter/kernel to scan over the image and adjust the value of each pixel based on the values of its neighbors, or even by some more complex function of a larger area of the image. The authors test their defense separately using a median smoothing filter, which did particularly well against ℓ_0-based attacks such C&W and JSMA, and a variant of the Gaussian kernel, which performed better on higher resolution, color images such as those of the ImageNet dataset.

3 Hypotheses

Our project aimed to test two hypothesis about adversarial attacks and defenses.

Hypothesis 1: Because distinct adversarial attacks employ different methods for generating adversarial examples, we believe that no defense method will allow classifiers to be robust against all attacks.

Hypothesis 2: Because larger networks (deeper or more nodes) allow for more complex decision surfaces, adversarial attacks will decrease the accuracy of deeper networks more than they will for shallower networks.

4 Methods

4.1 Datasets and Models

We evaluate these attacks and defenses on MNIST, a dataset of 28x28 images of handwritten digits. This dataset contains 60,000 images as a training set and 10,000 separate images as a testing set.[1]

We use standard neural net architectures for classification. We use a 2-layer feedforward neural network, a 4-layer feedforward neural network, a 7-layer feedforward neural network, and a 10-layer feedforward neural network. After training, these all achieve at least 95% classification accuracy on MNIST on both the training and test set. We had also attempted to run the attacks/defenses on shallow convolutional neural nets (CNN), but resource constraints prevented us training these long enough to produce accurate performance results on the five defenses we have implemented. Thus, while we include a 2-layer CNN in our discussion of attack method performance, we exclude convolutional nets from our discussion of defenses.

4.2 Attack Methods

We use Keras, Tensorflow and the IBM Adversarial Robustness Toolbox to implement Jacobian-based Saliency Map Attack, Basic Iterative Method, Carlini-Wagner, DeepFool, and NewtonFool.

4.3 Defense Methods

Again, we use Keras, Tensorflow, and the IBM Adversarial Robustness Toolbox to implement Adversarial Training, Gaussian Data Augmentation, Feature Squeezing, Label Smoothing, and Spatial Smoothing.

[1]All code for simulations available at https://github.com/kevchn/adversarial

4.4 Evaluation

For every classifier, we independently implement each of the five defenses described in the previous section to fortify the classifier from adversarial inputs. For each defense, we then run each of the five previously described attacks to generate adversarial inputs for the classifier. Each time, we record the resulting accuracy and pickle the adversarial inputs. We repeat this for every classifier architecture described in the preceding sections.

5 Results and Discussion

5.1 Adversarial Attacks

We find that all five test adversarial attacks do indeed decrease the accuracy of the baseline classifier, though some are more successful at this than others. From Figure 2, we see that FGSM and BIM decrease both the train and test time accuracy of the 2-layer feedforward classifier from over 97 percent to under 18 percent. We also see that JSMA significantly lowers 2-layer feedforward classifier's accuracy to 0.43 percent on the training set and 0.16 percent on the test set. We generally see the same trends mirrored in the 2-layer convolutional net case, as shown in Figure 3.

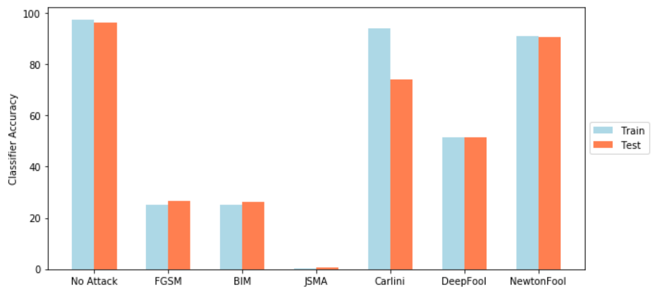

Figure 2: Classifier accuracy of 2-layer feedforward net over adversarial inputs

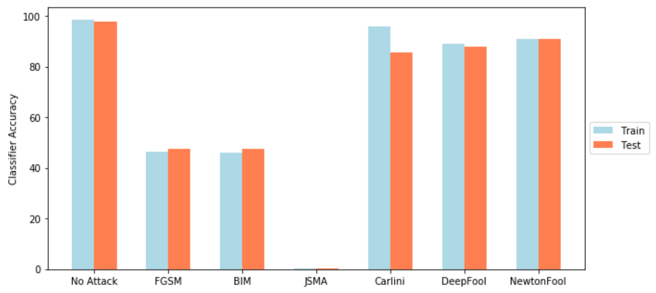

Figure 3: Classifier accuracy of 2-layer CNN over adversarial inputs

One noticeable difference however, is that FGSM and BIM are slightly less effective against our convolutional network, hovering slightly below 50 percent accuracy, as opposed to under 18 percent in the feedforward neural network case. Another difference between the two categories of architectures is that DeepFool performs significantly worse in the convolutional net – only dropping classifier accuracy to just under 90 percent.

It appears from these experimental results that while all adversarial attacks work to some extent, JSMA drops classification to a near-zero accuracy, FGSM and BIM – which are both efficient, iterative methods – drop classification accuracy significantly, while Carlini, DeepFool, and NewtonFool don't affect accuracy as much as the first three. We also see that these attacks are more effective on feedforward neural networks than convolutional neural networks, except in the case of JSMA.

5.1.1 Adversarial Inputs

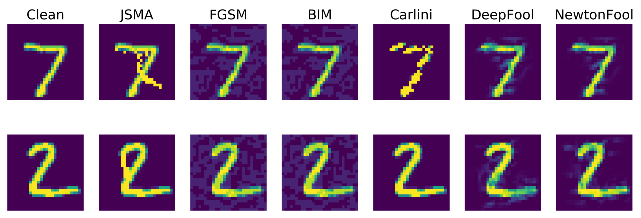

Figure 4: First two adversarial inputs from attack methods on a 2-layer convolutional net

In Figure 4, we can examine a sample of adversarial inputs produced by each attack method. On the leftmost column, we see the first two baseline MNIST digits in the training set: a seven and a two. From inspection, we can see that all adversarial attack methods result in images that resemble seven and two. However, these images have all been altered in such a way as to maximize their probability of being misclassified by the classifier.

Recall from the previous section that JSMA had the highest efficacy on the accuracy of the tested classifiers. We can see here that JSMA has added significant artifacts: a small 'L' shape on the seven and a line which creates a closed loop on the two. We know that convolutional neural networks use low-level features such as edges and curves as the building blocks for vision, so it make sense to create artifacts that target this aspect of convolutional nets.

Interestingly, we see that FGSM and BIM do not create artifacts on the edges themselves, but rather add background noise around the numeric figures to encourage misclassification. This is a far more low-level attack than JSMA, and reveals that classifiers don't necessarily only look at edges and curves, but can instead be significantly perturbed by simple noise.

DeepFool and NewtonFool seem to result in the same category of visual images with added background noise as FGSM and BIM, albeit with a smaller space of influence. Since DeepFool and NewtonFool failed to be as successful as FGSM and BIM in lowering classifier accuracy, this suggests that DeepFool and NewtonFool were not able to converge to find proper solutions, given their computationally taxing processes.

Finally, we recall that the Carlini method only had very minute impacts on accuracy. The resulting images we see here seem to neither add background noise, nor add additional high level artifacts. Instead, seven has been occluded in several areas, while two has been hardly affected at all. This may suggest that the process of taking away from existing figures may not be an optimal way to shift an input to a different class, at least in the case of handwritten integers.

Overall, these images seem to suggest that JSMA effectively targets high level visual features of images to shift the input to a new class, whereas FGSM and BIM (as well as DeepFool/NewtonFool, to a lesser degree) are exploiting the nonsmooth geometric decision surface of our classifiers.

5.2 Adversarial Defenses

After evaluating the effectiveness of the six attacks we consider on a set of baseline networks, we fortified the networks with each of five defense described in section 2.2. Plots showing how each defense performs against each attack and on each architecture can be found in the Supplementary Figures section, but we include a few of these here to illustrate our findings.

Figure 5 shows show each attack performed against undefended networks of varying depths. As section 5.1 pointed out, JSMA dropped the networks' accuracies to almost 0. When any of the defenses considered was used, however, JSMA's efficacy dropped significantly. Figure 6 summarizes the defenses' performance against JSMA. The 2-layer NN was able to achieve over 90% test accuracy on JSMA adversarial examples using any of the defenses except Feature Squeezing and Label Smoothing. Even these two defenses still increased accuracy to over 70%, which is a significant boost from the baseline results. While still performing significantly better than the undefended versions, most defenses were not as helpful to the deeper networks tested. For example, for the 10-layer feedforward classifier, none of the defenses were able to achieve over 80% accuracy against JSMA, and two defenses failed to even hit 50% accuracy.

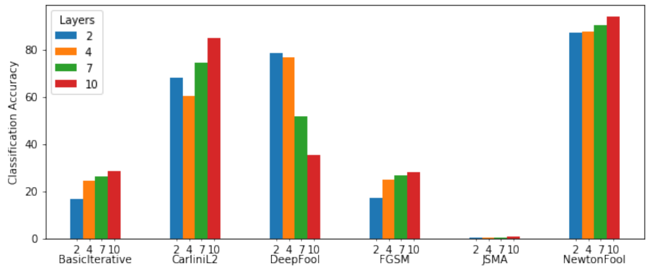

Figure 5: Test accuracy of feedforward neural net with no defense

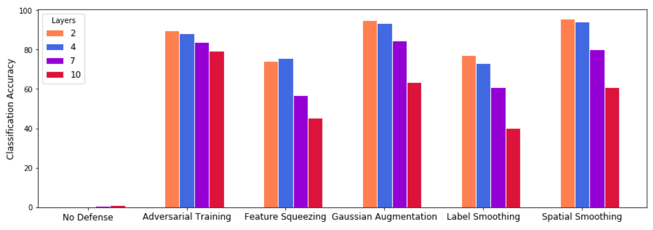

Figure 6: Test accuracy of JSMA attack method across defenses

For undefended networks, BIM was about tied with FGSM as the second most effective attack method. Once we switch to defended networks, BIM is the most effective form of attack. As Figure 7 indicates, BIM actually stays within 5% of the forced error rate against all defenses, except Adversarial Training.

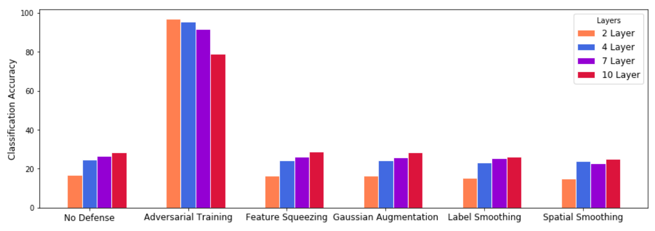

Figure 7: Test accuracy of Basic Iterative attack method across defenses

While most of the defensive considered showed mixed results against attacks, Adversarial Training was able to successfully defend against all six attacks. Both BIM and JSMA did cause a noticeable decrease in accuracy, with the 10-layer network dropping just below 80% accuracy against both attacks. However, these same two attacks dropped the accuracy of the 10-layer network to 60% or below for everyother defense tried.

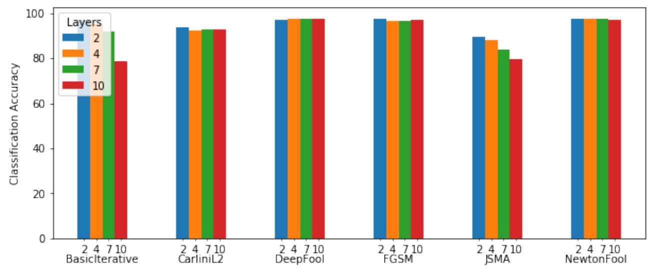

Figure 8: Test accuracy of feedforward neural net with adversarial training

6 Conclusion and Discussion

Using the simulation results presented above, we now discuss the conclusions we draw regarding our hypotheses. We also discuss limitations on our results and suggestions for future work.

6.1 Hypothesis 1

We originally hypothesized that no defense method would be robust against all adversarial attacks. Our results show that most of the the defenses considered were vulnerable – and often to multiple attacks. Adversarial training, however, performed shockingly well in recovering the classifier accuracy for all adversarial attacks that we tested. We found that adversarial training recovered to a minimum of 80 percent accuracy across all architectures on not only the training set, but even the test set as well. This suggests that adversarial training may have some capability to generalize through a meaningful shifting of decision boundaries, rather than a rote memorization of the adversarial inputs.

6.2 Hypothesis 2

We also originally hypothesized that complex networks (greater network depth or number of nodes) would be more vulnerable than simpler networks to the threat of adversarial attacks. We took this position because complex networks may have jagged, compound decision surfaces, which increases the likelihood that adversaries will be able to generate examples that are highly similar to true examples, yet receive different predicted labels from the classifier.

Support for this hypothesis was mixed, though generally counts against it. Only one attack method, DeepFool, performed better against larger undefended networks than against smaller ones, and most other attacks actually showed trends in the opposite direction. The data does support indicate that defenses may be more effective against attacks when used on deeper networks, but the evidence is not conclusive and this is a different hypothesis than we sought to test.

6.3 Limitations and Future Work

Given increased time and availability of computational resources, it would be beneficial to expand our results to encompass a larger diversity of neural architectures, attacks, defenses, and datasets.

In the future, we would like to explore the efficacy of adversarial attacks and defenses on deeper networks, such as ResNet [8], on larger and more challenging datasets (from a classification perspective) such as ImageNet [6] or CIFAR-100 [10].

We would also like to evaluate whether or not the robust Adversarial Training defense appears to confer on networks that are attacked by methods that are significantly different than those used to generate the adversarial examples used for training. We suspect not, but this would depend on how distinct the sets of adversarial examples generated by different methods actually are – which is itself an interesting question for further inquiry.

References

[1] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2016.

[2] Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. Hidden voice commands. In *USENIX Security Symposium*, pages 513–530, 2016.

[3] Nicholas Carlini and David Wagner. Defensive distillation is not robust to adversarial examples. *arXiv preprint arXiv:1607.04311*, 2016.

[4] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[5] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[7] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples (2014). *arXiv preprint arXiv:1412.6572*.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[9] Uyeong Jang, Xi Wu, and Somesh Jha. Objective metrics and gradient descent algorithms for adversarial examples in machine learning. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 262–277. ACM, 2017.

[10] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[13] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[15] Charlotte Middlehurst. China unveils world's first facial recognition atm, 2015.

[16] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[17] Nicolas Papernot. Characterizing the limits and defenses of machine learning in adversarial settings. 2018.

[18] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.

[19] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE, 2016.

[20] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[22] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[23] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering*, pages 303–314. ACM, 2018.

[24] David Warde-Farley and Ian Goodfellow. 11 adversarial perturbations of deep neural networks. *Perturbations, Optimization, and Statistics*, page 311, 2016.

[25] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[26] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. Efficient defenses against adversarial attacks. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 39–49. ACM, 2017.

7 Supplementary Material

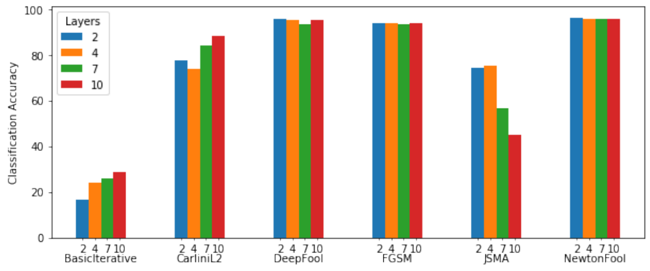

Figure 9: Test accuracy of feedforward neural net with feature squeezing

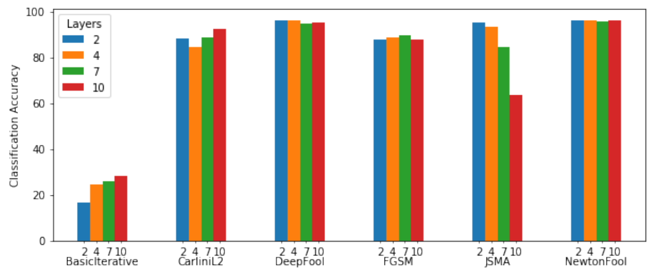

Figure 10: Test accuracy of feedforward neural net with Gaussian augmentation

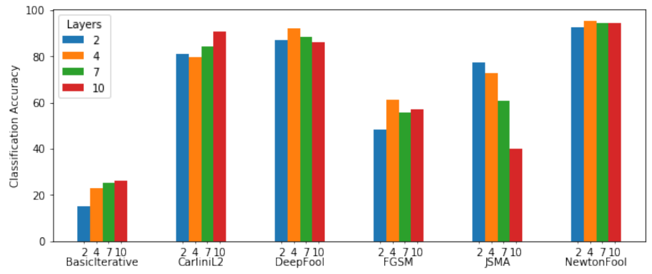

Figure 11: Test accuracy of feedforward neural net with label smoothing

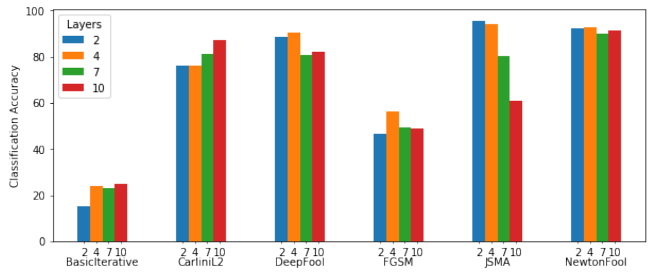

Figure 12: Test accuracy of feedforward neural net with spatial smoothing

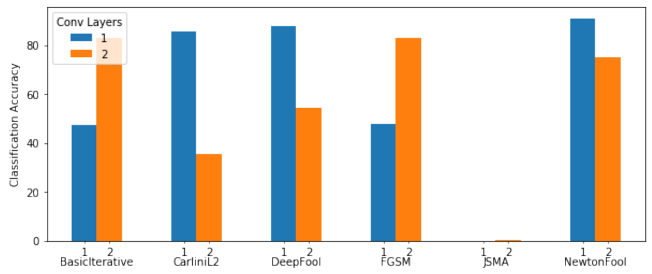

Figure 13: Test accuracy of convolutional neural net with no defense