Evaluation and Design of Robust Neural Network Defenses

by

Nicholas Carlini

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor David Wagner, Chair
Professor Dawn Song
Professor David Bamann

Summer 2018

Evaluation and Design of Robust Neural Network Defenses

Abstract

Evaluation and Design of Robust Neural Network Defenses

by

Nicholas Carlini

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor David Wagner, Chair

Neural networks provide state-of-the-art results for most machine learning tasks. Unfortunately, neural networks are vulnerable to test-time evasion attacks (i.e., *adversarial examples*): inputs specifically designed by an adversary to cause a neural network to misclassify them. This makes applying neural networks in security-critical areas concerning.

In this dissertation, we introduce a general framework for evaluating the robustness of neural network through optimization-based methods. We apply our framework to two different domains, image recognition and automatic speech recognition, and find it provides state-of-the-art results for both. To further demonstrate the power of our methods, we apply our attacks to break 14 defenses that have been proposed to alleviate adversarial examples.

We then turn to the problem of designing a secure classifier. Given this apparently-fundamental vulnerability of neural networks to adversarial examples, instead of taking an existing classifier and attempting to make it robust, we construct a new classifier which is provably robust by design under a restricted threat model. We consider the domain of malware classification, and construct a neural network classifier that is can not be fooled by an insertion adversary, who can only insert new functionality, and not change existing functionality.

We hope this dissertation will provide a useful starting point for both evaluating and constructing neural networks robust in the presence of an adversary.

Dedications

bbd7cf8166e40c0ab87b683a68a5ce3f

d3b01ccd6aa1e3856239a03ea9ecbfdd

Contents

List of Figures

List of Tables

Acknowledgments

This dissertation would not have been possible without the support I have received from many: my advisor, David Wagner, for mentoring me during both my undergraduate and graduate years; Linda Lee and Christopher Thompson, for the conversations in our office, Grant Ho and Paul Pearce, for useful discussions on malware; Brad Miller and Alex Kantchelian, for useful discussions on machine learning, Michael McCoyd, for his continuous feedback on talks and papers; William Robertson and Adrienne Porter Felt, for their mentorship as an undergraduate; Dawn Song, Anthony Joseph, and David Bamman, for their advice and guidance through this dissertation process; my parents, Susan Rendina and Giuliano Carlini, for everything else; and finally, Anensshiya Govinthasamy, for always being there.

Chapter 1

Introduction

Recent advances in machine learning have resulted in it being used in increasingly many domains: image recognition, once seen as one of the most difficult tasks in computer science [91] is now reaching human-levels of accuracy [124]; a machine learning algorithm has defeated the world's best Go players [118], a feat believed to be more than a decade away; and self-driving cars [14, 16] can now be seen on many streets.

The same approach has driven many of the recent successes in machine learning: neural networks. All of the previously-mentioned successes (image recognition, playing go, and self-driving cars) as well as many others (speech recognition [132], NLP [107], caption generation [133], etc) rely heavily on neural networks. While the ideas date back to the 1940's, neural networks' first true success came the 1990s when they were shown to have higher accuracy than any other machine learning algorithm at recognizing hand-written digits. Further still, it wasn't until 2012 that their true power was put to use when Krizhevsky *et al.* submitted the only neural-network-based approach and blew away the competition in the 2012 ImageNet Large Scale Visual Recognition Challenge. In a testament to the efficacy of this approach, in 2013, *all* top submissions used neural networks [110]. In the three years following, the error rate has dropped by a factor of four to 3% top-5 accuracy.

The security of machine learning, and neural networks in particular, is therefore of utmost importance. Without a strong understanding of the ways in which neural networks fail when attacked, it would be worrying to apply them in security-critical domains.

In this dissertation we investigate evasion attacks on these classifiers. An adversary is given a classifier and a correctly classified sample, and wishes to distort the sample so the classifier assigns this distorted sample a different label. We construct powerful

evasion attacks and apply them to many domains, finding that nearly all neural networks are vulnerable to our attacks. Then, we propose our own classifier to detect malware that is provably robust to specific classes of attack.

1.1 Motivation

The problem of evasion attacks on machine learning problems is not new. One of the first wide-scale uses of machine learning for classification in the presence of an adversary was spam [26], followed immediately by evasion attacks by adversaries who still wanted to send spam. Much of the existing work on evasion attacks, however, has studied conventional machine-learning tasks, such as SVMs, decision trees, or simple Bayesian prediction.

Neural networks currently provide state-of-the-art results across many tasks that were previously unsolvable (or, nearly unsolvable), each of which must consider adversaries who seeks to cause misclassifications:

1. In **image recognition**, a model is given an image and it must output a classification of what the object in that image is. Current image recognition networks perform at human levels of accuracy [124]. A related problem is that of **caption generation** where we train a model to create a caption for an image; neural networks have become accurate at this, too [133].

   Neural networks in these cases must be robust against an adversary who can make slight changes to the input pixels, to cause the object classification to be changed. For example, it has been shown that an adversary can generate images where, even after taking a picture of them, the photographed image remains misclassified [66].

2. Neural networks have been trained to **play games** at levels higher than any human. This includes the game of Go [118], Chess, and traditional video games [85, 28].

   An adversary in this case can be seen as the opponent, who wishes to beat the neural network agent. Recent work has demonstrated evasion attacks are possible on simple games [49].

3. **Self-driving cars** [14, 16] are now common occurrences on the roads. In this setting, the neural networks make decisions about how a car should drive.

   There are many possible goals an adversary may have. The simplest is simply to cause harm, by causing a car to drive the wrong direction after incorrectly

reading a sign changed in only a minor manner. Other objectives could be to cause less traffic down a given road, or to direct the passengers to the wrong destination.

4. **Speech recognition** systems that transcribe a given audio sequence to text have achieved human-level performance [132].

   There is a significant threat in this space, as increasingly voice commands are used for controlling users' devices. If an adversary could construct audio that sounded like a benign phrase, but is transcribed as a different command to the device, an adversary could use this to control others devices' without their knowledge.

When applied to neural networks, such evasion attacks are known as *adversarial examples*, and were first observed in the space of image classification [125].

As neural networks are used in an increasingly large number of domains, we must consider their security more seriously.

1.2 Background

1.2.1 Neural Networks

A neural network is a function $F(x) = y$ that accepts an input $x \in \mathbb{R}^n$ and produces an output $y \in \mathbb{R}^m$. F also implicitly depends on some model parameters θ; in our work the model is fixed, so for convenience we don't show the dependence on θ.

We focus exclusively on neural networks used as an m-class classifier. The output of the network is computed using the softmax function, which ensures that the output vector y satisfies $0 \leq y_i \leq 1$ and $y_1 + \cdots + y_m = 1$. The output vector y is thus treated as a probability distribution, i.e., y_i is treated as the probability that input x has class i. The classifier assigns the label $C(x) = \arg\max_i F(x)_i$ to the input x. Let $C^*(x)$ be the correct label of x. The inputs to the softmax function are called *logits*.

Define F to be the full neural network including the softmax function, $Z(x) = z$ to be the output of all layers except the softmax (so z are the logits), and

$$F(x) = \text{softmax}(Z(x)) = y.$$

A neural network consists of layers

$$F = \text{softmax} \circ F_n \circ F_{n-1} \circ \cdots \circ F_1$$

where

$$F_i(x) = \sigma(\theta_i \cdot x) + \hat{\theta}_i$$

for some non-linear activation function σ, some matrix θ_i of model weights, and some vector $\hat{\theta}_i$ of model biases. Together θ and $\hat{\theta}$ make up the model parameters. Common choices of σ are tanh [84], sigmoid, ReLU [76], or ELU [23]. In this dissertation we focus primarily on networks that use a ReLU activation function, as it currently is the most widely used activation function [124, 122, 84, 103].

Image Representation. An $h \times w$-pixel grey-scale image is a two-dimensional vector $x \in \mathbb{R}^{hw}$, where x_i denotes the intensity of pixel i and is scaled to be in the range $[0, 1]$. [1] A color RGB image is a three-dimensional vector $x \in \mathbb{R}^{3hw}$. We do not convert RGB images to HSV, HSL, or other cylindrical coordinate representations of color images: the neural networks act on raw pixel values.

Audio Representation. An audio waveform is represented in one of two ways, depending on the context.

1. **Raw encoding:** the input is represented as a vector in $\mathbb{Z}^{sd}$ where s is the sample rate (in samples per second) and d is the duration (in seconds). The samples are typically within the range $[-2^{15}, 2^{15} - 1]$.

2. **MFCC encoding:** instead of encoding the raw audio, the audio is preprocessed using an encoding called the MFCC (described in Section 4.1) that transforms the input to a vector in $\mathbb{Z}^{mdf}$ where m is the dimensionality of an MFCC vector, d is the duration (in seconds), and f is the number of MFCC vectors per second.

Malware Representation. We represent malware by a collection of discrete feature vectors extracted both statically from the executable and dynamically from simple dynamic analysis. These features include system calls issued (with arguments and return values), files accessed (read, written, moved, or deleted), and all network traffic. We describe our feature extraction further, later.

[1] In practice, each pixel must be one of 256 discrete values chosen among $\{0, \frac{1}{256}, \frac{2}{256}, \ldots, 1\}$. However we often omit this requirement for simplicity.

1.3 Related Work

1.3.1 Evasion Attacks

As mentioned earlier, one of the first instances of evasion attacks on machine learning classifiers was in the case of spam. There has been a significant amount os research on simple classifiers to detect spam, and the attacks that an adversary must perform to defeat them. [7, 94, 130, 74, 26] Most work in this space relies on one of two attacks: (a) inject *good words* [74] into spam to make it appear like legitimate email, or (b) modify the bad words [94] that make an email appear as spam so they are not recognized.

In the space of malware, machine learning has been used to detect malware from benign files [21, 104, 20, 29, 60, 108] (even using neural networks [24]) and an equal amount of work has been performed on attacking such classifiers [75, 134, 89, 92]. And in the space of intrusion detection, there is a similar story of research on detecting intrusions [137, 138, 54, 47] (and again, even using neural networks [71]) and evading intrusion detection [120, 30].

1.3.2 Adversarial Examples

In this dissertation, we focus more narrowly on neural networks used for classification, and evasion attacks on those neural networks.

Szegedy *et al.* were the first to specifically study evasion attacks on image classification neural networks [125], and found that the total amount of change required for the network to shift classification is so small as to be undetectable; they called such images *adversarial examples*.

Specifically, given a valid input x and a target $t \neq C^*(x)$, it is often possible to find a similar input x' such that $C(x') = t$ yet x, x' are close according to some distance metric. An example x' with this property is known as a *targeted* adversarial example.

A less powerful attack also discussed in the literature instead asks for *untargeted* adversarial examples: instead of classifying x as a given target class, we only search for an input x' so that $C(x') \neq C^*(x)$ and x, x' are close.

Following this initial observation, many others began studying different aspects of this phenomenon, from attempting to understand why adversarial examples exist [34], to generating images that look completely random but are strongly classified as any given target image [96], to generating better attacks and constructing defenses

(discussed in the following sections).

1.3.3 Finding Adversarial Examples

There are many existing algorithms for constructing adversarial examples. We review these algorithms and discuss their strengths and weaknesses.

1.3.4 L-BFGS

Szegedy *et al.* [125] generated adversarial examples using box-constrained L-BFGS. Given an image x, their method finds a different image x' that is similar to x under L_2 distance, yet is labeled differently by the classifier. They model the problem as a constrained minimization problem:

$$\begin{aligned} \text{minimize} \quad & \|x - x'\|_2^2 \\ \text{such that} \quad & C(x') = l \\ & x' \in [0,1]^n \end{aligned}$$

This problem can be very difficult to solve, however, so Szegedy *et al.* instead solve the following problem:

$$\begin{aligned} \text{minimize} \quad & c \cdot \|x - x'\|_2^2 + \text{loss}_{F,l}(x') \\ \text{such that} \quad & x' \in [0,1]^n \end{aligned}$$

where $\text{loss}_{F,l}$ is a function mapping an image to a positive real number. One common loss function to use is cross-entropy. Line search is performed to find the constant $c > 0$ that yields an adversarial example of minimum distance: in other words, we repeatedly solve this optimization problem for multiple values of c, adaptively updating c using bisection search or any other method for one-dimensional optimization.

1.3.5 Fast Gradient Sign

The fast gradient sign [34] method has two key differences from the L-BFGS method: first, it is optimized for the L_∞ distance metric, and second, it is designed primarily to be fast instead of producing very close adversarial examples. Given an image x the fast gradient sign method sets

$$x' = x - \epsilon \cdot \text{sign}(\nabla \text{loss}_{F,t}(x)),$$

where ϵ is chosen to be sufficiently small so as to be undetectable, and t is the target label. Intuitively, for each pixel, the fast gradient sign method uses the gradient of the loss function to determine in which direction the pixel's intensity should be changed (whether it should be increased or decreased) to minimize the loss function; then, it shifts all pixels simultaneously.

It is important to note that the fast gradient sign attack was designed to be *fast*, rather than optimal. It is not meant to produce the minimal adversarial perturbations.

Iterative Gradient Sign Kurakin *et al.* introduce a simple refinement of the fast gradient sign method [66] where instead of taking a single step of size ϵ in the direction of the gradient-sign, multiple smaller steps α are taken, and the result is clipped by the same ϵ. Specifically, begin by setting

$$x'_0 = 0$$

and then on each iteration

$$x'_i = x'_{i-1} - \text{clip}_\epsilon(\alpha \cdot \text{sign}(\nabla \text{loss}_{F,t}(x'_{i-1})))$$

Iterative gradient sign was found to produce superior results to fast gradient sign [66].

1.3.6 JSMA

Papernot *et al.* introduced an attack optimized under L_0 distance [102] known as the Jacobian-based Saliency Map Attack (JSMA). We give a brief summary of their attack algorithm; for a complete description and motivation, we encourage the reader to read their original paper [102].

At a high level, the attack is a greedy algorithm that picks pixels to modify one at a time, increasing the target classification on each iteration. They use the gradient $\nabla Z(x)_l$ to compute a *saliency map*, which models the impact each pixel has on the resulting classification. A large value indicates that changing it will significantly increase the likelihood of the model labeling the image as the target class l. Given the saliency map, it picks the most important pixel and modify it to increase the likelihood of class l. This is repeated until either more than a set threshold of pixels are modified which makes the attack detectable, or it succeeds in changing the classification.

In more detail, we begin by defining the saliency map in terms of a pair of pixels p, q.

Define

$$\alpha_{pq} = \sum_{i \in \{p,q\}} \frac{\partial Z(x)_t}{\partial x_i}$$

$$\beta_{pq} = \left(\sum_{i \in \{p,q\}} \sum_{j} \frac{\partial Z(x)_j}{\partial x_i} \right) - \alpha_{pq}$$

so that α_{pq} represents how much changing both pixels p and q will change the target classification, and β_{pq} represents how much changing p and q will change all other outputs. Then the algorithm picks

$$(p^*, q^*) = \arg\max_{(p,q)} (-\alpha_{pq} \cdot \beta_{pq}) \cdot (\alpha_{pq} > 0) \cdot (\beta_{pq} < 0)$$

so that $\alpha_{pq} > 0$ (the target class is more likely), $\beta_{pq} < 0$ (the other classes become less likely), and $-\alpha_{pq} \cdot \beta_{pq}$ is largest.

Notice that JSMA uses the output of the second-to-last layer Z, the logits, in the calculation of the gradient: the output of the softmax F is *not* used. We refer to this as the **JSMA-Z** attack.

Sometimes, the computation uses the output of the softmax (F) instead of the logits (Z). We refer to this modification as the **JSMA-F** attack.

When an image has multiple color channels (e.g., RGB), this attack considers the L_0 difference to be 1 for each color channel changed independently (so that if all three color channels of one pixel change change, the L_0 norm would be 3). While we do not believe this is a meaningful threat model, when comparing to this attack, we evaluate under both models.

1.3.7 Deepfool

Deepfool [87] is an untargeted attack technique optimized for the L_2 distance metric. It is efficient and produces closer adversarial examples than the L-BFGS approach discussed earlier.

The authors construct Deepfool by imagining that the neural networks are totally linear, with a hyperplane separating each class from another. From this, they analytically derive the optimal solution to this simplified problem, and construct the adversarial example.

Then, since neural networks are not actually linear, they take a step towards that solution, and repeat the process a second time. The search terminates when a true adversarial example is found.

The exact formulation used is rather sophisticated; interested readers should refer to the original work [87].

1.3.8 Defending Against Adversarial Examples

There have been many defenses proposed for adversarial examples. In this dissertation, we evaluate many and break most of these.

1.3.9 Poisoning Attacks

In contrast to evasion attacks (where we are given a model and attempt to construct instances that evade the model), a poisoning attack controls some small fraction of the *training* data, and attempts to use this to cause a specific testing instance to be misclassified.

The first work done in this space is due to Kearns and Li [57] who study poisoning attacks on a model known as Probably Approximately Correct (PAC) due to Valiant [127]. They formally study this space and prove bounds by which an adversary is able to influence classification accuracy.

Nelson and Joseph [95] study the problem of poisoning in the space of a anomaly detection system. Given a set of points in $n-$dimensional space, they consider an anomaly detection system as a simple hypersphere where points interior to the sphere are normal, and exterior are anomalous. In this domain it is possible to prove bounds on the adversary.

A problem related to this has been studied in the case of malware signature generation by Perdisci *et al.* [106] who show it is possible to construct malware samples so that if one controls 50% of the training data, it is possible to make the signature generator fail.

Following this, Biggio *et al.* [12, 13] study the poisoning problem on support vector machines (SVMs). By looking at both random training-data label flipping and adversarial training-data label flipping they find that an adversary must control a relatively large fraction of the training data (about 20%) to cause a significant drop in accuracy. In their followup study, they consider the specific case of causing a single instance to be misclassified and develop an algorithm to do this effectively. SVMs have also been analyzed by others [81] to similar effect.

Chapter 2

Evaluating the Robustness of Neural Networks

In order to construct robust defenses for neural networks, we must first be able to effectively evaluate the robustness of any given neural network. There are two ways in which security is argued in any domain:

1. **A proof of robustness** formally argues why the proposed defense is secure under some given threat model for some specific class of attacks. If the proof is sound, no attack can do better than the provided lower bound.
2. **An empirical robustness analysis** takes the given defense and attacks it. The best attack found is an upper bound on the robustness of the system. Stronger attacks may exist, but the defense can be no more secure than the best attack found.

In general, it is much more difficult to construct proofs of robustness than empirical robustness arguments. In some domains (e.g., cryptography) this is possible with very restrictive threat models. However, for other areas, it is often impractical to construct such an argument: either the set of required assumptions is so large as produce a weak argument, or the set of assumptions is sufficiently small but the guaranteed security is minimal.

On the other hand, empirical robustness arguments are a useful method of quickly ruling out defenses. As soon as a defense fails against one concrete attack, it can be dismissed as a strong defense to all attacks.

In this section, we construct a framework for empirically evaluating the robustness of neural networks by attacking them.

2.1 Background

2.1.1 Adversarial Examples

As mentioned earlier, if we are given a valid input x and a target $t \neq C^*(x)$, it is often possible to find a similar input x' such that $C(x') = t$ yet x, x' are close according to some distance metric. An example x' with this property is known as a *targeted* adversarial example.

A less powerful attack also discussed in the literature instead asks for *untargeted* adversarial examples: instead of classifying x as a given target class, we only search for an input x' so that $C(x') \neq C^*(x)$ and x, x' are close. Untargeted attacks are strictly less powerful than targeted attacks and we do not consider them here. [1]

Instead, we consider three different approaches for how to choose the target class, in a targeted attack:

- *Average Case:* select the target class *uniformly at random* among the labels that are not the correct label.
- *Best Case:* perform the attack against all incorrect classes, and report the target class that was *least difficult* to attack.
- *Worst Case:* perform the attack against all incorrect classes, and report the target class that was *most difficult* to attack.

In all of our evaluations we perform all three types of attacks: best-case, average-case, and worst-case. Notice that if a classifier is only accurate 80% of the time, then the best case attack will require a change of 0 in 20% of cases.

On ImageNet, we approximate the best-case and worst-case attack by sampling 100 random target classes out of the 1,000 possible for efficiency reasons.

2.1.2 Distance Metrics

In our definition of adversarial examples, we require use of a distance metric to quantify similarity. There are three widely-used distance metrics in the literature for generating adversarial examples, all of which are L_p norms.

[1] An untargeted attack is simply a more efficient (and often less accurate) method of running a targeted attack for each target and taking the closest. We focus on identifying the most accurate attacks, and do not consider untargeted attacks.

The L_p distance is written $\|x - x'\|_p$, where the p-norm $\|\cdot\|_p$ is defined as

$$\|v\|_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}}.$$

In more detail:

1. L_0 distance measures the number of coordinates i such that $x_i \neq x'_i$. Thus, the L_0 distance corresponds to the number of pixels that have been altered in an image.[2]

   Papernot *et al.* argue for the use of the L_0 distance metric, and it is the primary distance metric under which defensive distillation's security is argued [103].

2. L_2 distance measures the standard Euclidean (root-mean-square) distance between x and x'. The L_2 distance can remain small when there are many small changes to many pixels.

   This distance metric was used in the initial adversarial example work [125].

3. L_∞ distance measures the maximum change to any of the coordinates:

   $$\|x - x'\|_\infty = \max(|x_1 - x'_1|, \ldots, |x_n - x'_n|).$$

   For images, we can imagine there is a maximum budget, and each pixel is allowed to be changed by up to this limit, with no limit on the number of pixels that are modified.

   Goodfellow *et al.* argue that L_∞ is the optimal distance metric to use [129] and in a follow-up paper Papernot *et al.* argue distillation is secure under this distance metric [99].

No distance metric is a perfect measure of human perceptual similarity, and we pass no judgement on exactly which distance metric is optimal. We believe constructing and evaluating a good distance metric is an important research question we leave to future work.

However, since most existing work has picked one of these three distance metrics, and since defensive distillation argued security against two of these, we too use these

[2] In RGB images, there are three channels that each can change. We count the number of *pixels* that are different, where two pixels are considered different if *any* of the three colors are different. We do not consider a distance metric where an attacker can change one color plane but not another meaningful. We relax this requirement when comparing to other L_0 attacks that do not make this assumption to provide for a fair comparison.

Layer Type	MNIST Model	CIFAR Model
Convolution + ReLU	3×3×32	3×3×64
Convolution + ReLU	3×3×32	3×3×64
Max Pooling	2×2	2×2
Convolution + ReLU	3×3×64	3×3×128
Convolution + ReLU	3×3×64	3×3×128
Max Pooling	2×2	2×2
Fully Connected + ReLU	200	256
Fully Connected + ReLU	200	256
Softmax	10	10

Table 2.1: Model architectures for the MNIST and CIFAR models. This architecture is identical to that of the original defensive distillation work. [103]

Parameter	MNIST Model	CIFAR Model
Learning Rate	0.1	0.01 (decay 0.5)
Learning Rate	0.9	0.9 (decay 0.5)
Delay Decay	-	10 epochs
Batch Size	128	128
Epochs	50	50

Table 2.2: Model parameters for the MNIST and CIFAR models. These parameters are identical to that of the original defensive distillation work. [103]

distance metrics and construct attacks that perform superior to the state-of-the-art for each of these distance metrics.

When reporting all numbers in this paper, we report using the distance metric as defined above, on the range $[0, 1]$. (That is, changing a pixel in a greyscale image from full-on to full-off will result in L_2 change of 1.0 and a L_∞ change of 1.0, not 255.)

2.2 Experimental Setup

Before we develop our attack algorithms to break distillation, we describe how we train the models on which we will evaluate our attacks.

We train two networks for the MNIST [69] and CIFAR-10 [65] classification tasks, and

use one pre-trained network for the ImageNet classification task [111]. Our models and training approaches are identical to those presented in [103]. We achieve 99.5% accuracy on MNIST, comparable to the state of the art. On CIFAR-10, we achieve 85% accuracy, nearly identical to the accuracy given in the distillation work. [3]

MNIST The model architecture is given in Table 2.1 and the hyperparameters selected in Table 2.2. We use a momentum-based SGD optimizer. We perform no data-set augmentation.

CIFAR-10 The model architecture is given in Table 2.1 and the hyperparameters selected in Table 2.2. We use a momentum-based SGD optimizer.

We perform data-set augmentation to increase the accuracy. We randomly crop a 24x24 region out of the 32x32 images, and whiten this crop to have mean value 0 and standard deviation 1. We additionally randomly flip images left/right, randomly adjust the contrast, and randomly adjust the brightness of each image. This achieves 85% accuracy on the test set.

ImageNet Along with considering MNIST and CIFAR, which are both relatively small datasets, we also consider the ImageNet dataset. Instead of training our own ImageNet model, we use the pre-trained Inception v3 network [124], which achieves 96% top-5 accuracy. Inception takes images as $299 \times 299 \times 3$ dimensional vectors.

2.3 Our Approach

We now turn to our approach for constructing adversarial examples. To begin, we rely on the initial formulation of adversarial examples [125] and formally define the problem of finding an adversarial instance for an image x as follows:

$$\begin{aligned} \text{minimize} \quad & \mathcal{D}(x, x+\delta) \\ \text{such that} \quad & C(x+\delta) = t \\ & x+\delta \in [0,1]^n \end{aligned}$$

where x is fixed, and the goal is to find δ that minimizes $\mathcal{D}(x, x+\delta)$. That is, we want to find some small change δ that we can make to an image x that will change

[3] This is compared to the state-of-the-art result of 95% [35, 122, 84]. However, in order to provide the most accurate comparison to the original work, we feel it is important to reproduce their model architectures.

its classification, but so that the result is still a valid image. Here $\mathcal{D}$ is some distance metric; for us, it will be either L_0, L_2, or L_∞ as discussed earlier.

We solve this problem by formulating it as an appropriate optimization instance that can be solved by existing optimization algorithms. There are many possible ways to do this; we explore the space of formulations and empirically identify which ones lead to the most effective attacks.

2.3.1 Objective Function

The above formulation is difficult for existing algorithms to solve directly, as the constraint $C(x+\delta) = t$ is highly non-linear. Therefore, we express it in a different form that is better suited for optimization. We define an objective function f such that $C(x+\delta) = t$ if and only if $f(x+\delta) \leq 0$. There are many possible choices for f:

$$\begin{aligned}
f_1(x') &= -\text{loss}_{F,t}(x') + 1 \\
f_2(x') &= (\max_{i \neq t}(F(x')_i) - F(x')_t)^+ \\
f_3(x') &= \text{softplus}(\max_{i \neq t}(F(x')_i) - F(x')_t) - \log(2) \\
f_4(x') &= (0.5 - F(x')_t)^+ \\
f_5(x') &= -\log(2F(x')_t - 2) \\
f_6(x') &= (\max_{i \neq t}(Z(x')_i) - Z(x')_t)^+ \\
f_7(x') &= \text{softplus}(\max_{i \neq t}(Z(x')_i) - Z(x')_t) - \log(2)
\end{aligned}$$

where s is the correct classification, $(e)^+$ is short-hand for $\max(e, 0)$, $\text{softplus}(x) = \log(1 + \exp(x))$, and $\text{loss}_{F,s}(x)$ is the cross entropy loss for x.

Notice that we have adjusted some of the above formula by adding a constant; we have done this only so that the function respects our definition. This does not impact the final result, as it just scales the minimization function.

Now, instead of formulating the problem as

$$\begin{aligned}
\text{minimize} \quad & \mathcal{D}(x, x+\delta) \\
\text{such that} \quad & f(x+\delta) \leq 0 \\
& x + \delta \in [0,1]^n
\end{aligned}$$

we use the alternative formulation:

$$\begin{aligned}
\text{minimize} \quad & \mathcal{D}(x, x+\delta) + c \cdot f(x+\delta) \\
\text{such that} \quad & x + \delta \in [0,1]^n
\end{aligned}$$

	Best Case						Average Case						Worst Case					
	Change of Variable		Clipped Descent		Projected Descent		Change of Variable		Clipped Descent		Projected Descent		Change of Variable		Clipped Descent		Projected Descent	
	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob
f_1	2.46	100%	2.93	100%	2.31	100%	4.35	100%	5.21	100%	4.11	100%	7.76	100%	9.48	100%	7.37	100%
f_2	4.55	80%	3.97	83%	3.49	83%	3.22	44%	8.99	63%	15.06	74%	2.93	18%	10.22	40%	18.90	53%
f_3	4.54	77%	4.07	81%	3.76	82%	3.47	44%	9.55	63%	15.84	74%	3.09	17%	11.91	41%	24.01	59%
f_4	5.01	86%	6.52	100%	7.53	100%	4.03	55%	7.49	71%	7.60	71%	3.55	24%	4.25	35%	4.10	35%
f_5	1.97	100%	2.20	100%	1.94	100%	3.58	100%	4.20	100%	3.47	100%	6.42	100%	7.86	100%	6.12	100%
f_6	1.94	100%	2.18	100%	1.95	100%	3.47	100%	4.11	100%	3.41	100%	6.03	100%	7.50	100%	5.89	100%
f_7	1.96	100%	2.21	100%	1.94	100%	3.53	100%	4.14	100%	3.43	100%	6.20	100%	7.57	100%	5.94	100%

Table 2.3: Evaluation of all combinations of one of the seven possible objective functions with one of the three box constraint encodings. We show the average L_2 distortion, the standard deviation, and the success probability (fraction of instances for which an adversarial example can be found). Evaluated on 1000 random instances. When the success is not 100%, mean is for successful attacks only.

where $c > 0$ is a suitably chosen constant. These two are equivalent, in the sense that there exists $c > 0$ such that the optimal solution to the latter matches the optimal solution to the former. After instantiating the distance metric $\mathcal{D}$ with an l_p norm, the problem becomes: given x, find δ that solves

$$\begin{aligned} &\text{minimize} \quad \|\delta\|_p + c \cdot f(x+\delta) \\ &\text{such that} \quad x + \delta \in [0,1]^n \end{aligned}$$

Choosing the constant c.

Empirically, we have found that often the best way to choose c is to use the smallest value of c for which the resulting solution x^* has $f(x^*) \leq 0$. This causes gradient descent to minimize both of the terms simultaneously instead of picking only one to optimize over first.

We verify this by running our f_6 formulation (which we found most effective) for values of c spaced uniformly (on a log scale) from $c = 0.01$ to $c = 100$ on the MNIST dataset. We plot this line in Figure 2.1. [4]

Further, we have found that if choose the smallest c such that $f(x^*) \leq 0$, the solution is within 5% of optimal 70% of the time, and within 30% of optimal 98% of the time, where "optimal" refers to the solution found using the best value of c. Therefore, in our implementations we use modified binary search to choose c.

[4]The corresponding figures for other objective functions are similar; we omit them for brevity.

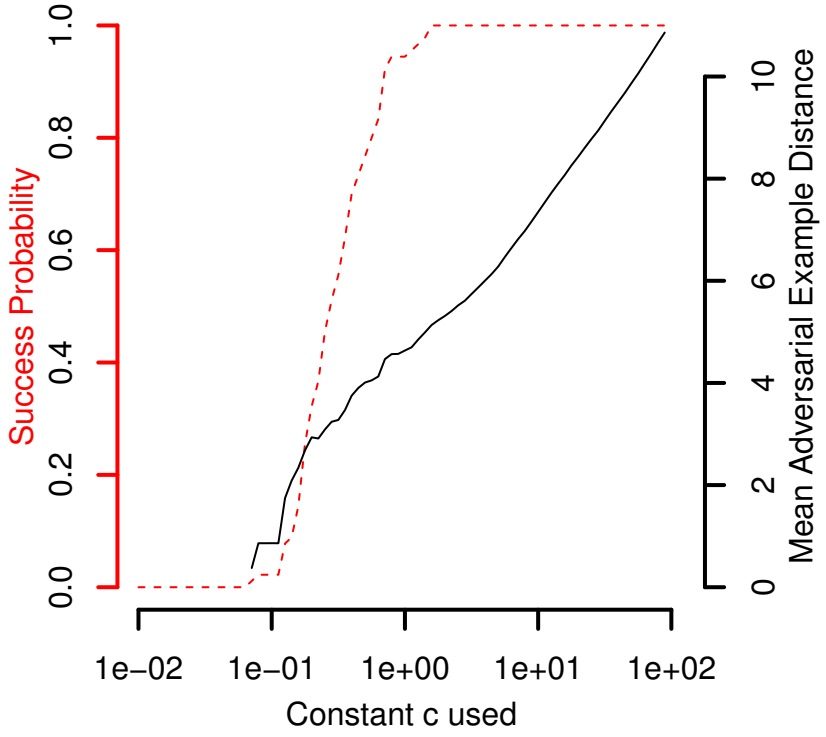

Figure 2.1: Sensitivity on the constant c. We plot the L_2 distance of the adversarial example computed by gradient descent as a function of c, for objective function f_6. When $c < .1$, the attack rarely succeeds. After $c > 1$, the attack becomes less effective, but always succeeds.

2.3.2 Box constraints

To ensure the modification yields a valid image, we have a constraint on δ: we must have $0 \leq x_i + \delta_i \leq 1$ for all i. In the optimization literature, this is known as a "box constraint." Previous work uses a particular optimization algorithm, L-BFGS-B, which supports box constraints natively.

We investigate three different methods of approaching this problem.

1. *Projected gradient descent* performs one step of standard gradient descent, and then clips all the coordinates to be within the box.

   This approach can work poorly for gradient descent approaches that have a complicated update step (for example, those with momentum): when we clip the actual x_i, we unexpectedly change the input to the next iteration of the algorithm.

2. *Clipped gradient descent* does not clip x_i on each iteration; rather, it incorporates the clipping into the objective function to be minimized. In other words, we replace $f(x+\delta)$ with $f(\min(\max(x+\delta, 0), 1))$, with the min and max taken component-wise.

   While solving the main issue with projected gradient descent, clipping introduces a new problem: the algorithm can get stuck in a flat spot where it has increased some component x_i to be substantially larger than the maximum allowed. When this happens, the partial derivative becomes zero, so even if some improvement is possible by later reducing x_i, gradient descent has no way to detect this.

3. *Change of variables* introduces a new variable w and instead of optimizing over the variable δ defined above, we apply a change-of-variables and optimize over w, setting
   $$\delta_i = \frac{1}{2}(\tanh(w_i) + 1) - x_i.$$
   Since $-1 \leq \tanh(w_i) \leq 1$, it follows that $0 \leq x_i + \delta_i \leq 1$, so the solution will automatically be valid. [5]

   We can think of this approach as a smoothing of clipped gradient descent that eliminates the problem of getting stuck in extreme regions.

These methods allow us to use other optimization algorithms that don't natively support box constraints. We use the Adam [61] optimizer almost exclusively, as we

[5] Instead of scaling by $\frac{1}{2}$ we scale by $\frac{1}{2} + \epsilon$ to avoid dividing by zero.

have found it to be the most effective at quickly finding adversarial examples. We tried three solvers — standard gradient descent, gradient descent with momentum, and Adam — and all three produced identical-quality solutions. However, Adam converges substantially more quickly than the others.

2.3.3 Evaluation of approaches

For each possible objective function $f(\cdot)$ and method to enforce the box constraint, we evaluate the quality of the adversarial examples found.

To choose the optimal c, we perform 20 iterations of binary search over c. For each selected value of c, we run $10,000$ iterations of gradient descent with the Adam optimizer. [6]

The results of this analysis are in Table 2.3. We evaluate the quality of the adversarial examples found on the MNIST and CIFAR datasets. The relative ordering of each objective function is identical between the two datasets, so for brevity we report only results for MNIST.

There is a factor of three difference in quality between the best objective function and the worst. The choice of method for handling box constraints does not impact the quality of results as significantly for the best minimization functions.

In fact, the worst performing objective function, cross entropy loss, is the approach that was most suggested in the literature previously [125, 114].

Why are some loss functions better than others? When $c = 0$, gradient descent will not make any move away from the initial image. However, a large c often causes the initial steps of gradient descent to perform in an overly-greedy manner, only traveling in the direction which can most easily reduce f and ignoring the $\mathcal{D}$ loss — thus causing gradient descent to find sub-optimal solutions.

This means that for loss function f_1 and f_4, there is no good constant c that is useful throughout the duration of the gradient descent search. Since the constant c weights the relative importance of the distance term and the loss term, in order for a fixed constant c to be useful, the relative value of these two terms should remain approximately equal. This is not the case for these two loss functions.

To explain why this is the case, we will have to take a side discussion to analyze how adversarial examples exist. Consider a valid input x and an adversarial example x'

[6] Adam converges to 95% of optimum within $1,000$ iterations 92% of the time. For completeness we run it for $10,000$ iterations at each step.

on a network.

What does it look like as we linearly interpolate from x to x'? That is, let $y = \alpha x + (1-\alpha)x'$ for $\alpha \in [0,1]$. It turns out the value of $Z(\cdot)_t$ is mostly linear from the input to the adversarial example, and therefore the $F(\cdot)_t$ is a logistic. We verify this fact empirically by constructing adversarial examples on the first $1,000$ test images on both the MNIST and CIFAR dataset with our approach, and find the Pearson correlation coefficient $r > .9$.

Given this, consider loss function f_4 (the argument for f_1 is similar). In order for the gradient descent attack to make any change initially, the constant c will have to be large enough that

$$\epsilon < c(f_1(x+\epsilon) - f_1(x))$$

or, as $\epsilon \to 0$,

$$1/c < |\nabla f_1(x)|$$

implying that c must be larger than the inverse of the gradient to make progress, but the gradient of f_1 is identical to $F(\cdot)_t$ so will be tiny around the initial image, meaning c will have to be extremely large.

However, as soon as we leave the immediate vicinity of the initial image, the gradient of $\nabla f_1(x+\delta)$ increases at an exponential rate, making the large constant c cause gradient descent to perform in an overly greedy manner.

We verify all of this theory empirically. When we run our attack trying constants chosen from 10^{-10} to 10^{10} the average constant for loss function f_4 was 10^6.

The average gradient of the loss function f_1 around the valid image is 2^{-20} but 2^{-1} at the closest adversarial example. This means c is a million times larger than it has to be, causing the loss function f_4 and f_1 to perform worse than any of the others.

2.3.4 Discretization

We model pixel intensities as a (continuous) real number in the range $[0,1]$. However, in a valid image, each pixel intensity must be a (discrete) integer in the range $\{0,1,\ldots,255\}$. This additional requirement is not captured in our formulation. In practice, we ignore the integrality constraints, solve the continuous optimization problem, and then round to the nearest integer: the intensity of the ith pixel becomes $\lfloor 255(x_i+\delta_i) \rceil$.

This rounding will slightly degrade the quality of the adversarial example. If we need to restore the attack quality, we perform greedy search on the lattice defined by the

discrete solutions by changing one pixel value at a time. This greedy search never failed for any of our attacks.

Prior work has largely ignored the integrality constraints.[7] For instance, when using the fast gradient sign attack with $\epsilon = 0.1$ (i.e., changing pixel values by 10%), discretization rarely affects the success rate of the attack. In contrast, in our work, we are able to find attacks that make much smaller changes to the images, so discretization effects cannot be ignored. We take care to always generate valid images; when reporting the success rate of our attacks, they always are for attacks that include the discretization post-processing.

2.4 Our Three Attacks

2.4.1 Our L_2 Attack

Putting these ideas together, we obtain a method for finding adversarial examples that will have low distortion in the L_2 metric. Given x, we choose a target class t (such that we have $t \neq C^*(x)$) and then search for w that solves

$$\text{minimize} \;\; \|\frac{1}{2}(\tanh(w)+1) - x\|_2^2 + c \cdot f(\frac{1}{2}(\tanh(w)+1)$$

with f defined as

$$f(x') = \max(\max\{Z(x')_i : i \neq t\} - Z(x')_t, -\kappa).$$

This f is based on the best objective function found earlier, modified slightly so that we can control the confidence with which the misclassification occurs by adjusting κ. The parameter κ encourages the solver to find an adversarial instance x' that will be classified as class t with high confidence. We set $\kappa = 0$ for our attacks but we note here that a side benefit of this formulation is it allows one to control for the desired confidence. This is discussed further in Section 3.1.1.

Figure 2.2 shows this attack applied to our MNIST model for each source digit and target digit. Almost all attacks are visually indistinguishable from the original digit.

No attack is visually distinguishable from the baseline image on CIFAR.

Multiple starting-point gradient descent. The main problem with gradient descent is that its greedy search is not guaranteed to find the optimal solution and can become

[7] One exception: The JSMA attack [102] handles this by only setting the output value to either 0 or 255.

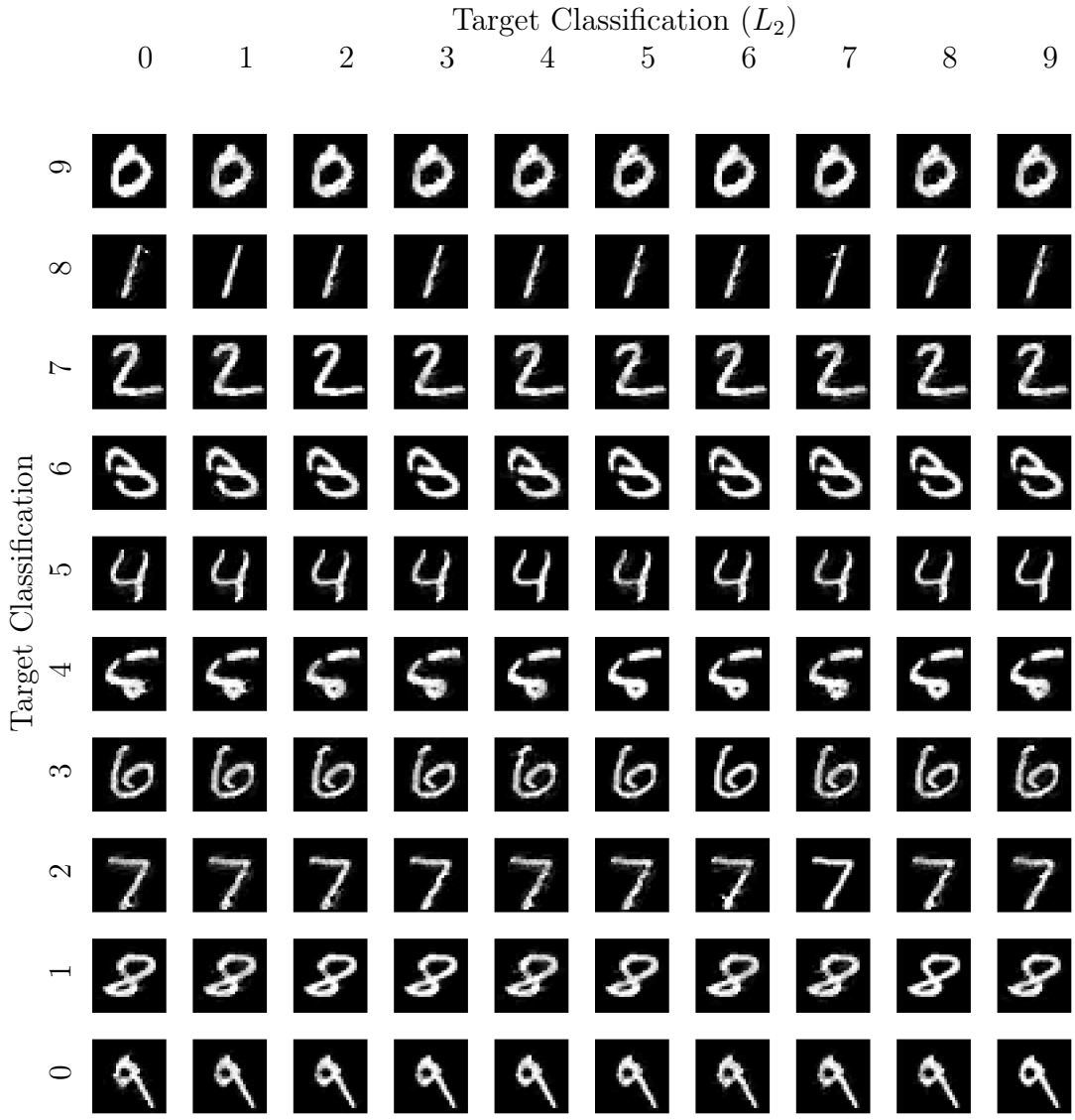

Figure 2.2: Our L_2 adversary applied to the MNIST dataset performing a targeted attack for every source/target pair. Each digit is the first image in the dataset with that label.

stuck in a local minimum. To remedy this, we pick multiple random starting points close to the original image and run gradient descent from each of those points for a fixed number of iterations. We randomly sample points uniformly from the ball of radius r, where r is the closest adversarial example found so far. Starting from multiple starting points reduces the likelihood that gradient descent gets stuck in a bad local minimum.

2.4.2 Our L_0 Attack

The L_0 distance metric is non-differentiable and therefore is ill-suited for standard gradient descent. Instead, we use an iterative algorithm that, in each iteration, identifies some pixels that don't have much effect on the classifier output and then fixes those pixels, so their value will never be changed. The set of fixed pixels grows in each iteration until we have, by process of elimination, identified a minimal (but possibly not minimum) subset of pixels that can be modified to generate an adversarial example. In each iteration, we use our L_2 attack to identify which pixels are unimportant.

In more detail, on each iteration, we call the L_2 adversary, restricted to only modify the pixels in the allowed set. Let δ be the solution returned from the L_2 adversary on input image x, so that $x + \delta$ is an adversarial example. We compute $g = \nabla f(x + \delta)$ (the gradient of the objective function, evaluated at the adversarial instance). We then select the pixel $i = \arg\min_i g_i \cdot \delta_i$ and fix i, i.e., remove i from the allowed set.[8] The intuition is that $g_i \cdot \delta_i$ tells us how much reduction to $f(\cdot)$ we obtain from the ith pixel of the image, when moving from x to $x + \delta$: g_i tells us how much reduction in f we obtain, per unit change to the ith pixel, and we multiply this by how much the ith pixel has changed. This process repeats until the L_2 adversary fails to find an adversarial example.

There is one final detail required to achieve strong results: choosing a constant c to use for the L_2 adversary. To do this, we initially set c to a very low value (e.g., 10^{-4}). We then run our L_2 adversary at this c-value. If it fails, we double c and try again, until it is successful. We abort the search if c exceeds a fixed threshold (e.g., 10^{10}).

JSMA *grows* a set — initially empty — of pixels that are allowed to be changed and sets the pixels to maximize the total loss. In contrast, our attack *shrinks* the set of pixels — initially containing every pixel — that are allowed to be changed.

Our algorithm is significantly more effective than JSMA (see Section 2.5 for an eval-

[8] Selecting the index i that minimizes δ_i is simpler, but it yields results with 1.5× higher L_0 distortion.

uation). It is also efficient: we introduce optimizations that make it about as fast as our L_2 attack with a single starting point on MNIST and CIFAR; it is substantially slower on ImageNet. Instead of starting gradient descent in each iteration from the initial image, we start the gradient descent from the solution found on the previous iteration ("warm-start"). This dramatically reduces the number of rounds of gradient descent needed during each iteration, as the solution with k pixels held constant is often very similar to the solution with $k+1$ pixels held constant.

Figure 2.3 shows the L_0 attack applied to one digit of each source class, targeting each target class, on the MNIST dataset. The attacks are visually noticeable, implying the L_0 attack is more difficult than L_2. Perhaps the worst case is that of a 7 being made to classify as a 6; interestingly, this attack for L_2 is one of the only visually distinguishable attacks. Almost no differences are noticeable for CIFAR.

2.4.3 Our L_∞ Attack

The L_∞ distance metric is not fully differentiable and standard gradient descent does not perform well for it. We experimented with naively optimizing

$$\text{minimize} \quad c \cdot f(x+\delta) + \|\delta\|_\infty$$

However, we found that gradient descent produces very poor results: the $\|\delta\|_\infty$ term only penalizes the largest (in absolute value) entry in δ and has no impact on any of the other. As such, gradient descent very quickly becomes stuck oscillating between two suboptimal solutions. Consider a case where $\delta_i = 0.5$ and $\delta_j = 0.5 - \epsilon$. The L_∞ norm will only penalize δ_i, not δ_j, and $\frac{\partial}{\partial \delta_j}\|\delta\|_\infty$ will be zero at this point. Thus, the gradient imposes no penalty for increasing δ_j, even though it is already large. On the next iteration we might move to a position where δ_j is slightly larger than δ_i, say $\delta_i = 0.5 - \epsilon'$ and $\delta_j = 0.5 + \epsilon''$, a mirror image of where we started. In other words, gradient descent may oscillate back and forth across the line $\delta_i = \delta_j = 0.5$, making it nearly impossible to make progress.

We resolve this issue using an iterative attack. We replace the L_2 term in the objective function with a penalty for any terms that exceed τ (initially 1, decreasing in each iteration). This prevents oscillation, as this loss term penalizes all large values simultaneously. Specifically, in each iteration we solve

$$\text{minimize} \quad c \cdot f(x+\delta) + \cdot \sum_i \left[(\delta_i - \tau)^+\right]$$

After each iteration, if $\delta_i < \tau$ for all i, we reduce τ by a factor of 0.9 and repeat; otherwise, we terminate the search.

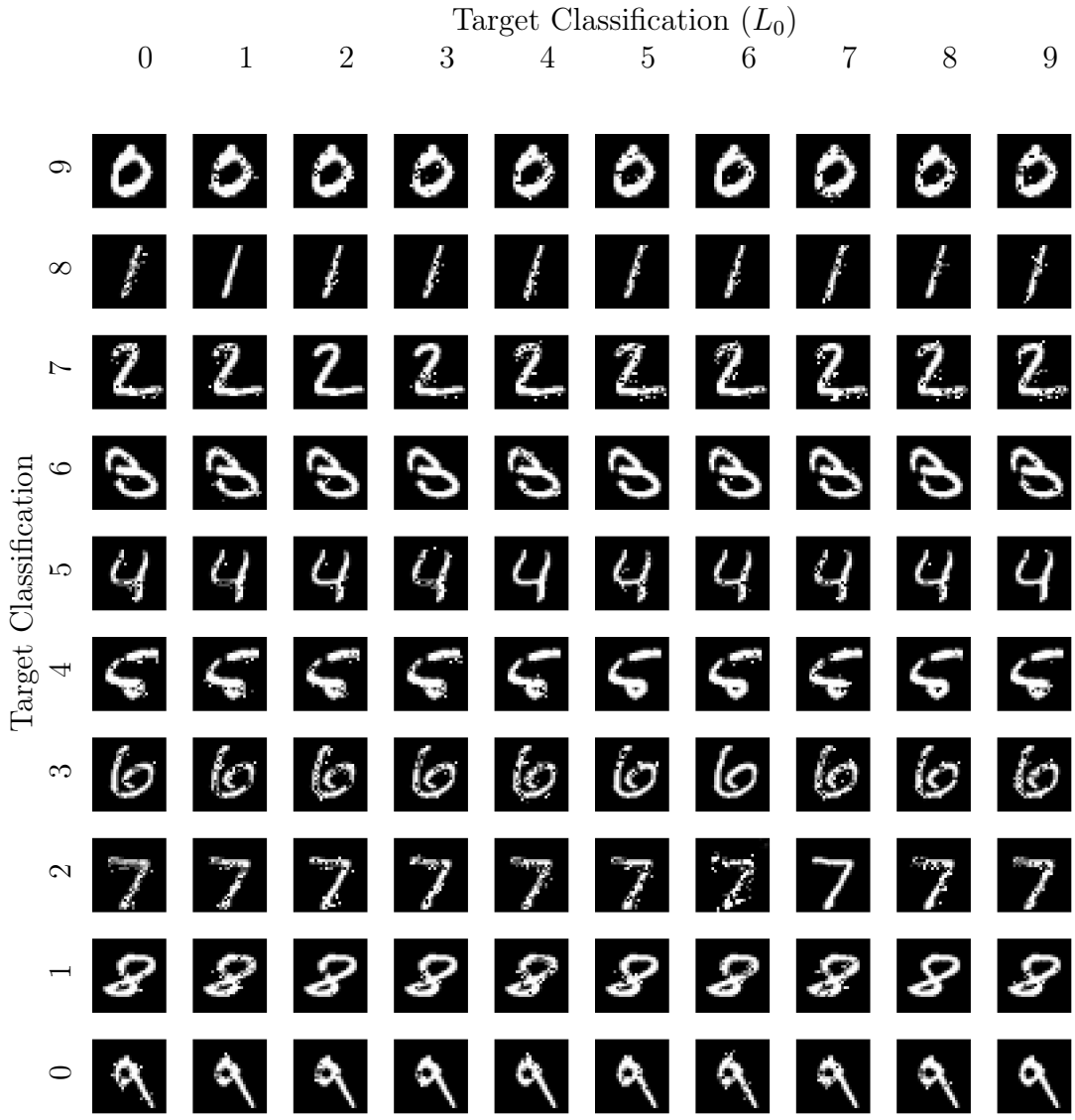

Figure 2.3: Our L_0 adversary applied to the MNIST dataset performing a targeted attack for every source/target pair. Each digit is the first image in the dataset with that label.

	Best Case				Average Case				Worst Case			
	MNIST		CIFAR		MNIST		CIFAR		MNIST		CIFAR	
	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob
Our L_0	8.5	100%	5.9	100%	16	100%	13	100%	33	100%	24	100%
JSMA-Z	20	100%	20	100%	56	100%	58	100%	180	98%	150	100%
JSMA-F	17	100%	25	100%	45	100%	110	100%	100	100%	240	100%
Our L_2	1.36	100%	0.17	100%	1.76	100%	0.33	100%	2.60	100%	0.51	100%
Deepfool	2.11	100%	0.85	100%	–	-	–	-	–	-	–	-
Our L_∞	0.13	100%	0.0092	100%	0.16	100%	0.013	100%	0.23	100%	0.019	100%
Fast Gradient Sign	0.22	100%	0.015	99%	0.26	42%	0.029	51%	–	0%	0.34	1%
Iterative Gradient Sign	0.14	100%	0.0078	100%	0.19	100%	0.014	100%	0.26	100%	0.023	100%

Table 2.4: Comparison of the three variants of targeted attack to previous work for our MNIST and CIFAR models. When success rate is not 100%, the mean is only over successes.

Again we must choose a good constant c to use for the L_∞ adversary. We take the same approach as we do for the L_0 attack: initially set c to a very low value and run the L_∞ adversary at this c-value. If it fails, we double c and try again, until it is successful. We abort the search if c exceeds a fixed threshold.

Using "warm-start" for gradient descent in each iteration, this algorithm is about as fast as our L_2 algorithm (with a single starting point).

Figure 2.4 shows the L_∞ attack applied to one digit of each source class, targeting each target class, on the MNSIT dataset. While most differences are not visually noticeable, a few are. Again, the worst case is that of a 7 being made to classify as a 6.

No attack is visually distinguishable from the baseline image on CIFAR.

2.5 Attack Evaluation

We compare our targeted attacks to the best results previously reported in prior publications, for each of the three distance metrics.

We re-implement Deepfool, fast gradient sign, and iterative gradient sign. For fast gradient sign, we search over ϵ to find the smallest distance that generates an adversarial example; failures is returned if no ϵ produces the target class. Our iterative gradient sign method is similar: we search over ϵ (fixing $\alpha = \frac{1}{256}$) and return the smallest successful.

For JSMA we use the implementation in CleverHans [98] with only slight modification (we improve performance by 50× with no impact on accuracy).

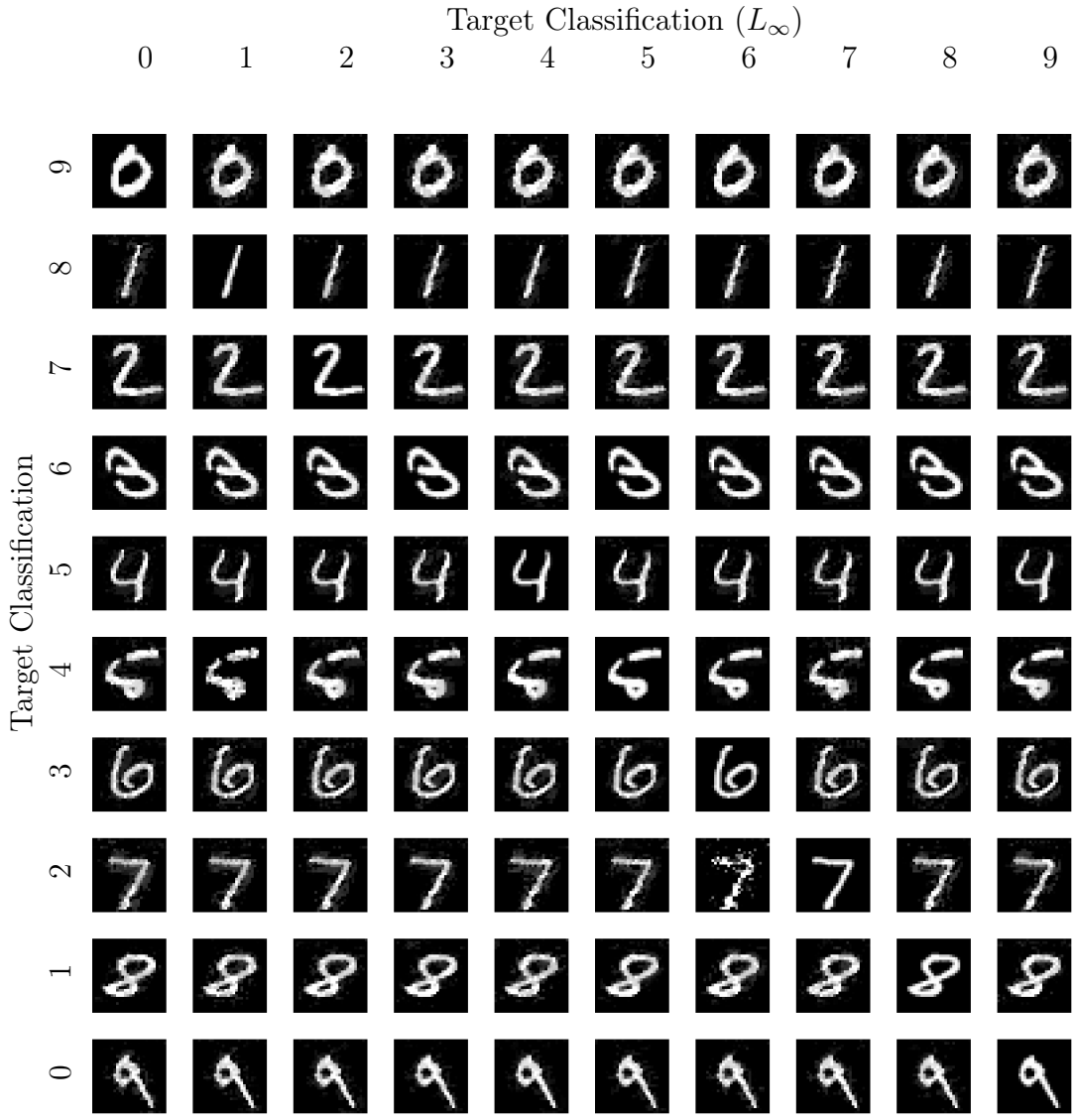

Figure 2.4: Our L_∞ adversary applied to the MNIST dataset performing a targeted attack for every source/target pair. Each digit is the first image in the dataset with that label.

	Untargeted		Average Case		Least Likely	
	mean	prob	mean	prob	mean	prob
Our L_0	48	100%	410	100%	5200	100%
JSMA-Z	-	0%	-	0%	-	0%
JSMA-F	-	0%	-	0%	-	0%
Our L_2	0.32	100%	0.96	100%	2.22	100%
Deepfool	0.91	100%	-	-	-	-
Our L_∞	0.004	100%	0.006	100%	0.01	100%
FGS	0.004	100%	0.064	2%	-	0%
IGS	0.004	100%	0.01	99%	0.03	98%

Table 2.5: Comparison of the three variants of targeted attack to previous work for the Inception v3 model on ImageNet. When success rate is not 100%, the mean is only over successes.

JSMA is unable to run on ImageNet due to an inherent significant computational cost: recall that JSMA performs search for a pair of pixels p, q that can be changed together that make the target class more likely and other classes less likely. ImageNet represents images as $299 \times 299 \times 3$ vectors, so searching over all pairs of pixels would require 2^{36} work on each step of the calculation. If we remove the search over pairs of pixels, the success of JSMA falls off dramatically. We therefore report it as failing always on ImageNet.

We report success if the attack produced an adversarial example with the correct target label, no matter how much change was required. Failure indicates the case where the attack was entirely unable to succeed.

We evaluate on the first $1,000$ images in the test set on CIFAR and MNSIT. On ImageNet, we report on $1,000$ images that were initially classified correctly by Inception v3 [9]. On ImageNet we approximate the best-case and worst-case results by choosing 100 target classes (10%) at random.

The results are found in Table 2.4 for MNIST and CIFAR, and Table 2.5 for ImageNet. [10]

For each distance metric, across all three datasets, our attacks find closer adversarial

[9] Otherwise the best-case attack results would appear to succeed extremely often artificially low due to the relatively low top-1 accuracy

[10] The complete code to reproduce these tables and figures is available online at `http://nicholas.carlini.com/code/nn_robust_attacks`.

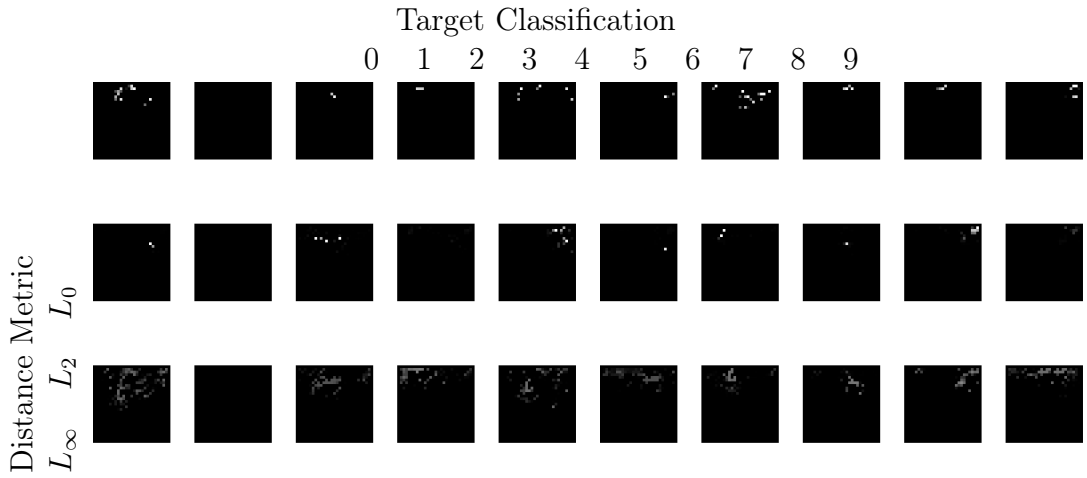

Figure 2.5: Targeted attacks for each of the 10 MNIST digits where the starting image is totally black for each of the three distance metrics.

examples than the previous state-of-the-art attacks, and our attacks never fail to find an adversarial example. Our L_0 and L_2 attacks find adversarial examples with $2\times$ to $10\times$ lower distortion than the best previously published attacks, and succeed with 100% probability. Our L_∞ attacks are comparable in quality to prior work, but their success rate is higher. Our L_∞ attacks on ImageNet are so successful that we can change the classification of an image to any desired label by only flipping the lowest bit of each pixel, a change that would be impossible to detect visually.

As the learning task becomes increasingly more difficult, the previous attacks produce worse results, due to the complexity of the model. In contrast, our attacks perform even better as the task complexity increases. We have found JSMA is unable to find targeted L_0 adversarial examples on ImageNet, whereas ours is able to with 100% success.

It is important to realize that the results between models are not directly comparable. For example, even though a L_0 adversary must change 10 times as many pixels to switch an ImageNet classification compared to a MNIST classification, ImageNet has $114\times$ as many pixels and so the *fraction of pixels* that must change is significantly smaller.

Generating synthetic digits. With our targeted adversary, we can start from *any* image we want and find adversarial examples of each given target. Using this, in Figure 2.5 we show the minimum perturbation to an entirely-black image required to make it classify as each digit, for each of the distance metrics.

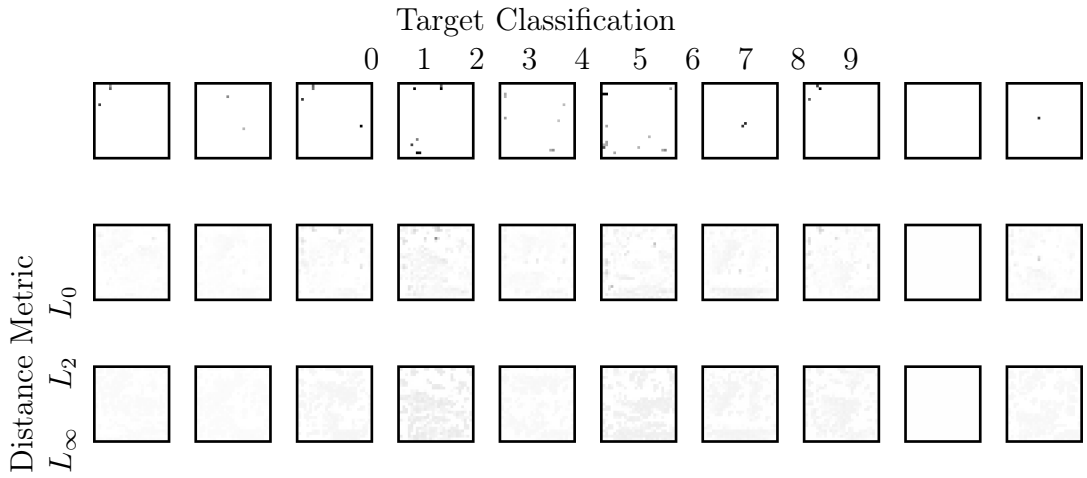

Figure 2.6: Targeted attacks for each of the 10 MNIST digits where the starting image is totally white for each of the three distance metrics.

This experiment was performed for the L_0 task previously [102], however when mounting their attack, "for classes 0, 2, 3 and 5 one can clearly recognize the target digit." With our more powerful attacks, none of the digits are recognizable. Figure 2.6 performs the same analysis starting from an all-white image.

Notice that the all-black image requires no change to become a digit 1 because it is initially classified as a 1, and the all-white image requires no change to become a 8 because the initial image is already an 8.

Runtime Analysis. We believe there are two reasons why one may consider the runtime performance of adversarial example generation algorithms important: first, to understand if the performance would be prohibitive for an adversary to actually mount the attacks, and second, to be used as an inner loop in adversarial re-training [34].

Comparing the exact runtime of attacks can be misleading. For example, we have parallelized the implementation of our L_2 adversary allowing it to run hundreds of attacks simultaneously on a GPU, increasing performance from $10\times$ to $100\times$. However, we did not parallelize our L_0 or L_∞ attacks. Similarly, our implementation of fast gradient sign is parallelized, but JSMA is not. We therefore refrain from giving exact performance numbers because we believe an unfair comparison is worse than no comparison.

All of our attacks, and all previous attacks, are plenty efficient to be used by an adversary. No attack takes longer than a few minutes to run on any given instance.

When compared to L_0, our attacks are $2 \times -10\times$ slower than our optimized JSMA algorithm (and significantly faster than the un-optimized version). Our attacks are typically $10 \times -100\times$ slower than previous attacks for L_2 and L_∞, with exception of iterative gradient sign which we are $10\times$ slower.

Chapter 3

Attack Application: Breaking Defenses

Having developed a strong approach to constructing adversarial examples, we now turn to applying this attack algorithm to various tasks.

3.1 Assorted Defenses

3.1.1 Distillation as a Defense

Distillation was initially proposed as an approach to reduce a large model (the *teacher*) down to a smaller *distilled* model [45]. At a high level, distillation works by first training the teacher model on the training set in a standard manner. Then, we use the teacher to label each instance in the training set with soft labels (the output vector from the teacher network). For example, while the hard label for an image of a hand-written digit 7 will say it is classified as a seven, the soft labels might say it has a 80% chance of being a seven and a 20% chance of being a one. Then, we train the distilled model on the soft labels from the teacher, rather than on the hard labels from the training set. Distillation can potentially increase accuracy on the test set as well as the rate at which the smaller model learns to predict the hard labels [45, 82].

Defensive distillation uses distillation in order to increase the robustness of a neural network, but with two significant changes. First, both the teacher model and the distilled model are identical in size — defensive distillation does not result in smaller models. Second, and more importantly, defensive distillation uses a large *distillation temperature* (described below) to force the distilled model to become more confident

	Best Case				Average Case				Worst Case			
	MNIST		CIFAR		MNIST		CIFAR		MNIST		CIFAR	
	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob
Our L_0	10	100%	7.4	100%	19	100%	15	100%	36	100%	29	100%
Our L_2	1.7	100%	0.36	100%	2.2	100%	0.60	100%	2.9	100%	0.92	100%
Our L_∞	0.14	100%	0.002	100%	0.18	100%	0.023	100%	0.25	100%	0.038	100%

Table 3.1: Comparison of our attacks when applied to defensively distilled networks. Compare to Table 2.4 for undistilled networks.

in its predictions.

Recall that, the softmax function is the last layer of a neural network. Defensive distillation modifies the softmax function to also include a temperature constant T:

$$\text{softmax}(x, T)_i = \frac{e^{x_i/T}}{\sum_j e^{x_j/T}}$$

It is easy to see that $\text{softmax}(x, T) = \text{softmax}(x/T, 1)$. Intuitively, increasing the temperature causes a "softer" maximum, and decreasing it causes a "harder" maximum. As the limit of the temperature goes to 0, softmax approaches max; as the limit goes to infinity, $\text{softmax}(x)$ approaches a uniform distribution.

Defensive distillation proceeds in four steps:

1. Train a network, the teacher network, by setting the temperature of the softmax to T during the training phase.
2. Compute soft labels by apply the teacher network to each instance in the training set, again evaluating the softmax at temperature T.
3. Train the distilled network (a network with the same shape as the teacher network) on the soft labels, using softmax at temperature T.
4. Finally, when running the distilled network at test time (to classify new inputs), use temperature 1.

Fragility of existing attacks

We briefly investigate the reason that existing attacks fail on distilled networks, and find that existing attacks are very fragile and can easily fail to find adversarial examples even when they exist.

L-BFGS and Deepfool fail due to the fact that the gradient of $F(\cdot)$ is zero almost always, which prohibits the use of the standard objective function.

When we train a distilled network at temperature T and then test it at temperature 1, we effectively cause the inputs to the softmax to become larger by a factor of T. By minimizing the cross entropy during training, the output of the softmax is forced to be close to 1.0 for the correct class and 0.0 for all others. Since $Z(\cdot)$ is divided by T, the distilled network will learn to make the $Z(\cdot)$ values T times larger than they otherwise would be. (Positive values are forced to become about T times larger; negative values are multiplied by a factor of about T and thus become even more negative.) Experimentally, we verified this fact: the mean value of the L_1 norm of $Z(\cdot)$ (the logits) on the undistilled network is 5.8 with standard deviation 6.4; on the distilled network (with $T = 100$), the mean is 482 with standard deviation 457.

Because the values of $Z(\cdot)$ are 100 times larger, when we test at temperature 1, the output of F becomes ϵ in all components except for the output class which has confidence $1 - 9\epsilon$ for some very small ϵ (for tasks with 10 classes). In fact, in most cases, ϵ is so small that the 32-bit floating-point value is rounded to 0. For similar reasons, the gradient is so small that it becomes 0 when expressed as a 32-bit floating-point value.

This causes the L-BFGS minimization procedure to fail to make progress and terminate. If instead we run L-BFGS with our stable objective function identified earlier, rather than the objective function $\text{loss}_{F,l}(\cdot)$ suggested by Szegedy *et al.* [125], L-BFGS does not fail. An alternate approach to fixing the attack would be to set

$$F'(x) = \text{softmax}(Z(x)/T)$$

where T is the distillation temperature chosen. Then minimizing $\text{loss}_{F',l}(\cdot)$ will not fail, as now the gradients do not vanish due to floating-point arithmetic rounding. This clearly demonstrates the fragility of using the loss function as the objective to minimize.

JSMA-F (whereby we mean the attack uses the output of the final layer $F(\cdot)$) fails for the same reason that L-BFGS fails: the output of the $Z(\cdot)$ layer is very large and so softmax becomes essentially a hard maximum. This is the version of the attack that Papernot *et al.* use to attack defensive distillation in their paper [103].

JSMA-Z (the attack that uses the logits) fails for a completely different reason. Recall that in the $Z(\cdot)$ version of the attack, we use the input to the softmax for computing the gradient instead of the final output of the network. This removes any potential issues with the gradient vanishing, however this introduces new issues. This version of the attack is introduced by Papernot *et al.* [102] but it is not used to attack

distillation; we provide here an analysis of why it fails.

Since this attack uses the Z values, it is important to realize the differences in relative impact. If the smallest input to the softmax layer is -100, then, after the softmax layer, the corresponding output becomes practically zero. If this input changes from -100 to -90, the output will still be practically zero. However, if the largest input to the softmax layer is 10, and it changes to 0, this will have a massive impact on the softmax output.

Relating this to parameters used in their attack, α and β represent the size of the change at the input to the softmax layer. It is perhaps surprising that JSMA-Z works on un-distilled networks, as it treats all changes as being of equal importance, regardless of how much they change the softmax output. If changing a single pixel would increase the target class by 10, but also increase the least likely class by 15, the attack will not increase that pixel.

Recall that distillation at temperature T causes the value of the logits to be T times larger. In effect, this magnifies the sub-optimality noted above as logits that are extremely unlikely but have slight variation can cause the attack to refuse to make any changes.

Fast Gradient Sign fails at first for the same reason L-BFGS fails: the gradients are almost always zero. However, something interesting happens if we attempt the same division trick and divide the logits by T before feeding them to the softmax function: distillation still remains effective [99]. We are unable to explain this phenomenon.

Applying Our Attacks

When we apply our attacks to defensively distilled networks, we find distillation provides only marginal value. We re-implement defensive distillation on MNIST and CIFAR-10 as described [103] using the same model we used for our evaluation above. We train our distilled model with temperature $T = 100$, the value found to be most effective [103].

Table 3.1 shows our attacks when applied to distillation. All of the previous attacks fail to find adversarial examples. In contrast, our attack succeeds with 100% success probability for each of the three distance metrics.

When compared to Table 2.4, distillation has added almost no value: our L_0 and L_2 attacks perform slightly worse, and our L_∞ attack performs approximately equally. All of our attacks succeed with 100% success.

Effect of Temperature

In the original work, increasing the temperature was found to consistently reduce attack success rate. On MNIST, this goes from a 91% success rate at $T = 1$ to a 24% success rate for $T = 5$ and finally 0.5% success at $T = 100$.

We re-implement this experiment with our improved attacks to understand how the choice of temperature impacts robustness. We train models with the temperature varied from $t = 1$ to $t = 100$.

When we re-run our implementation of JSMA, we observe the same effect: attack success rapidly decreases. However, with our improved L_2 attack, we see no effect of temperature on the mean distance to adversarial examples: the correlation coefficient is $\rho = -0.05$. This clearly demonstrates the fact that increasing the distillation temperature does not increase the robustness of the neural network, it only causes existing attacks to fail more often.

Transferability

Recent work has shown that an adversarial example for one model will often *transfer* to be an adversarial on a different model, even if they are trained on different sets of training data [125, 34], and even if they use entirely different algorithms (i.e., adversarial examples on neural networks transfer to random forests [100]).

Therefore, any defense that is able to provide robustness against adversarial examples *must* somehow break this transferability property; otherwise, we could run our attack algorithm on an easy-to-attack model, and then transfer those adversarial examples to the hard-to-attack model.

Even though defensive distillation is not robust to our stronger attacks, we demonstrate a second break of distillation by transferring attacks from a standard model to a defensively distilled model.

We accomplish this by finding *high-confidence adversarial examples*, which we define as adversarial examples that are strongly misclassified by the original model. Instead of looking for an adversarial example that just barely changes the classification from the source to the target, we want one where the target is much more likely than any other label.

Recall the loss function defined earlier for L_2 attacks:

$$f(x') = \max(\max\{Z(x')_i : i \neq t\} - Z(x')_t, -\kappa).$$

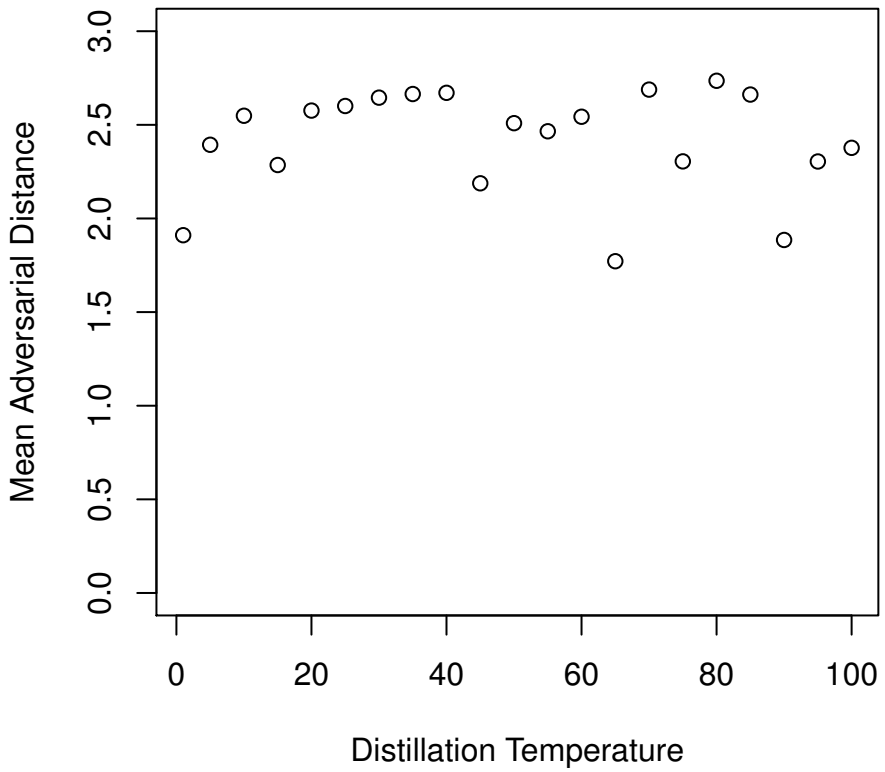

Figure 3.1: Mean distance to targeted (with random target) adversarial examples for different distillation temperatures on MNIST. Temperature is uncorrelated with mean adversarial example distance.

The purpose of the parameter κ is to control the strength of adversarial examples: the larger κ, the stronger the classification of the adversarial example. This allows us to generate high-confidence adversarial examples by increasing κ.

We first investigate if our hypothesis is true that the stronger the classification on the first model, the more likely it will transfer. We do this by varying κ from 0 to 40.

Our baseline experiment uses two models trained on MNIST as described in Section 2.2, with each model trained on half of the training data. We find that the transferability success rate increases linearly from $\kappa = 0$ to $\kappa = 20$ and then plateaus at near-100% success for $\kappa \approx 20$, so clearly increasing κ increases the probability of a successful transferable attack.

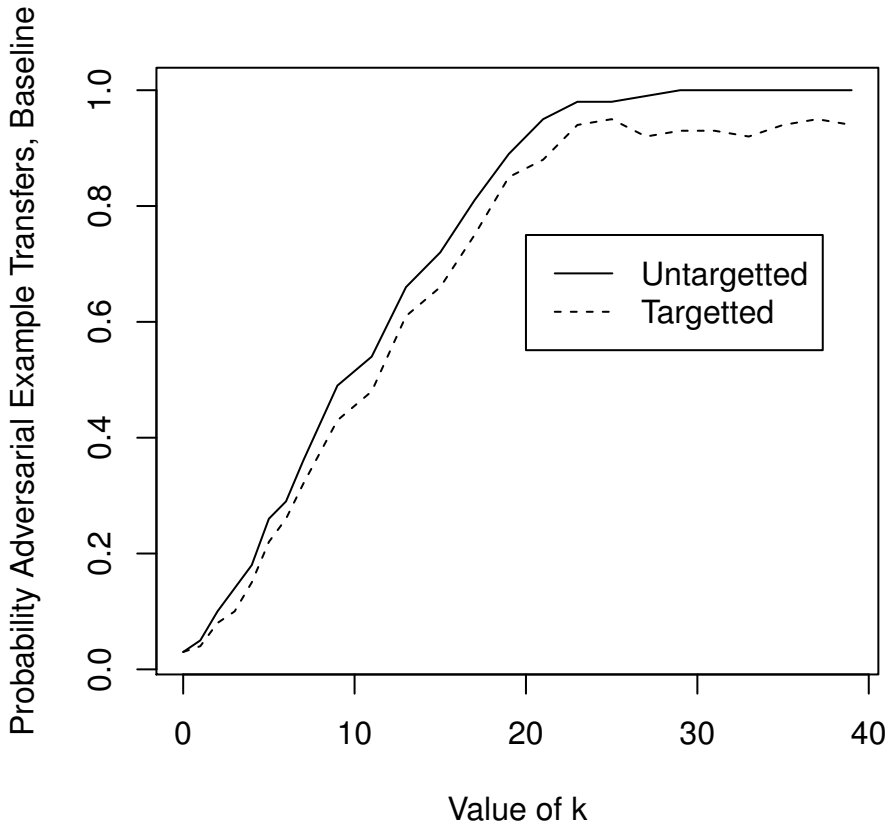

Figure 3.2: Probability that adversarial examples transfer from one model to another, for both targeted (the adversarial class remains the same) and untargeted (the image is not the correct class).

We then run this same experiment only instead we train the second model with defensive distillation, and find that adversarial examples *do* transfer. This gives us another attack technique for finding adversarial examples on distilled networks.

However, interestingly, the transferability success rate between the unsecured model and the distilled model only reaches 100% success at $\kappa = 40$, in comparison to the previous approach that only required $\kappa = 20$.

We believe that this approach can be used in general to evaluate the robustness of defenses, even if the defense is able to completely block flow of gradients to cause our gradient-descent based approaches from succeeding.

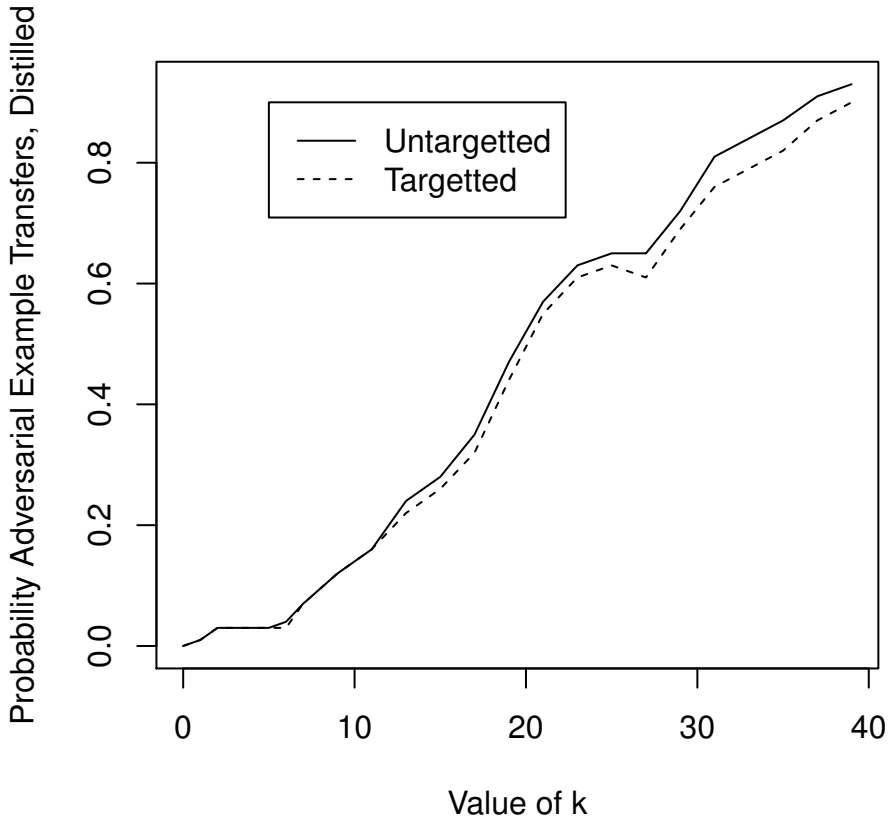

Figure 3.3: Probability that adversarial examples transfer from the baseline model to a model trained with defensive distillation at temperature 100.

3.1.2 MagNet

MagNet [83] was recently proposed as a defense to adversarial examples under the *grey box* threat model: here, the adversary is aware that MagNet is being used to secure a given classifier, as well as the parameters of the classifier, but is not aware of the parameters of the defensive models. MagNet has two defense components:

1. The *detector* examines inputs and decides if they are adversarial or benign. There can be multiple detectors, in which case if any report "adversarial" the input is rejected.

2. The *reformer* takes inputs classified as benign by the detector(s) and reforms them to remove any remaining minor adversarial nature to the instances. The

defender trains multiple reformers and selects one at random, to prevent the adversary from learning its parameters.

The authors argue these defenses complement each other: the detector should be able to detect any adversarial examples with large distortion and are far away from the true data manifold. The reformer takes instances which are close to the data manifold and maps them directly on to the manifold to remove any minor adversarial perturbations that may not have been detected.

The specific instantiation of the detector and reformer suggested by the authors are based on auto-encoders. Given an auto-encoder AE, to detect if an input is adversarial, compare the mean squared error $\|x - AE(x)\|_2$ with a threshold τ chosen to minimize the false positive rate. On CIFAR, the authors suggest one further detector: compute the Jensen-Shannon divergence $JSD(F(x)/T, F(AE(x))/T)$ with temperature T (set to 10 and 40), and report "adversarial" if the difference is greater than a different threshold τ', also chosen to minimize the false positive rate.

Similarly, to reform an adversarial example run it through the auto-encoder and let $x \leftarrow AE(x)$: since the auto-encoder is trained to reconstruct typical instances, this operation projects it back to the data manifold.

Break

We demonstrate MagNet is not robust by generating transferable adversarial examples with our L_2 attack. To do this, we (as the adversary) train 32 local auto-encoders to use as detectors and reformers (different than the 16 the defender will use). To generate adversarial examples, we modify the optimization formulation to the following, where reformers are denoted R_j and the detectors D_j:

$$\text{minimize} \quad \|x' - x\|_2^2 + c \cdot \ell_c(x') + d \cdot \ell_d(x')$$

where the *classifier* loss function ℓ_c is defined as

$$\ell_c(x') = \sum_j \max(\max_{i \neq t} Z(R_j(x'))_i - Z(R_j(x'))_t, -\kappa)$$

and the *detector* loss function ℓ_d is defined as

$$\ell_d(x') = \sum_j \max(D_j(x') - \tau_j, 0).$$

Here, c and d are chosen via binary search simultaneously. The threshold τ_j is chosen such that 1% of validation images are detected as adversarial. (The true defender sets

Dataset	Model	Success	Distortion (L_2)
MNIST	Unsecured	100%	1.64
	MagNet	99%	2.25
CIFAR	Unsecured	100%	0.30
	MagNet	100%	0.45

Table 3.2: The success rate of our attack on MagNet. The last column shows the mean distance to the nearest targeted adversarial example, across the first 1000 test instances, with the target chosen uniformly at random from the incorrect classes.

their remote τ to reject only 0.1% of instances [83]: ours is more strict to help the attack construct adversarial examples that fool the remote detectors more often.) We set the confidence $\kappa = 1$ so that the adversarial examples are more likely to transfer.

We attack by performing 10000 iterations of gradient descent with a learning rate of 10^{-2}. We did not perform hyperparameter search (e.g., picking 32 auto-encoders, $\kappa = 1$, $\tau_j = 0.01$); improved search over these parameters would yield lower distortion adversarial examples.

Figures 1 and 2 contain images of targeted adversarial examples on on the secured network, and Table 3.2 the mean distortion required across the first 1000 instances of the test set with targets chosen uniformly at random among the incorrect classes.

3.1.3 "Efficient Defenses against Adversarial Attack"

Efficient Defenses Against Adversarial Attacks [135] works by making two modifications to standard neural networks. First, the authors propose use of the Bounded ReLU activation function, defined as $BReLU(x) = \min(\max(x, 0), 1)$ instead of standard ReLU [93] which is unbounded above. Second, instead of training on the standard training data $\{(x_i, y_i)\}_{i=1}^n$ they train on $\{(x_i + N_i, y_i)\}_{i=1}^n$ where $N_i \sim \mathcal{N}(0, \sigma^2)$ is chosen fresh for each training instance. On MNIST, $\sigma = 0.3$; for CIFAR, $\sigma = 0.05$. The authors claim that despite training on noise, it is successful on the attacks presented in Chapter 2..

Break

We demonstrate this defense is not robust by generating adversarial examples with our L_2 attack. We do nothing more than apply the attack to the defended network.

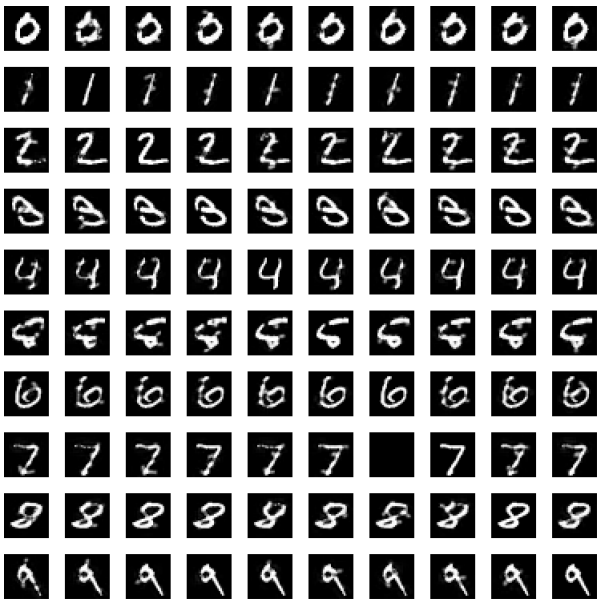

Figure 3.4: MagNet targeted adversarial examples for each source/target pair of images on MNIST. We achieve a 99% grey-box success (the 7 $\rightarrow$ 6 attack failed to transfer).

Figure 3.5: MagNet targeted adversarial examples for each source/target pair of images on CIFAR. We achieve a 100% grey-box success.

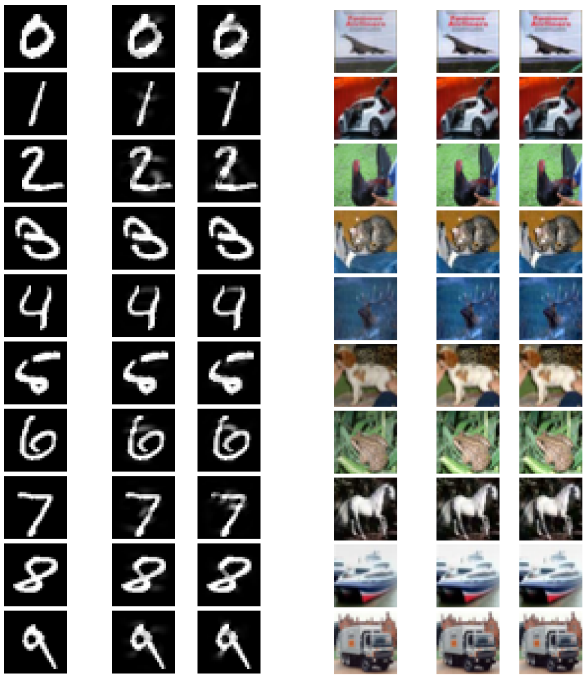

Figure 3.6: Attacks on "Efficient Defenses..." on MNIST and CIFAR-10: *(a)* original reference image; *(b)* adversarial example on the defense with only BReLU; *(c)* adversarial example on the complete defense with Gaussian noise and BReLU.

Figure 3 contains images of adversarial examples on the secured network, and Table 3.3 the mean distortion required across the first 1000 instances of the test set with targets chosen at random among the incorrect classes.

On MNIST, the full defense increases mean distance to the nearest adversarial example by 30%, and on CIFAR by 20%. This is in contrast with other forms of retraining, such as adversarial retraining [77], which increase distortion by a significantly larger amount. Interestingly, we find that BReLU provides some increase in distortion when trained without Gaussian augmentation, but when trained with it, does not help.

3.1.4 APE-GAN

APE-GAN [117] works by constructing a pre-processing network $G(\cdot)$ trained to project both normal instances and adversarial examples back to the data manifold as done in MagNet. The network G is trained with a GAN instead of an auto-encoder. Note that unlike a standard GAN which takes as input a noise vector and

Dataset	Model	Distortion (L_2)
MNIST	Unsecured	2.04
	BReLU	2.14
	Gaussian Noise	2.66
	Gaussian Noise + BReLU	2.58
CIFAR	Unsecured	0.56
	BReLU	0.58
	Gaussian Noise	0.66
	Gaussian Noise + BReLU	0.67

Table 3.3: Neither adding Gaussian data augmentation during training nor using the BReLU activation significantly increases robustness to adversarial examples on the MNIST or CIFAR-10 datasets; success rate is always 100%.

must produce an output image, the generator in APE-GAN takes in an adversarial example and must make it appear non-adversarial. During training, the authors train on adversarial examples generated with the Fast Gradient Sign algorithm [34]; despite this, the authors claim robustness on a wide range of attacks (including the attacks presented in Chapter 2).

Break

We demonstrate APE-GAN is not robust by generating adversarial examples with our L_2 attack. We do nothing more than apply the attack to defended network. That is, we change the loss function to account for the fact that the manifold-projection is done before classification. Specifically, we let

$$\ell(x') = \max(\max\{Z(G(x'))_i : i \neq t\} - Z(G(x'))_t, 0)$$

and solve the same minimization formulation.

Figure 4 contains images of adversarial examples on APE-GAN, and Table 3.4 the mean distortion required across the first 1000 instances of the test set with targets chosen at random among the incorrect classes.

Investigating APE-GAN's Failure. Why are we able to fool APE-GAN? We compare (a) the mean distance between the original inputs and the adversarial examples, and (b) the mean distance between the original inputs and the recovered adversarial examples. We find that the recovered adversarial examples are *less similar* to the

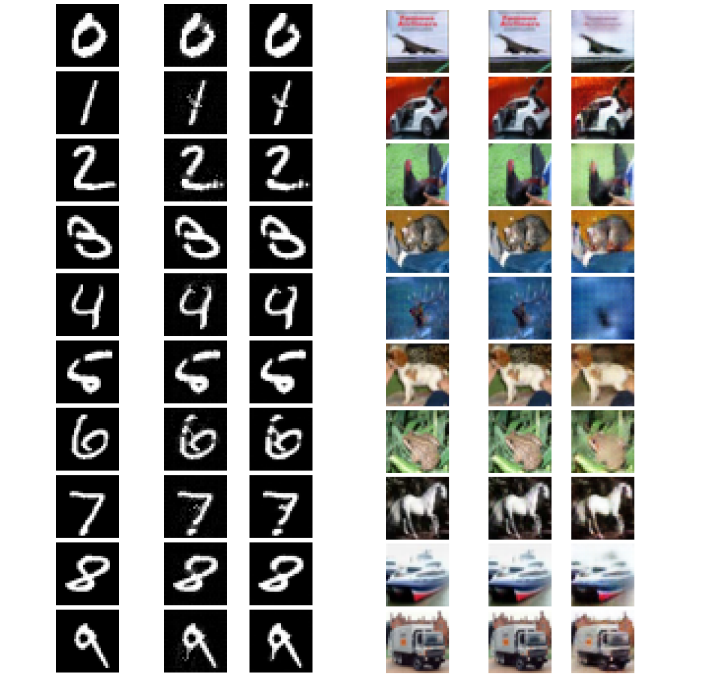

Figure 3.7: Attacks on APE-GAN on MNIST and CIFAR-10: *(a)* original reference image; *(b)* adversarial example on APE-GAN; *(c)* reconstructed adversarial example.

Dataset	Model	Success	Distortion (L_2)
MNIST	Unsecured	100%	2.04
	APE-GAN	100%	2.17
CIFAR	Unsecured	100%	0.43
	APE-GAN	100%	0.72

Table 3.4: APE-GAN does not significantly increase robustness to adversarial examples on the MNIST or CIFAR-10 datasets.

original than the adversarial examples. Specifically, the mean distortion between the adversarial examples and the original instances is 4.3, whereas the mean distortion between the recovered instances and original instances is 5.8.

This indicates that what our adversarial examples have done is fool the generator G into giving reconstructions that are even less similar from the original than the adversarial example. This effect can be observed in Figure 4: faint lines introduced become more pronounced after reconstruction.

3.2 Detection Defenses

Next, we turn to the problem of breaking defenses that attempt to detect adversarial examples. We consider the following defenses:

1. Grosse *et al.* [38] propose two schemes. The first uses a high-dimensional statistical test (Maximum Mean Discrepancy) to detect adversarial examples. The second trains the neural network with a new "adversarial" class.

2. Gong *et al.* [33] detect adversarial examples by building a second neural network that detects adversarial examples from natural images.

3. Metzen *et al* [43] follow a similar approach, but train the detector on the inner layers of the classifier.

4. Li *et al.* [70] propose two schemes. The first performs PCA on the internal convolutional layers of the primary network and trains classifier to distinguish between natural and adversarial data. The second scheme applies a mean-blur to images before feeding them to the network.

5. Hendrycks & Gimpel [44] perform PCA on the pixels of an image and argue adversarial examples place higher emphasis on larger components.

6. Feinman *et al.* [27] detect adversarial examples by keeping dropout [123] on during evaluation; additionally, they construct a kernel density measure and show that adversarial examples are drawn from a different distribution than natural images.

7. Bhagoji *et al.* [10] show that adversarial images require use of more PCA dimensions than natural images.

3.2.1 Attack Approach

In order to evaluate the robustness of each of the above defenses, we take three approaches to target each of the three threat models introduced earlier.

Evaluate with a strong attack (Zero-Knowledge): In this step we generate adversarial examples with our attack and check whether the defense can detect this strong attack. This evaluation approach has the weakest threat model (the attacker is not even aware the defense is in place), so any defense should trivially be able to detect this attack. Failing this test implies that the second two tests will also fail.

Perform an adaptive, white-box attack (Perfect-Knowledge): The most powerful threat model, we assume here the adversary has access to the detector and can mount an adaptive attack. To perform this attack, we construct a new loss function, and generate adversarial examples that both fool the classifier and also evade the detector.

The most difficult step in this attack is to construct a loss function that can be used to generate adversarial examples. In some cases, such a loss function might not be readily available. In other cases, one may exist, but it may not be well-suited to performing gradient descent over. It is of critical importance to choose a good loss function, and we describe how to construct such a loss function for each attack.

Construct a black-box attack (Limited-Knowledge): This attack is the most difficult for the adversary. We assume the adversary knows what type of defense is in place but does not know the detector's paramaters. This evaluation is only interesting if (a) the zero-knowledge attack failed to generate adversarial examples, and (b) the perfect-knowledge attack succeeded. If the strong attack alone succeeded, when the adversary was not aware of the defense, they could mount the same attack in this black-box case. Conversely, if the white-box attack failed, then a black-box attack will also fail (since the threat model is strictly harder).

In order to mount this attack, we rely on the transferability property: the attacker trains a substitute model in the same way as the original model, but on a separate training set (of similar size, and quality). The attacker can access substitute model's parameters, and performs a white-box attack on the substitute model. Finally, we evaluate whether these adversarial examples transfer to the original model.

When the classifier and detector are separate models, we assume the adversary has access to the classifier but not the detector (we are analyzing the increase in security by using the detector).

If the detector and classifier are not separable (i.e., the classifier is trained to also act as a detector), then to perform a fair evaluation, we compare the adversarial examples generated with black-box access to the (unsecured) classifier to adversarial examples generated with only black-box access to both the classifier and detector.

3.2.2 Secondary Classification Based Detection

We now turn to evaluating the ten defenses. The first category of detection schemes we study build a second classifier which attempts to detect adversarial examples. Three of the approaches take this direction.

For the remainder of this subsection, define $F(\cdot)$ to be the classification network and $D(\cdot)$ to be the detection network. $F(\cdot)$ outputs a probability distribution over the 10 classes, and $D : \mathbb{R}^{w \cdot h \cdot c} \to (-\infty, \infty)$ represent the logits of the likelihood the instance is adversarial. That is, $\text{sigmoid}(D(x)) : \mathbb{R}^{w \cdot h \cdot c} \to [0, 1]$ represents the probability the instance is adversarial.

Adversarial Retraining

Grosse *et al.* [38] propose a variant on adversarial re-training. Instead of attempting to classify the adversarial examples correctly (by adding adversarial examples to the training set, with their correct labels), they introduce a new $N + 1$st class — solely for adversarial examples — and train the network to detect adversarial examples. Specifically, they propose the following procedure:

1. Train a model F_{base} on the training data $\mathcal{X}_0 = \mathcal{X}$.
2. Generate adversarial examples on model F_{base} for each $(x_i, y_i) \in X$. Call these examples x'_i.

3. Let $\mathcal{X}_1 = \mathcal{X}_0 \cup \{(x'_i, N+1) : i \in |\mathcal{X}|\}$ where $N+1$ is the new label for adversarial examples.

4. Train a model F_{secured} on the training data $\mathcal{X}_1$.

Gong *et al.* [33] construct a very similar defense technique. Instead of re-training the model F_{secured} completely, they construct a binary classifier D that simply learns to partitions the instances x from x' by training on the training set

$$\mathcal{X}_1 = \{(x_i, 1) : i \in |\mathcal{X}|\} \cup \{(x'_i, 0) : i \in |\mathcal{X}|\}.$$

We re-implement these two defenses and find that adversarial retraining is able to detect adversarial examples when generated with the fast gradient sign and JSMA attacks with near-100% accuracy.

Zero-Knowledge Attack Evaluation. We train these two schemes on the entire MNIST training set, using our attack to generate adversarial examples. In this way we construct a model F_{secured} and D.

We then construct adversarial examples for F_{base} from each image in the test set using our attack. Both approaches detect these previously unseen test adversarial examples. Grosse *et al.* detects 98.5% of attacks as adversarial. Further, it classifies half of the remaining 1.5% correctly. Gong *et al.* achieve 98% accuracy in detecting adversarial examples.

Investigating further, we find that even if we train on adversarial examples generated using an *untargeted* attack, both schemes can detect *targeted* adversarial examples.

Neither of these defenses claim robustness against CIFAR, however when we perform this evaluation, we find to achieve a 70% detection rate requires a 40% false positive rate. This alone can be seen as a failure of these approaches on CIFAR.

Perfect-Knowledge Attack Evaluation. Next, we evaluate these defenses assuming the adversary is aware of these defenses and parameters. That is, we directly attack the defended model. Our experiments revealed that these defenses are ineffective and add almost no increase in robustness.

For Grosse's defense, we use our attack on F_{secured} to generate adversarial examples; it succeeds 100% of the time. We computed the mean L_2-distance from the original sample to the adversarial example. Adversarial examples against F_{base} are at average L_2 distance of 2.05 from the original sample; adversarial examples against F_{secured}

have an average distance of 2.26. Thus the defense has not reduced the success rate at generating adversarial examples, and has only increased the mean distortion by 10%.

Gong's defense does not fare any better. To help construct adversarial examples that will simultaneously fool F_{base} and D, we define a new function $G(\cdot)$ that represents the combination of the classifier (with logits $Z_F(\cdot)$) and detector (with logits $Z_D(\cdot)$). In particular, we define

$$G(x)_i = \begin{cases} Z_F(x)_i & \text{if } i \leq N \\ (Z_D(x)+1) \cdot \max\limits_j Z_F(x)_j & \text{if } i = N+1 \end{cases}$$

Effectively, G acts as a classifier on $N+1$ classes. It is constructed to have two useful properties: if $Z_D(x) > 0$ (i.e., if the detector classifies x as malicious) then we will have

$$\arg\max_i(G(x)_i) = N+1$$

(where N is the new adversarial class), and if $Z_D(x) < 0$ (i.e., if the detector classifies x as natural) then we will have

$$\arg\max_i(G(x)_i) = \arg\max_i(Z_F(x)_i).$$

Why did we choose this particular function $G(\cdot)$? Recall from earlier that when using a gradient-descent based attack algorithm, there is a constant c in the loss function that trades off the relative importance of reducing the distance from the natural instance vs. classifying the input as the target class. If the value of the loss function changes by several orders of magnitude, no value of c will be optimal and adversarial examples will be very hard to find. Therefore, we construct G so that it is mostly linear.

Now we treat this function G as if it were a neural network, and feed it directly into our attack algorithm instead of the function Z. It is easy to see that if an instance x' is classified by G as label $l \leq N$ then both $C(x') = l$ and the detector classifies x' as natural (non-adversarial). This approach successfully generates adversarial examples against Gong's defense with 100% success. The mean L_2 distortion of these adversarial examples is 2.31, an improvement of about 10% over the unprotected scheme.

We therefore conclude neither of these schemes are effective on MNIST. While adversarial examples are detectable with neural networks when not attempting to be evasive, it is possible to construct evasive adversarial examples.

On CIFAR, the above attacks work in an identical manner and give very similar results: the distortion is increased less than 5%.

Limited-Knowledge Attack Evaluation. For brevity, we only describe the attack on Grosse's scheme (Gong's results are identical). We generate transferable adversarial examples as described above. Instead of training one model on the complete training set, we split the training set in half. We assume the first half is known to the attacker and the second half is used by the defender to train their network. Thus, we construct two models R_1 and R_2 each trained on half of the training data. We treat R_2 as the defender's model (the target model for which we want to produce adversarial examples) and give the attacker white-box access to all parameters of model R_1. Then we apply our attack to each instance in the test set to generate adversarial examples for R_1, and we test whether they fool R_2.

As a baseline, we applied this procedure to the unsecured model. The attack succeeded 98% of the time, with a mean L_2 distortion of 5.1. Then, we generated two secured models R_1 and R_2 implementing Grosse's defense, and applied the same attack. We observed a 98% success rate, with a mean L_2 distortion of 5.3, which is only 4% higher than the baseline. Therefore, we conclude that Grosse's defense is ineffective and can be attacked even by an attacker who does not the exact model parameters of the secured model.

Examining Convolutional Layers

In contrast to the prior approach, which attempts to detect adversarial examples based on the contents of the image itself, Metzen *et al.* [43] detect adversarial examples by looking at the inner convolutional layers of the network. They augment the classification neural network with a detection neural network that takes its input from various intermediate layers of the classification network. This detection network is trained identically to Gong's defense above. We refer interested readers to the original paper for complete details on the detector setup [43]. This defense only argues robustness against CIFAR, since it looks at the inner layers of a ResNet (which are not usually used for MNIST).

We then train the detector by attaching it to the output of the first residual block as done in [43]. We confirmed that their detector is able to detect adversarial examples with 99% probability on simple attacks (fast gradient sign or JSMA).

Zero-Knowledge Attack Evaluation. We found training the detector to be very difficult and highly sensitive; we experimented with different optimizers, hyper-parameters, and initializers to reduce the effect of this.

For evaluation, we generate adversarial examples for each instance in the test set, using our attack. The best detector we were able to train correctly obtains an 81%

true positive rate at 28% false positive rate. This is in stark contrast to the 99% success at detecting simpler attacks, which highlights that weak attacks give a biased view of defense robustness.

Similar to the prior defense (that also had a high false positive rate), this defense is unable to perform well even when the adversary is not attempting to evade it.

Perfect-Knowledge Attack Evaluation. Our white-box attack completely defeats Metzen's defense: it is able to produce adversarial examples that simultaneously are mis-classified by the original network and evade the detector. We generate adversarial examples using our attack applied to the same function $G(\cdot)$ defined in Section 3.2.2. The mean distance to adversarial examples increases from 0.169 L_2 distortion on the unsecured model to 0.227 on the secured scheme, an improvement of 34%. However, in absolute terms, the adversarial examples generated are still indistinguishable from the original inputs.

Limited-Knowledge Attack Evaluation. To investigate if this defense is robust to attacks in a black-box setting, we perform a standard transferability test as done above. We split the training data in half, and train two detector models, one on each half of the training data. Then, we attack the second detector given only white-box access to the first detector.

On MNIST, we found that even low-confidence adversarial examples transfer 84% of the time between the two detectors when the classifier network is known by the adversary. By using high-confidence adversarial examples, the attack success rate can be increased to 98% at the cost of increasing the mean distortion by a further 28%, which is small enough that adversarial examples remain indistinguishable from the original images.

3.2.3 Principal Component Analysis Detection

Principal Component Analysis (PCA) transforms a set of points in a n-dimensional space to a new set of points in a k-dimensional space ($k \leq n$) through a linear transformation. We assume the reader is familiar with PCA for the remainder of this section.

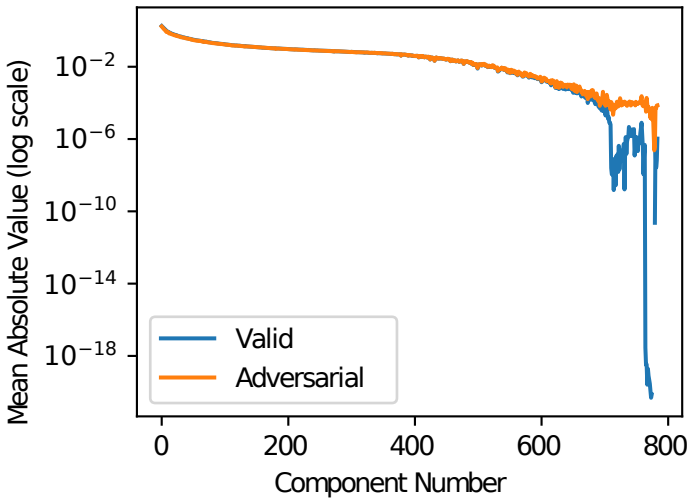

Figure 3.8: PCA on the MNIST dataset reveals a difference between natural images and adversarial images, however this is caused by an artifact of MNIST: border pixels on natural images are often 0 but slightly-positive on adversarial examples.

Input Image PCA

Hendrycks & Gimpel [44] use PCA to detect natural images from adversarial examples, finding that adversarial examples place a higher weight on the larger principal components than natural images (and lower weight on the earlier principal components).

Zero-Knowledge Attack Evaluation. We first reproduce their results by running PCA on MNIST. To see if adversarial examples really do use larger principal components more often, we compute how much each component is used. Let $X_1, \ldots, X_n$ be the training set instances. We define the score $S(j)$ of the jth PCA component as

$$S(j) = \frac{1}{N} \sum_{i=1}^{N} |PCA(X_i)_j|.$$

We train a classification network on the training set and compute the component scores $S(1), \ldots, S(784)$. Then, for each image in the test set, we find the nearest adversarial example with our attack and we compute the component scores on these adversarial examples. The results are plotted in Figure 3.8.

Our results agree with Hendrycks *et. al* [44]: there is no difference on the first principal components, but there is a substantial difference between natural and adversarial

instances on the later components. On the MNIST data set, their defense does detect zero-knowledge attacks, if the attacker does not attempt to defeat the defense.

Looking Deeper. At first glance, this might lead us to believe that PCA is a powerful and effective method for detecting adversarial examples. However, whenever there are large abnormalities in the data, one must be careful to understand their cause.

In this case, the reason for the difference is that there are pixels on the MNIST dataset that are almost always set to 0. Since the MNIST dataset is constructed by taking 24x24 images and centering them (by center-of-mass) on a 28x28 grid, the majority of the pixels on the boundary of natural images are zero. Because these border pixels are essentially always zero for natural instances, the last principal components are heavily concentrated on these border pixels. This explains why the last 74 principal components (9.4% of the components) contribute less than 10^{-30} of the variance on the training set.

In short, the detected difference between the natural and adversarial examples is because the border pixels are nearly always zero for natural MNIST instances, whereas typical adversarial examples have non-zero values on the border. While adversarial examples are different from natural images on MNIST in this way, this is not an intrinsic property of adversarial examples; it is instead due to an artifact of the MNIST dataset. When we perform the above evaluation on CIFAR, there is no detectable difference between adversarial examples and natural data. As a result, the Hendrycks defense is not effective for CIFAR — it is specific to MNIST. Also, this deeper understanding of why the defense works on MNIST suggests that adaptive attacks might be able to avoid detection by simply leaving those pixels unchanged.

Perfect-Knowledge Attack Evaluation. We found that the Hendrycks defense can be broken by a white-box attacker with knowledge of the defense. Details are deferred to Section 3.2.3, where we break a strictly stronger defense. In particular, we found in our experiments that we can generate adversarial examples that are restricted to change only the first k principal components (i.e., leave all later components unchanged), and these adversarial examples that are not detected by the Hendrycks defense.

Dimensionality Reduction

Bhagoji *et al.* [10] propose a defense based on dimensionality reduction: instead of training a classifier on the original training data, they reduce the $W \cdot H \cdot C = N$-

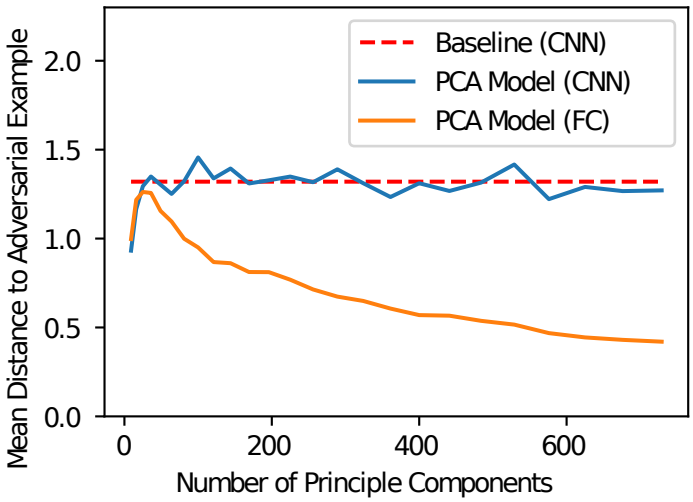

Figure 3.9: Performing dimensionality reduction increases the robustness of a 100-100-10 fully-connected neural network, but is still less secure than just using an unsecured CNN (the baseline). Dimensionality reduction does not help on a network that is already convolutional.

dimensional input (e.g., 784 for MNIST) to a much smaller K-dimensional input (e.g., 20) and train a classifier on this smaller input. The classifier uses a fully-connected neural network: PCA loses spatial locality, so a convolutional network cannot be used (we therefore consider only MNIST).

This defense restricts the attacker so they can only manipulate the first K components: the classifier ignores other components. If adversarial examples rely on the last principal components (as hypothesized), then restricting the attack to only the first K principal components should dramatically increase the required distortion to produce an adversarial example. We test this prediction empirically.

We reimplement their algorithm with their same model (a fully-connected network with two hidden layers of 100 units). We train 26 models with different values of K, ranging from 9 to 784 dimensions. Models with fewer than 25 dimensions have lower accuracy; all models with more than 25 dimensions have 97% or higher accuracy.

Perfect-Knowledge Attack Evaluation. We evaluate Bhagoji's defense by constructing targeted attacks against all 26 models we trained. We show the mean distortion for each model in Figure 3.9. The most difficult model to attack uses only the first 25 principal components; it is nearly 3× more robust than the model that

keeps all 784 principal components.

However, crucially, we find that even the model that keeps the first 25 principal components is *less* robust than almost any standard, unsecured convolutional neural network; an unprotected network achieves both higher accuracy (99.5% accuracy) and better robustness to adversarial examples (measured by the mean distortion). In summary, Bhagoji's defense is not secure against white-box attacks.

Looking Deeper. Next, we show that this result is not an artifact of the network architecture — it is not caused just because fully-connected network are less robust than convolutional networks. We study a second algorithm that Bhagoji *et al.* present but did not end up using, which combines PCA with a convolutional neural network architecture. This allows us to perform an experiment where the network architecture is held fixed, and the only change is whether dimensionality reduction is used or not. In particular, instead of using the first K principal components as features for a fully-connected network, they use PCA to map each image into the reduced-dimensionality PCA space, and then immediately map it back to the image space. This effectively projects the image into a reduced-dimension manifold. They train a convolutional classifier on the projected images. This classifier achieves a higher accuracy (99% when using at least 25 dimensions).

In our experiments we found that this approach is no more robust than an unsecured convolutional network (applied to the original image), despite only using a limited number of the principal components. We conclude that defenses based on limiting the adversary to only the first principal components are not effective. It follows that Hendrycks *et. al*'s defense is broken as well, as the adversarial images generated in this way change only the first K components and leave all later components unchanged.

Hidden Layer PCA

Li *et al.* [70] apply PCA to the values after inner convolutional layers of the neural network, and use a cascade classifier to detect adversarial examples. Specifically, they propose building a *cascade classifier* that accepts the input as natural only if all classifiers C_i accept the input, but rejects it if any do. Each classifier C_i a linear SVM that acts on the PCA of the ith convolutional layer of the network.

They evaluate their scheme on ImageNet. In the remainder of this section we demonstrate their defense is not effective on MNIST and CIFAR; others have shown that attacking an ImageNet classifier would be even easier [88].

Zero-Knowledge Attack Evaluation. Li *et al.* generated adversarial examples with Szegedy *et. al*'s L-BFGS algorithm [125] and found that the first linear SVM achieved 80% true positive rate at 0% false positive rate – an ideal use-case for a cascade classifier. We evaluated the effectiveness of their method at detecting adversarial examples generated with our attack (when trained on adversarial examples generated with our attack). For MNIST, their scheme only achieves a 62% true positive rate at a 37% false positive rate (and 56% at 40% on CIFAR) on the first layer of the cascade classifier. This results causes the cascade classifiers to have a 92% and 98% false positive rates on MNIST/CIFAR. We conclude that Li's defense is not effective against the our attack.

3.2.4 Distributional Detection

Next, we study two defenses that detect adversarial examples by comparing the distribution of natural images to the distribution of adversarial examples. They use classical statistical methods to distinguish natural images from adversarial images.

Maximum Mean Discrepancy

Grosse *et al.* [38] consider a very powerful threat model: assume we are given two sets of images S_1 and S_2, such that we know S_1 contains only natural images, and we know that S_2 contains either all adversarial examples, or all natural images. They ask the question: can we determine which of these two situations is the case?

To achieve this, they use the Maximum Mean Discrepancy (MMD) test [15, 37], a statistical hypothesis test that answers the question "are these two sets drawn from the same underlying distribution?"

The MMD is a theoretically useful technique that can be formally shown to always detect a difference if one occurs. However, it is computationally infeasible to compute, so a simple polynomial-time approximation is almost always used. In our experiments, we use the same approximation used by Grosse *et al.* [37].

To test whether X_1 and X_2 are drawn from the same distribution, Grosse *et al.* use Fisher's permutation test [97] with the MMD test statistic. To do this, initially let $a = MMD(X_1, X_2)$. Then, shuffle the elements of X_1 and X_2 into two new sets Y_1 and Y_2, and let $b = MMD(Y_1, Y_2)$. If $a < b$ then reject the null hypothesis, and conclude they are drawn from different distributions. Repeat this process many times, and repor the p-value as the fraction of times the null hypothesis is rejected.

As a first step, we verified that MMD correctly detects natural instances as being

natural. To do this, we assign X_1 and X_2 to be random samples from the test set. The procedure above correctly fails to reject the null hypothesis ($p > 0.05$) in distinguishing the two distributions.

Zero-Knowledge Attack Evaluation. We repeated this experiment, this time producing targeted adversarial examples with our attack algorithm. Even when using a set of 100 images, MMD fails to reject the null hypothesis ($p > 0.05$). MMD also fails to detect attacks on CIFAR, even with a set size as large as $10,000$ (which uses every sample in the test set). Since MMD is one of the most powerful multidimensional statistical tests, and even it is not effective, we argue that without significant modification, statistical tests will not be able to detect adversarial examples.

Kernel Density Estimation

Feinman *et al.* [27] present a defense they call *kernel density estimation*. They use a Gaussian Mixture Model to model outputs from the final hidden layer of a neural network, and argue that adversarial examples belong to a different distribution than that of natural images.

Specifically, given an instance x classified as label t, kernel density estimation estimates the likelihood of x as

$$KDE(x) = \frac{1}{|X_t|} \sum_{s \in X_t} \exp(\frac{|F^{n-1}(x) - F^{n-1}(s)|^2}{\sigma^2})$$

where X_t is the set of training instances with label t and $F^{n-1}(x)$ is the output of the final hidden layer on input x. The detector is therefore constructed by selecting a threshold τ and reporting x as adversarial if $KDE(x) < \tau$, otherwise reporting x as natural.

The motivation behind this approach is that the later hidden layers of a neural network have been shown to capture high-level semantic information about the input. Therefore, using a simple classifier on this final layer will be more accurate than if it were applied to the original input images, as the prior defense did.

Zero-Knowledge Attack Evaluation. Feinman's defense is able to detect adversarial examples generated with our attack on MNIST, but not on CIFAR. Looking deeper, on CIFAR, for each image in the test set x and closest adversarial example x', we compare $KDE(x')$ to $KDE(x)$. Surprisingly, we find that 80% of the time, the adversarial example has a *higher* likelihood score than the original image.Therefore,

Feinman's defense cannot work on CIFAR. In the remainder of this section, we show how to break this defense on MNIST with increased distortion.

Perfect-Knowledge Attack Evaluation. To mount a white-box attack, we construct a new minimization formulation that differs from the original only in that we introduce a new loss term $\ell_2(x')$ that penalizes being detected by the detector:

$$\text{minimize} \quad \|x - x'\|_2^2 + c \cdot (\ell(x') + \ell_2(x'))$$

where we define

$$\ell_2(x') = \max(-\log(KDE(x')) - \epsilon, 0)$$

where ϵ controls the likelihood measure of the adversarial examples. In our attack, we set ϵ to the median of $-\log(KDE(\cdot))$ on the training set, so that $\ell_2(x') \leq 0$ if and only if $KDE(x')$ is greater than half of the training instances KDE.

In practice, we mount this attack in two phases. First, we solve the original minimization formulation to obtain an adversarial example $\hat{x}$. Typically $\hat{x}$ will be detected by the detector, so in the second phase we modify it to no longer be detected: we use this $\hat{x}$ as the initial value of x' in the above optimization problem and use gradient descent to improve it. Performing this two-step optimization is useful to allow for different constants c chosen for initially generating an adversarial example and for making it not detected.

This approach is effective at generating adversarial examples for MNIST that are not detected by Feinman's defense. The mean distortion is 3.81, up from 2.05 on an unsecured network. In order to implement this attack more efficiently, instead of computing the KDE across all training instances during the inner optimization loop, we randomly sample 100 instances ($< 5\%$ of the total) at first and use only those. A distortion of 3.81 begins to allow humans to notice the adversarial nature of these images (Figure 3.10).

We believe the reason this defense is so much more effective on MNIST than on CIFAR because this defense is effectively a K-nearest-neighbor classifier with a neural network feature extractor. In fact, prior work has shown that using an RBF kernel SVM (essentially identical to what is done here) is effective at defending MNIST to adversarial examples.

Limited-Knowledge Attack Evaluation. Again we only demonstrate the vulnerability on MNIST: the defense does not work on CIFAR. Note that our white-box attack algorithm didn't use the entire training set; it uses only a random sample of 100 instances from the training set (less than 5% of the training instances). This

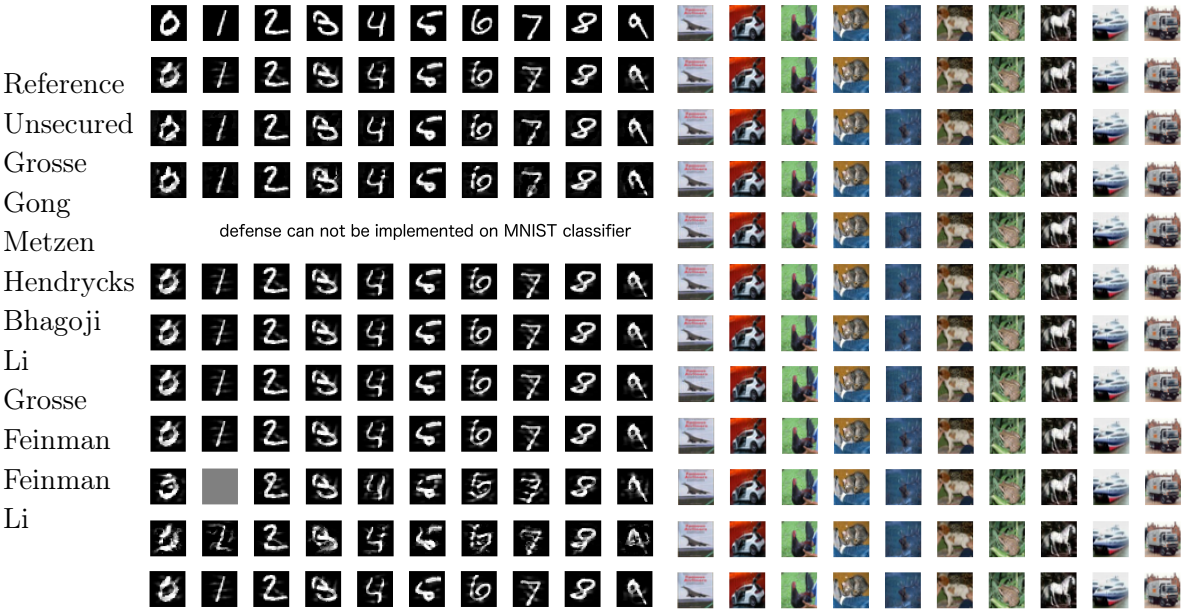

Figure 3.10: **Summary of Results**: adversarial examples on the MNIST and CIFAR datasets for each defense we study. The first row corresponds to the original images.

hints that black-box attacks might succeed. This is indeed the case: we found that when the defender uses the first 95% of the training data, and the last 5% of the training data is left for the adversary (who uses it to form their own KDE estimator), the attack still succeeds with equal distortion.

3.2.5 Normalization Detection

Dropout Randomization

Feinman *et al.* propose a second detection method called *Bayesian neural network uncertainty* that measures the uncertainty of the neural network on the given input. Instead of relying on the reported confidence of the network (which can easily be controlled by an adversary), they add randomization to the network. The hope is that a natural image will have the same (correct) label regardless of the random values chosen, while adversarial examples won't always be predicted with the same label. *Dropout* [123] is used as the method of adding randomness.

To determine the uncertainty of the network on a given instance, we apply the randomized network F_r (with dropout enabled) to the input L times. Then, the uncer-

tainty is defined as

$$U(x) = \left(\frac{1}{L}\sum_{i=1}^{L}\|F_r(x)\|\right) - \left\|\frac{1}{L}\sum_{i=1}^{L}F_r(x)\right\|$$

where we take $\|y\|$ to mean $y^T y$ (the squared L_2 norm). Notice that if the predictions $F_r(x)$ are the same every time, the uncertainty is 0. In other words, the uncertainty is the sum of the variances of each component of the output, computed over the L outputs from the randomized network.

To classify an input x as adversarial or natural, we pick a threshold τ and a number of times to evaluate the network L and compute $U(x)$. We have found the algorithm is not sensitive to the choice of L as long as it is greater than 20 (and we pick $L = 30$ for the remainder of this section). We return "adversarial" if $U(x) > \tau$ and "natural" otherwise.

When we run this detection scheme on the adversarial examples generated by the FGS and JSMA attack algorithms on MNIST, it detects 96% of adversarial examples with a false positive rate lower than 1% when $\tau = 0.02$.

Zero-Knowledge Attack Evaluation. We generated adversarial examples with our attack and find 75% of these adversarial examples are detected by this defense on MNIST. Surprisingly, this defense is even more effective at detecting attacks on CIFAR: it detects 95% of our adversarial examples.

Perfect-Knowledge Attack Evaluation. Our white-box attacks work by defining an objective function that combines the model and detector. In an initial attempt, we follow the approach taken to break the prior detection schemes, let

$$G(x)_i = \begin{cases} Z(x)_i & \text{if } i \leq N \\ (1 + U(x) - \tau) \cdot \max\limits_i Z(x)_i & \text{if } i = N+1 \end{cases}$$

(where $N + 1$ is the new adversarial class). This objective function is defined so that if $U(x) > \tau$ (i.e., x is classified as adversarial by Feinman's defense) then we will have

$$\arg\max_i(G(x)_i) = N + 1,$$

and if $U(x) < \tau$ (i.e., x is classified as natural) then

$$\arg\max_i(G(x)_i) = \arg\max_i(F(x)_i).$$

We then attempt to generated adversarial examples by applying our attack to $G(\cdot)$. However, this attack fails: the defense is still able to detect 60% of adversarial examples.

Looking Deeper. We investigate why this attack failed. Because values of the function $U(\cdot)$ are computed by calling $F(\cdot)$ (and not $Z(\cdot)$), there is no good constant c that our algorithm can select. This causes gradient descent to perform poorly and rarely returns successful adversarial examples.

Therefore, for this defense, we abandon the approach of constructing a single function to optimize over and instead change the loss function ℓ used by the attack. Recall that the typical loss function used (for a given network $Z(\cdot)$) is

$$\ell_Z(x') = \max(\max\{Z(x')_i : i \neq t\} - Z(x')_t, -\kappa).$$

If every prediction $F_r(x)$ of the randomized network produced a very similar result, the uncertainty value $U(x)$ would be very small.

We sample K different deterministic networks $\{Z_j(\cdot) : j \in [1, K]\}$ each with different randomness used during dropout. If we were able to have $\arg\max_i Z_j(x)_i = t$ for every network j, for K big enough, it would be highly likely that $F_r(x)$ would always produce label t for any randomness. Thus, we construct a new loss function $\ell'(x') = \sum_{j=1}^{K} \ell_{Z_j}(x')$ as the average of the loss functions on each fixed model Z_j. Then we use our attack with this revised loss function.

This approach successfully generates adversarial examples that fool the dropout defense with 98% success. On MNIST, the mean l_2 distortion is 3.68, up from the baseline of 2.05. This is the largest distortion required by any of the defenses we have evaluated; the distortion here is nearing the levels of human perceptibility (Figure 3.10). On CIFAR the distortion required again increases to 1.1, a factor of 5× larger, but is still entirely imperceptible (Figure 3.10).

Limited-Knowledge Attack Evaluation. It turns out that adversarial examples generated with the white-box approach transfer with high probability across models. This is due to the fact that our white-box attack assumes we do not know the exact randomization settings, and therefore construct adversarial examples that are effective regardless of randomization. This is similar to the black-box threat model, where the adversary does not have access to the model parameters.

However, to improve the rate of transferability, we again construct two models $F(\cdot)$ and $G(\cdot)$ on different subsets of the training data. We provide the adversary access to the parameters of F and use the white-box attack above to generate adversarial examples for F; we then test whether they fool G.

On MNIST, we find that adversarial examples generated with our algorithm transfer to G with 90% success. We can increase the transfer rate to 98% at the cost of increasing the mean distortion only 15%, to 4.23. While this 15% increase may seem

like a failure of this defense under the black-box threat model, we see this instead as a success of defense under the white-box threat model. It has made constructing adaptive white-box attacks nearly as hard as constructing black-box attacks.

In fact, we find that this is the reason why the CIFAR network has such a larger increase in distortion: to obtain transferable cifar adversarial examples requires a distortion of approximately $4\times$ larger than non-transferable attacks. We consider this the only defense we evaluate that is not completely broken (even though we still can generate adversarial examples that are imperceptible on CIFAR).

Mean Blur

The second detection method proposed by Li *et al.* applies a 3×3 average filter to blur the image before applying the classifier. The authors admit this defense is "overly simplistic" but still argue it is effective at alleviating adversarial examples. We confirm this simple defense can remove adversarial examples generated with fast gradient sign, as they found in their paper.

Zero-Knowledge Attack Evaluation. When we use our attack, we find that this defense effectively removes low-confidence adversarial examples: 80% of adversarial examples (at a mean L_2 distortion of 2.05) are no longer classified incorrectly.

This attack can even partially alleviate high-confidence adversarial examples. To ensure they remain adversarial after blurring, we must increase the distortion by a factor of $3\times$.

Perfect-Knowledge Attack Evaluation. Observe that taking the mean over every 3×3 region on the image is the same as adding another convolutional layer to the beginning of the neural network with one output channel that performs this calculation. Given the network F, we define $F'(x) = F(\text{blur}(x))$ and apply our attack against F'. When we do so, we find that the mean distance to adversarial examples does not increase. Therefore, blurring is not an effective defense.

3.3 Lessons

3.3.1 Properties of adversarial examples

After examining these ten defenses, we now draw conclusions about the nature of the space of adversarial examples and the ability to detect them with different approaches.

Randomization can increase required distortion. By far the most effective defense technique, dropout randomization, made generating adversarial examples nearly five times more difficult on CIFAR. In particular, it makes generating adversarial examples on the network as difficult as generating transferable adversarial examples, a task known to be harder [100]. Additionally, if it were possible to find a way to eliminate transferability, a randomization-based defense may be able to detect adversarial examples. At this time, we believe this is the most promising direction of future work.

MNIST properties may not hold on CIFAR Most defenses that increased the distortion on MNIST had a significantly lower distortion increase on CIFAR. In particular, kernel density estimation, the most effective defense on MNIST, was completely ineffective on CIFAR.

Detection neural networks can be bypassed. Across all of the defenses we evaluate, the least effective schemes used another neural network (or more neural network layers) to attempt to identify adversarial examples. Given that adversarial examples can fool a single classifier, it makes sense that adversarial examples can fool a classifier and detector.

Operating on raw pixel values is ineffective. Defenses that operated directly on the pixel values were too simple to succeed. On MNIST, these defenses provided reasonable robustness against weak attacks; however when evaluating on stronger attacks, these defenses all failed. This should not be surprising: the reason neural networks are used is that they are able to extract deep and meaningful features from the input data. A simple linear detector is not effective at classification when operating on raw pixel values, so it should not be surprising it does not work at detecting adversarial examples. (This can be seen especially well on CIFAR, where even weak attacks often succeed against defenses that operate on the input pixel space.)

3.3.2 Recommendations for Defenses

We have several recommendations for how researchers proposing new defenses can better evaluate their proposals. Many of these recommendations may appear to be obvious, however most of the papers we evaluate do not follow any.

Evaluate using a strong attack. Evaluate proposed defenses using the strongest attacks known. *Do not use fast gradient sign or JSMA exclusively*: most defenses that detect these attacks fail against stronger attacks. In particular, Fast gradient sign was not even designed to produce high-quality attacks: it was created to demonstrate neural networks are highly linear. Using these algorithms as a first test is reasonable first step, but is not sufficient. We recommend new schemes evaluate against strong iterative attacks.

Demonstrate white-box attacks fail. It is not sufficient to show that a defense can detect adversarial examples: one must also show that a adversary aware of the defense can not generate attacks that evade detection. We show how to perform that kind of evaluation: construct a differentiable function that is minimized when the image fools the classifier and is treated as natural by the detector, and apply a strong iterative attack (e.g., C&W's attack) to this function.

Report false positive and true positive rates. When constructing a detection-based defense, it is not enough to report the accuracy of the detector. A 60% accuracy can either be very useful (e.g., if it achieves a high true-positive rate at a 0% false-positive rate) or entirely useless (e.g., if it detects most adversarial images as adversarial at the cost of many natural images as adversarial). Instead, report both the false positive and true positive rates. To allow for comparisons with other work, we suggest reporting at least the true positive rate at 1% false positive rate; showing a ROC curve would be even better.

Evaluate on more than MNIST We have found that defenses that only evaluated on the MNIST dataset typically either (a) were unable to produce an accurate classifier on CIFAR, (b) were entirely useless on CIFAR and were not able to detect even the fast gradient sign attack, or (c) were even weaker against attack on CIFAR than the other defenses we evaluated. Future schemes need to be evaluated on multiple data sets — evaluating their security solely on MNIST is not sufficient. While we have found CIFAR to be a reasonable task for evaluating security, in the future

as defenses improve it may become necessary to evaluate on harder datasets (such as ImageNet [25]).

Release source code. In order to allow others to build on their work, authors should release the source code of their defenses. Not releasing source code only sets back the research community and hinders future security analysis. Seven of the ten we evaluate did not release their code (even after contacting the authors), requiring us to reimplement the defenses before evaluation.

Chapter 4

Attack Application: Speech Recognition

While image recognition has been the most popular domain for analyzing the security of machine learning, it is by no means the only important domain. If we wish to understand to what extent adversarial examples are a fundamental problem with neural networks, and not just a fundamental problem with image recognition, we will have to analyze other domains.

Image recognition can be seen as worst-case analysis for machine learning. The raw image pixels are directly used as features of the machine learning algorithm with no (or minimal) additional processing. Pixel values are uncorrelated and can be arbitrarily changed.

We therefore analyze the security of a second domain with high internal correlation: speech recognition. As discussed in Section 1.2.1, audio is represented as a sequence of magnitudes which corresponds to the waveform to be played. This representation, while exactly what is required to feed to a speaker, is difficult to analyze. Therefore, almost all speech recognition systems operate on the frequency domain, instead of raw audio samples.

In the space of speech recognition, an adversarial example corresponds to some audio sound that a human would transcribe as one phrase, but a machine would transcribe as a different phrase. Recent work on speech recognition is able to transcribe human speech with a *lower* error rate than humans are able to. [132]

4.1 Preliminaries

Neural Networks & Speech Recognition We represent audio as a N-dimensional vector $\vec{x}$. Each element x_i is a signed 16-bit value, sampled at 16KHz. To reduce the input dimensionality, the Mel-Frequency Cepstrum (MFC) transform is often used as a preprocessing step. The MFC splits the waveform into 50 *frames* per second, and maps each frame to the frequency domain through an FFT.

Simple neural networks for classification take a single input and produce a single output probability distribution over all possible output labels. However, in the case of speech-to-text systems, there are exponentially many possible labels (i.e., there are over 26^N possible phrases of length N). This makes it computationally infeasible to enumerate all possible phrases.

Therefore, speech recognition systems often use Recurrent Neural Networks (RNNs) to map an audio waveform to a sequence of probability distributions over individual characters, instead of over complete phrases An RNN is a function which maintains a state vector with $s_0 = \vec{0}$, and

$$(s_{i+1}, y^i) = f(s_i, x_i).$$

where input x_i is one frame of input, and each output y^i is a probability distribution over which character was being spoken during that frame.

We use the DeepSpeech [41] speech-to-text system, which has recently been reproduced and made open source by Mozilla [90]. Internally, it consists of a preprocessing layer which computes the MFC followed by a recurrent neural network using LSTMs [46].

Connectionist Temporal ClassiïňĄcation (CTC) [36] is a method of training a sequence-to-sequence neural network when the alignment between the input and output sequences is not known. DeepSpeech uses CTC because the inputs are an audio sample of a person speaking, and the unaligned transcribed sentences, where the exact position of each word in the audio sample is not known.

We briefly summarize the key details and notation. We refer readers to [40] for an excellent survey of CTC.

Let $\mathcal{X}$ be the input domain — a single frame of input — and $\mathcal{Y}$ be the range — the characters a-z, space, and the special ϵ token (described below). Our neural network $f : \mathcal{X}^N \to [0,1]^{N \cdot |\mathcal{Y}|}$ takes a sequence of N frames $x \in \mathcal{X}$ and returns a probability distribution over the output domain for each frame. We write $f(\vec{x})^i_j$ to mean that

the probability of frame $x_i \in \mathcal{X}$ having label $j \in \mathcal{Y}$. We use $\vec{p}$ to denote a phrase: a sequence of characters $\langle p_i \rangle$, where each $p_i \in \mathcal{Y}$.

While $f(\cdot)$ maps every frame to a probability distribution over the characters, this does not directly give a probability distribution over all *phrases*. The probability of a phrase is defined as a function of the probability of each character.

We begin with two short definitions. We say that a sequence π *reduces to* $\vec{p}$ if starting with π and making the following two operations (in order) yields $\vec{p}$:

1. Remove all sequentially duplicated tokens.
2. Remove all ϵ tokens.

For example, the sequence $a\ a\ b\ \epsilon\ \epsilon\ b$ reduces to $a\ b\ b$.

Further, we say that π is an alignment of $\vec{p}$ with respect to $\vec{y}$ (formally: $\pi \in \Pi(\vec{p}, \vec{y})$) if (a) π *reduces to* $\vec{p}$, and (b) the length of π is equal to the length of $\vec{y}$. The probability of alignment π under $\vec{y}$ is the product of the likelihoods of each of its elements:

$$\Pr(\pi|\vec{y}) = \prod_i \vec{y}^{\,i}_{\pi^i}$$

With these definitions, we can now define the probability of a given phrase $\vec{p}$ under the distribution $\vec{y} = f(\vec{x})$ as

$$\Pr(\vec{p}|\vec{y}) = \sum_{\pi \in \Pi(\vec{p},\vec{y})} \Pr(\pi|\vec{y}) = \sum_{\pi \in \Pi(\vec{p},\vec{y})} \prod_i \vec{y}^{\,i}_{\pi^i}$$

As is usually done, the loss function used to train the network is the negative log likelihood of the desired phrase:

$$\text{CTC-Loss}(f(\vec{x}), \vec{p}) = -\log \Pr(\vec{p}|f(\vec{x})).$$

Despite the exponential search space, this loss can be computed efficiently with dynamic programming [36].

Finally, to *decode* a vector $\vec{y}$ to a phrase $\vec{p}$, we search for the phrase $\vec{p}$ that best aligns to $\vec{y}$.

$$C(\vec{x}) = \arg\max_{\vec{p}} \Pr(\vec{p}|f(\vec{x})).$$

Because computing $C(\cdot)$ requires searching an exponential space, it is typically approximated in one of two ways.

- *Greedy Decoding* searches for the most likely alignment (which is easy to find) and then reduces this alignment to obtain the transcribed phrase:

$$C_{\text{greedy}}(\vec{x}) = \text{reduce}(\arg\max_{\pi} \Pr(\pi|f(\vec{x})))$$

- *Beam Search Decoding* simultaneously evaluates the likelihood of multiple alignments π and then chooses the most likely phrase $\vec{p}$ under these alignments. We refer the reader to [36] for a complete algorithm description.

Adversarial Examples. Evasion attacks have long been studied on machine learning classifiers [73, 7, 6], and are practical against many types of models [11].

When discussion neural networks, these evasion attacks are referred to as *adversarial examples* [125]: for any input x, it is possible to construct a sample x' that is similar to x (according to some metric) but so that $C(x) \neq C(x')$ [11]. In the audio domain, these untargeted adversarial example are usually not interesting: causing a speech-to-text system to transcribe "test sentence" as the misspelled "test *sentense*" does little to help an adversary.

Targeted Adversarial Examples are a more powerful attack: not only must the classification of x and x' differ, but the network must assign a specific label (chosen by the adversary) to the instance x'. The purpose the following section is to show that targeted adversarial examples are possible with only slight distortion on speech-to-text systems.

4.2 Audio Adversarial Examples

Existing work on adversarial examples has focused largely on the space of images, be it image classification [125], generative models on images [64], image segmentation [2], face detection [115], or reinforcement learning by manipulating the images the RL agent sees [8, 49]. In the discrete domain, there has been some study of adversarial examples over text classification [53] and malware classification [39, 48].

There has been comparatively little study on the space of audio, where the most common use is performing automatic speech recognition. In automatic speech recognition, a neural network is given an audio waveform x and perform the speech-to-text transform that gives the transcription y of the phrase being spoken (as used in, e.g., Apple Siri, Google Now, and Amazon Echo).

Constructing targeted adversarial examples on speech recognition has proven difficult. Hidden and inaudible voice commands [17, 136, 121] are targeted attacks, but require synthesizing new audio and can not modify existing audio (analogous to the observation that neural networks can make high confidence predictions for unrecognizable images [96]). Other work has constructed standard untargeted adversarial examples on different audio systems [59, 32]. The current state-of-the-art targeted attack on automatic speech recognition is Houdini [22], which can only construct audio adversarial examples targeting phonetically similar phrases, leading the authors to state

> targeted attacks seem to be much more challenging when dealing with speech recognition systems than when we consider artificial visual systems.

Contributions. In this paper, we demonstrate that targeted adversarial examples exist in the audio domain by attacking DeepSpeech [41], a state-of-the-art speech-to-text transcription neural network. Figure 4.1 illustrates our attack: given any natural waveform x, we are able to construct a perturbation δ that is nearly inaudible but so that $x + \delta$ is recognized as **any** desired phrase. We are able to achieve this by making use of strong, iterative, optimization-based attacks based on the work of [19].

Our white-box attack is end-to-end, and operates directly on the raw samples that are used as input to the classifier. This requires optimizing through the MFC preprocessing transformation, which is has been proven to be difficult [17]. Our attack works with 100% success, regardless of the desired transcription or initial source audio sample.

By starting with an arbitrary waveform, such as music, we can embed speech into audio that should not be recognized as speech; and by choosing silence as the target, we can hide audio from a speech-to-text system.

Audio adversarial examples give a new domain to explore these intriguing properties of neural networks. We hope others will build on our attacks to further study this field. To facilitate future work, we make our code and dataset available[1]. Additionally, we encourage the reader to listen to our audio adversarial examples.

4.2.1 Threat Model & Evaluation Benchmark

Threat Model. Given an audio waveform x, and target transcription y, our task is to construct another audio waveform $x' = x + \delta$ so that x and x' sound similar

[1] `http://nicholas.carlini.com/code/audio_adversarial_examples`

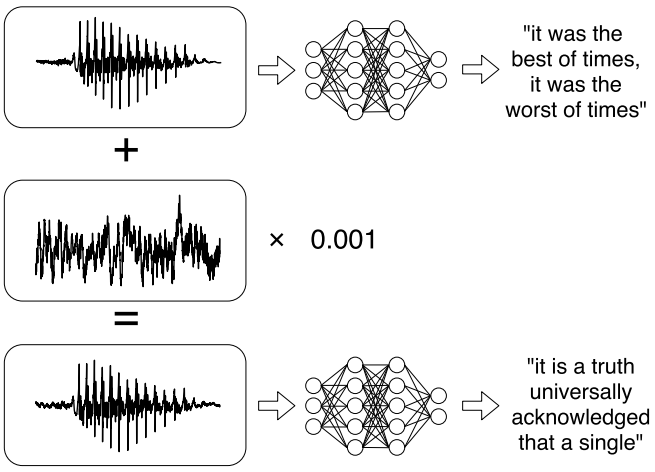

Figure 4.1: Illustration of our attack: given any waveform, adding a small perturbation makes the result transcribe as any desired target phrase.

(formalized below), but so that $C(x') = y$. We report success only if the output of the network matches exactly the target phrase (i.e., contains no misspellings or extra characters).

We assume a white-box setting where the adversary has complete knowledge of the model and its parameters. This is the threat model taken in most prior work [34]. Just as later work in the space of images showed black-box attacks are possible [101, 51]; we expect that our attacks can be extended to black-box attacks. Additionally, we assume our adversarial examples are directly classified without any noise introduced (e.g., by playing them over-the-air and then recording them with a microphone). Initial work on image-based adversarial examples also made this same assumption, which was later shown unnecessary [66, 5].

Distortion Metric. How should we quantify the distortion introduced by a perturbation δ? In the space of images, despite some debate [109], most of the community has settled on l_p metrics as discussed earlier, most often using l_∞ [34, 77], the maximum amount any pixel has been changed. We follow this convention for our audio attacks.

We measure distortion in Decibels (dB): a logarithmic scale that measures the relative loudness of an audio sample:

$$dB(x) = \max_i \; 20 \cdot \log_{10}(x_i).$$

To say that some signal is "10 dB" is only meaningful when comparing it relative to some other reference point. In this section, we compare the dB level of the distortion

δ to the original waveform x. To make this explicit, we write

$$dB_x(\delta) = dB(\delta) - dB(x).$$

Because the perturbation introduced is *quieter* than the original signal, the distortion is a negative number, where smaller values indicate quieter distortions.

While this metric may not be a perfect measure of distortion, as long as the perturbation is small enough, it will be imperceptible to humans. We encourage the reader to listen to our adversarial examples to hear how similar they sound. Alternatively, later, in Figure 4.2, we visualize two waveforms which transcribe to different phrases overlaid.

Evaluation Benchmark. To evaluate the effectiveness of our attack, we construct targeted audio adversarial examples on the first 100 test instances of the Mozilla Common Voice dataset. For each sample, we target 10 different *incorrect* transcriptions, chosen at random such that (a) the transcription is incorrect, and (b) it is theoretically possible to reach that target.

4.2.2 An Initial Formulation

As is commonly done [11, 125], we formulate the problem of constructing an adversarial example as an optimization problem: given a natural example x and any target phrase t, we solve the formulation

$$\begin{aligned} &\text{minimize } dB_x(\delta) \\ &\text{such that } C(x+\delta) = t \\ &\qquad\qquad x + \delta \in [-M, M] \end{aligned}$$

Here M represents the maximum representable value (2^{15} in our case). This constraint can be handled by clipping the values of δ; for notational simplicity we omit it from future formulation. Due to the non-linearity of the constraint $C(x+\delta) = t$, standard gradient-descent techniques do not work well with this formulation.

Prior work [125] has resolved this through the reformulation

$$\text{minimize } dB_x(\delta) + c \cdot \ell(x+\delta, t)$$

where the loss function $\ell(\cdot)$ is constructed so that $\ell(x', t) \leq 0 \iff C(x') = t$. The parameter c trades off the relative importance of being adversarial and remaining close to the original example.

Constructing a loss function $\ell(\cdot)$ with this property is much simpler in the domain of images than in the domain of audio; on images, $f(x')_y$ directly corresponds to the probability of the input x' having label y. In contrast, for audio, we use a second decoding step to compute $C(x')$, and so constructing a loss function is nontrivial.

To begin, we use the CTC loss as the loss function: $\ell(x', t) = \text{CTC-Loss}(x', t)$. For this loss function, one direction of the implication holds true (i.e., $\ell(x', t) \leq 0 \implies C(x') = t$) but the converse does not. Fortunately, this means that the resulting solution will still be adversarial, it just may not be minimally perturbed.

The second difficulty we must address is that when using a l_∞ distortion metric, this optimization process will often oscillate around a solution without converging. Therefore, instead we initially solve the formulation

$$\begin{aligned} &\text{minimize } |\delta|_2^2 + c \cdot \ell(x + \delta, t) \\ &\text{such that } dB_x(\delta) \leq \tau \end{aligned}$$

for some sufficiently large constant τ. Upon obtaining a partial solution δ^* to the above problem, we reduce τ and resume minimization, repeating until no solution can be found.

To solve this formulation, we differentiate through the entire classifier to generate our adversarial examples — starting from the audio sample, through the MFC, and neural network, to the final loss. We solve the minimization problem over the complete audio sample simultaneously. This is in contrast with prior work on hidden voice commands [17], which were generated sequentially, one frame at a time. We solve the minimization problem with the Adam [61] optimizer using a learning rate of 10, for a maximum of $5,000$ iterations.

Evaluation. We are able to generate targeted adversarial examples with 100% success for each of the source-target pairs with a mean perturbation of -31dB. For comparison, this is roughly the difference between ambient noise in a quiet room and a person talking [119]. We encourage the reader to listen to our audio adversarial examples[1]. The 95% interval for distortion ranged from -15dB to -45dB.

The longer a phrase is, the more difficult it is to target: every extra character requires approximately a 0.1dB increase in distortion. However, conversely, we observe that the longer the initial source phrase is, the *easier* it is to make it target a given transcription. These two effects roughly counteract each other (although we were not able to measure this to a statistically significant degree of certainty).

Generating a single adversarial example requires approximately one hour of compute time on commodity hardware (a single NVIDIA 1080Ti). However, due to the

massively parallel nature of GPUs, we are able to construct 10 adversarial examples simultaneously, reducing the time for constructing any given adversarial example to only a few minutes.[2]

4.2.3 Improved Loss Function

In Section 2.3.1, we demonstrate that the choice of loss function impacts the final distortion of generated adversarial examples by a factor of 3 or more. We now show the same holds in the audio domain, but to a lesser extent. While CTC loss is highly useful for training the neural network, we show that a carefully designed loss function allows generating better lower-distortion adversarial examples. For the remainder of this section, we focus on generating adversarial examples that are only effective when using greedy decoding.

In order to minimize the CTC loss (as done in § 4.2.2), an optimizer will make *every* aspect of the transcribed phrase more similar to the target phrase. That is, if the target phrase is "ABCD" and we are already decoding to "ABCX", minimizing CTC loss will still cause the "A" to be more "A"-like, despite the fact that the only important change we require is for the "X" to be turned into a "D".

This effect of making items classified more strongly as the desired label despite already having that label led to the design of a more effective loss function:

$$\ell(y,t) = \max\left(y_t - \max_{t' \neq t} y_{t'}, 0\right).$$

Once the probability of item y is larger than any other item, the optimizer no longer sees a reduction in loss by making it more strongly classified with that label.

We now adapt this loss function to the audio domain. Assume we were given an alignment π that aligns the phrase $\vec{p}$ with the probabilities $\vec{y}$. Then the loss of this sequence is

$$L(\vec{x},\pi) = \sum_i \ell(f(\vec{x})^i, \pi_i).$$

We make one further improvement on this loss function. The constant c used in the minimization formulation determines the relative importance of being close to the original symbol versus being adversarial. A larger value of c allows the optimizer to place more emphasis on reducing $\ell(\cdot)$.

[2]Due to implementation difficulties, after constructing adversarial examples simultaneously, we must fine-tune them individually afterwards.

In audio, consistent with prior work [17] we observe that certain characters are more difficult for the transcription to recognize. When we choose only one constant c for the complete phrase, it must be large enough so that we can make the most difficult character be transcribed correctly. This forces c to be larger than necessary for the easier-to-target segments. To resolve this issue, we instead use the following formulation:

$$\begin{aligned} &\text{minimize } |\delta|_2^2 + \sum_i c_i \cdot L_i(x+\delta, \pi_i) \\ &\text{such that } dB_x(\delta) < \tau \end{aligned}$$

where $L_i(\vec{x}, \pi_i) = \ell(f(\vec{x})^i, \pi_i)$. Computing the loss function requires choice of an alignment π. If we were not concerned about runtime efficiency, in principle we could try all alignments $\pi \in \Pi(\vec{p})$ and select the best one. However, this is computationally prohibitive.

Instead, we use a two-step attack:

1. First, we let x_0 be an adversarial example found using the CTC loss (following §4.2.2). CTC loss explicitly constructs an alignment during decoding. We extract the alignment π that is induced by x_0 (by computing $\pi = \arg\max_i f(x_0)^i$). We fix this alignment π and use it as the target in the second step.

2. Next, holding the alignment π fixed, we generate a less-distorted adversarial example x' targeting the alignment π using the improved loss function above to minimize $|\delta|_2^2 + \sum_i c_i \cdot \ell_i(x+\delta, \pi)$, starting gradient descent at the initial point $\delta = x_0 - x$.

Evaluation. We repeat the evaluation from Section 4.2.2 (above), and generate targeted adversarial examples for the first 100 instances of the Common Voice test set. We are able to reduce the mean distortion from -31dB to -38dB. However, the adversarial examples we generate are now only guaranteed to be effective against a greedy decoder; against a beam-search decoder, the transcribed phrases are often more similar to the target phrase than the original phrase, but do not perfectly match the target.

Figure 4.2 shows two waveforms overlaid; the blue, thick line is the original waveform, and the orange, thin line the modified adversarial waveform. This sample was chosen randomly from among the training data, and corresponds to a distortion of -30dB. Even visually, these two waveforms are nearly indistinguishable.

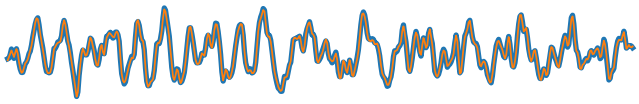

Figure 4.2: Original waveform (blue, thick line) with adversarial waveform (orange, thin line) overlaid; it is nearly impossible to notice a difference. The audio waveform was chosen randomly from the attacks generated and is 500 samples long.

4.2.4 Audio Information Density

Recall that the input waveform is converted into 50 *frames* per second of audio, and DeepSpeech outputs one probability distribution of characters per frame. This places the theoretical maximum density of audio at 50 characters per second. We are able to generate adversarial examples that produce output at this maximum rate. Thus, short audio clips can transcribe to a long textual phrase.

The loss function $\ell(\cdot)$ is simpler in this setting. The *only* alignment of $\vec{p}$ to $\vec{y}$ is the assignment $\pi = \vec{p}$. This means that the logit-based loss function can be applied directly without first heuristically finding an alignment; any other alignment would require omitting some character.

We perform this attack and find it is effective, although it requires a mean distortion of -18dB.

4.2.5 Starting from Non-Speech

Not only are we able to construct adversarial examples that cause DeepSpeech to transcribe the incorrect text for a person's speech, we are also able to begin with arbitrary non-speech audio sample and make that recognize as any target phrase. No technical novelty on top of what was developed above is required to mount this attack: we only let the initial audio waveform be non-speech.

To evaluate the effectiveness of this attack, we take five-second clips from classical music (which contain no speech) and target phrases contained in the Common Voice dataset. We have found that this attack requires more computational effort (we perform $20,000$ iterations of gradient descent) and the total distortion is slightly larger, with a mean of -20dB.

4.2.6 Targeting Silence

Finally, we find it is possible to *hide* speech by adding adversarial noise that causes DeepSpeech to transcribe nothing. While performing this attack without modification (by just targeting the empty phrase) is effective, we can slightly improve on this if we define silence to be an arbitrary length sequence of only the space character repeated. With this definition, to obtain silence, we should let

$$\ell(\vec{x}) = \sum_i \max\left(\max_{t\in\{\epsilon,\text{“ ”}\}} f(\vec{x})^i_t - \max_{t'\notin\{\epsilon,\text{“ ”}\}} f(\vec{x})^i_{t'}, 0\right).$$

We find that targeting silence is easier than targeting a specific phrase: with distortion less than -45dB below the original signal, we can turn any phrase into silence.

This partially explains why it is easier to construct adversarial examples when starting with longer audio waveforms than shorter ones: because the longer phrase contains more sounds, the adversary can silence the ones that are not required and obtain a subsequence that nearly matches the target. In contrast, for a shorter phrase, the adversary must synthesize new characters that did not exist previously.

4.3 Audio Adversarial Example Properties

4.3.1 Evaluating Single-Step Methods

In contrast to prior work which views adversarial examples as “blind spots” of a neural network, Goodfellow *et al.* [34] argue that adversarial examples are largely effective due to the locally linear nature of neural networks.

The Fast Gradient Sign Method (FGSM) [34] demonstrates that this is true in the space of images. FGSM takes a single step in the direction of the gradient of the loss function. That is, given network F with loss function ℓ, we compute the adversarial example as

$$x' \leftarrow x - \epsilon \cdot \text{sign}(\nabla_x \ell(x, y)).$$

Intuitively, for each pixel in an image, this attack asks “in which direction should we modify this pixel to minimize the loss?” and then taking a small step in that direction for every pixel simultaneously. This attack can be applied directly to audio, changing individual samples instead of pixels.

However, we find that this type of single-step attack is not effective on audio adversarial examples: the inherent non-linearity introduced in computing the MFCCs, along

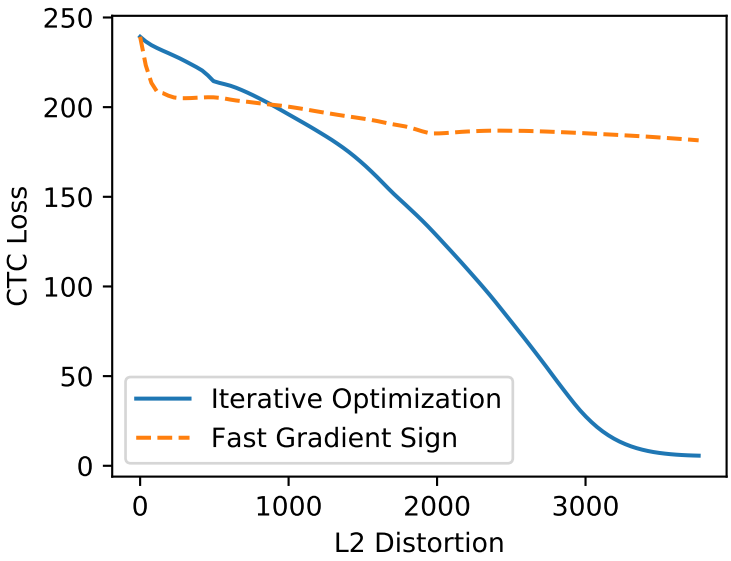

Figure 4.3: CTC loss when interpolating between the original audio sample and the adversarial example (blue, solid line), compared to traveling equally far in the direction suggested by the fast gradient sign method (orange, dashed line). Adversarial examples exist far enough away from the original audio sample that solely relying on the local linearity of neural networks is insufficient to construct targeted adversarial examples.

with the depth of many rounds of LSTMs, introduces a large degree of non-linearity in the output.

In Figure 4.3 we compare the value of the CTC loss when traveling in the direction of a known adversarial example, compared to traveling in the fast gradient sign direction. While initially (near the source audio sample), the fast gradient direction is more effective at reducing the loss function, it quickly plateaus and does not decrease afterwards. On the other hand, using iterative optimization-based attacks find a direction that eventually leads to an adversarial example. (Only when the CTC loss is below 10 does the phrase have the correct transcription.)

We do, however, observe that the FGSM can be used to produce *untargeted* audio adversarial examples, that make a phrase misclassified (although optimization methods again can do so with less distortion).

4.3.2 Robustness of Adversarial Examples

The minimally perturbed adversarial examples we construct in Section 4.2.2 can be made non-adversarial by trivial modifications to the input. Here, we demonstrate here that it is possible to construct adversarial examples robust to various forms of noise.

Robustness to pointwise noise. Given an adversarial example x', adding pointwise random noise σ to x' and returning $C(x+\sigma)$ will cause x' to lose its adversarial label, even if the distortion σ is small enough to allow normal examples to retain their classification.

We generate a high confidence adversarial example x' [11], and make use of Expectation over Transforms [5] to generate an adversarial example robust to this synthetic noise at $-30dB$. The adversarial perturbation increases by approximately 10dB when we do this.

Robustness to MP3 compression. Following [4], we make use of the straight-through estimator [9] to construct adversarial examples robust to MP3 compression. We generate an adversarial example x' such that $C(\text{MP3}(x'))$ is classified as the target label by computing gradients of the CTC-Loss assuming that the gradient of the MP3 compression is the identity function. While individual gradient steps are likely not correct, in aggregate the gradients average out to become useful. This allows us to generate adversarial examples with approximately $15dB$ larger distortion that remain robust to MP3 compression.

4.4 Open Questions

Can these attacks be played over-the-air? Image-based adversarial examples have been shown to be feasible in the physical world [66, 5]. In the audio space, both hidden voice commands and Dolphin Attack's inaudible voice commands are effective over-the-air when played by a speaker and recorded by a microphone [17, 136].

The audio adversarial examples we construct above do not remain adversarial after being played over-the-air, and therefore present a limited real-world threat; however, just as the initial work on image-based adversarial examples did not consider the physical channel and only later was it shown to be possible, we believe further work will be able to produce audio adversarial examples that are effective over-the-air.

Do universal adversarial perturbations [86] exist? One surprising observation is that on the space of images, it is possible to construct a single perturbation δ that when applied to an arbitrary image x will make its classification incorrect. These attacks would be powerful on audio, and would correspond to a perturbation that could be played to cause any other waveform to recognize as a target phrase.

Are audio adversarial examples transferable? That is, given an audio sample x, can we generate a single perturbation δ so that $f_i(x+\delta) = y$ for multiple classifiers f_i? Transferability is believed to be a fundamental property of neural networks [100], significantly complicates constructing robust defenses, and allows attackers to mount black-box attacks [72]. Evaluating transferability on the audio domain is an important direction for future work.

Which existing defenses can be applied audio? To the best of our knowledge, all existing defenses to adversarial examples have only been evaluated on image domains. If the defender's objective is to produce a robust neural network, then it should improve resistance to adversarial examples on all domains, not just on images. Audio adversarial examples give another point of comparison.

Chapter 5

Malware Classification

Up until now, we have shown that neural networks appear to be especially vulnerable to test time evasion attacks, allowing an attacker to deliberately modify an instance to be misclassified by a neural network classifier.

While it may be acceptable to deploy easily evadable deep neural networks in some domains, there are many security-critical domains where this is not the case. We now turn to the case of malware classification, where a defender trains a model to classify a given file sample as either malicious or benign. Because a malware classifier is inherently designed to classify something constructed by an active adversary, it is critical that it is robust against attack.

There are three key differences that differentiate malware classification from typical classification on, e.g., images, audio, or text:

- Features of a malware sample have significant interdependence (e.g., if a loaded library is removed, all other calls to that library must also be removed). In contrast, features for image recognition or audio processing are independent and uncorrelated (e.g., changing one pixel of an image does not mandate that pixels in a different portion of the image be changed as well).
- Not all features are equally easily modified. While it may be trivial to insert a new file access, removing a system call may be difficult if that functionality was a necessary piece of the malicious behavior; inserting a valid code signature may be nearly impossible. Again, this contrasts other domains where every feature is equally easy to modify.
- There is no simple distance metric that captures similarity between malware files. Whereas an image x' is called adversarial with respect to an unmodified,

clean image x if $\|x' - x\|_p < \epsilon$ for some metric $\|\cdot\|$ and distortion bound ϵ, no such metric exists for malware, where changing any one feature may have a dramatic impact on functionality.

For these reasons, in this chapter we focus on a restricted threat model that resolves all three difficulties simultaneously that we call an *insertion adversary*. An insertion adversary can not modify or remove *any* existing functionality, and can only insert *new* functionality that acts as an effective no-op: for example, adding a new file read is allowed; adding a file-deletion operation is not. Because no existing features are modified, any feature interdependence can be ignored. Similarly, inserting new functionality is often as trivial as inserting a few new instructions whose result is ignored. Under this limited threat model, we design attacks that bring the true positive rate of a state-of-the-art classifier to 0% (i.e., our attacks can make the classifier recognize any malware file as benign).

We then turn to constructing defenses. Instead of attempting to take a standard neural network—which is known to perform poorly under attack—and improve its robustness (as prior work does) we propose making architectural changes to neural networks to make neural networks **provably** robust to *all* attacks under our restricted threat model.

Indeed, architecture design has been one of the most significant developments which have allowed neural networks to succeed in other domains. Neural networks only began to succeed for image recognition once Convolutional Neural Networks (CNNs) [67] were developed, as they impose useful priors for images. Similarly, the design of Long Short-Term Memory (LSTM) [46] neural networks for sequence classification has proven esential.

Through careful design, on two different datasets we construct two different architectures which are *monotonic with respect to insertion*. That is, if an adversary inserts new functionality, our monotonic classifiers will only ever increase their maliciousness score. Through several small-scale experiments, we validate that our classifiers are able to learn to solve monotonic problems and are not overly constrained.

On our malware dataset, our unsecured, non-monotonic neural network classifiers far out perform baseline classifiers, but are exceptionally fragile when attacked (e.g., most malware sample need only insert 2 new benign file accesses to switch the classification from malicious to benign). We prove and empirically verify our monotonic classifiers are robust under a limited threat model, but find this monotonicity comes at a cost of a 10 to 20 percentage point reduction in accuracy.

Study	Domain	Dataset Size	Features	Low FPR? (≤5%)	Temporal Splitting?	Evasion Analysis?	Robustness Proof?
Kephart *et al.*, 1995 [58]	Boot Sector	350	Static	Yes	-	-	-
Tesauro *et al.*, 1996 [126]	Boot Sector	300	Static	Yes	-	-	-
Dahl *et al.* 2013 [24]	Malware	2,600,000	Static	Yes	-	-	-
Saxe and Berlin, 2015 [113]	Malware	430,000	Static	Yes	Yes	-	-
Makandar and Patrot, 2015 [79]	Malware	3,100	Static	-	-	-	-
Pascanu *et al.*, 2015 [105]	Malware	500,000	Dynamic	Yes	-	-	-
Grosse *et al.*, 2016 [39]	Android	120,000	Static	Yes	-	Yes	-
Kolosnjaji *et al.*, 2016 [63]	Malware	4,700	Dynamic	-	-	-	-
Kabanga *et al.*, 2017 [55]	Malware	9,500	Static	-	-	-	-
McLaughlin *et al.* [80]	Android	21,00	Static	Yes	-	-	-
Wang *et al.*, 2017 [128]	Malware	26,000	Dynamic	-	-	Yes	-
Our Classifier	**Malware**	**5,300,000**	**Dynamic**	**Yes**	**Yes**	**Yes**	**Yes**

Table 5.1: Survey of related work applying neural networks to malware classification. Only one paper performs an evaluation using proper temporal splitting (see Section 5.3.4), and only two perform any evasion analysis. This makes accuracy comparisons with prior work exceptionally difficult, as the focus of this chapter is robustness in the presence of evasion attacks.

5.1 Related Work

The area of malware classification has a long history. We refer the reader to Idika and Mathur [50] for an excellent survey which performs a detailed study of 45 different classification techniques. They break malware detection down into two broad categories: anomaly-based detection (which defines normal execution, and flags anything that is abnormal), and signature-based detection (which explicitly identifies known-bad behavior). Our work falls under the category of signature-based detection, because we explicitly train a detector to look for differences between malicious samples and benign samples. However, because we are applying neural networks, we do not need to hand-engineer potential signatures: these will be discovered automatically through the process of training the neural network.

Our work also directly follows an important line of work on adversarial machine learning, and specifically the line focused specifically on adversarial examples on neural networks [125, 11] that we have studied previously in this dissertation.

We are motivated to design a novel provable defense architecture in large part due to the failure of existing defenses to hold up to scrutiny.

We are not the first to combine these two areas of research and propose applying neural networks to malware classification. In fact, as early as 1995, researchers had already begun studying applying neural networks to detecting malware. These neural networks were incredibly simple—containing just hundreds of neurons in only one or two layers—but were effective at classifying simple boot sector malware.

However, as neural networks went out of favor, so too did attempts to apply them to malware classification. Then, in 2013, Dahl *et al.* applied neural networks to classify

an incredibly large malware dataset, consisting of 2.6 million samples. Their classifier was a simple architecture (a Multilayer Perceptron, which consists exclusively of fully-connected layers) and was effective due to heavy pre-processing to reduce the dimensionality of the input space. Since this, there have been several proposals to apply neural networks to malware classification. Each has subtle differences between their approach (using different styles of neural networks), objective, and domain (i.e., some classify typical desktop malware, whereas others target Android malware).

There are several significant differences that separate our work from prior work, and make comparison to prior work difficult. We summarize these key differences between our work and prior work in Table 5.1.

Dataset differences. Most obviously, the datasets used in prior papers often differs, in part due to the proprietary nature of malware datasets (indeed, our dataset is also unfortunately non-public). Many of these studies collected known-malware files, and then collect files which are clearly benign. In contrast, **all of our files are suspicious**: our dataset consists exclusively of samples that were submitted to VirusTotal by users who were uncertain of their maliciousness. The best commercial anti-virus programs achieve lower than a 70% true positive rate at a 0.1% false positive rate on our dataset. In contrast, many prior papers, on different datasets, achieve over 99% accuracy at 0.1% false positive rate on their datasets , indicating likely their dataset is much more simplistic. [1]

High False Positive Rates. Four of the ten prior papers did not report false positive rates or they were above 5%, too high to be useful. Due to a low base-rate, malware classifiers must have a very low false-positive rate to be useful.

***Temporally Inconsistent* Analysis.** Of the ten prior papers applying malware classification to neural networks, only *one* performed a proper *temporal* evaluation (see 5.3.4). On malware classification, it is critical that all malware used to *evaluate* the classifier was written **after** the oldest *training* sample. Malware evolves with the specific goal of becoming more difficult to detect. If we perform standard cross-validation [1], it is likely that Version 2 of some malware sample will be a part of the training set with Version 1 part of the test set. Partitioning data in a temporally-consistent manner often results in a 20 to 30 percentage point reduction in accuracy:

[1] For example, Saxe and Berlin [113], who otherwise perform the best evaluation evaluations, only consider a sample benign if no AV vendor labels it as malicious, and only consider a sample as malicious if *at least* 30% of AV vendors do. This means the hard samples–labeled as malware by some but not most AV vendors—will be removed from the dataset, significantly biasing accuracy.

a classifier that achieves 95% accuracy when evaluated temporally-**in**consistently will only achieve a 75% accuracy when evaluated in a temporally consistent manner [56].

Evasion Analysis. A malware classifier that is easily evaded is not a useful malware classifier, as malicious actors will quickly work to evade any deployed detection approach. Only *two* of the prior papers even discuss the possibility of an evasion attack. As such, the other classifiers will often be highly brittle. In contrast, the entire purpose of this chapter is to study the ability of an adversary to evade detection and develop defenses that prevent this. We introduce powerful attacks that easily evade our initial (non-robust) malware detectors, reducing true positive rate of our classifier to **zero** with only simple, functionality-preserving operations.

Achieves Robustness. Of the prior work that even considers evasion attacks against the classifier, none of it is effective at producing a robust classifier to the types of potentially unbounded, but easily automated, attacks we consider here. Our classifiers are designed to be provably robust against attacks: under the threat model we outline, they can not be fooled.

5.2 Motivation: Neural Network Priors

Neural networks are often said to succeed due to their ability to learn from raw features in a completely unsupervised manner without human knowledge [68]. This is in contrast to many other approaches that require feature-engineering to achieve high accuracy. While it is true that neural networks are able to learn from raw features, this does not mean that no human knowledge is necessary to make neural networks perform well. In fact, the opposite is true. Much of the success of neural networks can be directly attributed to clever neural network architectural design decisions.

These design decisions are made by humans to impose *priors* that are believed to be useful, and are *the* reason that neural networks have become effective over the last several years (along with increasing computational power). Although there are many examples of this (e.g., the attention mechanism [133], the ReLU [93] and SeLU [62] activation functions, residual connections [42], and batch normalization [52]) we will discuss the two most prominent examples of this:

- **Convolutional Neural Networks (CNNs)** [67] are perhaps the first instance of designing a neural network's architecture to match a problem domain. In the

domain of image classification, instead of processing the input image by throwing out all spatial locality and feeding the pixels of an image as uncorrelated floating point values, a convolutional neural network is specifically designed to make use of spatial locality. Specifically, the input is represented as a two-dimensional matrix and is operated on by a two-dimensional *convolution*. Performing a convolution has many benefits aside from preserving spatial locatlity, but most importantly it reduces the number of trainable weights and therefore reduces over-fitting.

- **Long Short-Term Memory (LSTM)** Neural Networks [46] cells are, after CNNs, the next most widely used example of architecture design. LSTMs are a form of recurrent neural networks used to process a sequence of values. Instead of performing a fully-connected operation on the hidden state, a LSTM implements multiple types of "gates" allowing the network to selectively *forget* pieces of information it had learned previously. This functionality significantly increases length of dependencies that a neural network can learn.

We draw inspiration from these approaches (and, specifically from LSTMs) to design a classifier that through its architectural design is provably robust to attack.

5.3 Problem Domain

5.3.1 Dataset

We obtained a dataset of 5.3 million queries made to VirusTotal over a period of 30 months. VirusTotal extracts static features (e.g., the set of libraries that are loaded) from each sample along with running them in a sandbox to extract dynamic features (e.g., the set of files that are actually accessed).

For each query, VirusTotal runs up to 50 different antivirus products to determine if the sample is malicious or benign. Consistent with prior work, we label a sample as malicious if at least 5 of these products reported malicious [56]. After running our evaluations, we re-ran our analysis using different thresholds for maliciousness (from 4 to 10) and again consistent with prior work [56] found our results were not significantly impacted.

Figure 5.1 shows a distribution of the data collection period. Roughly 45% of queries were for files determined to be malicious, and 55% benign. These queries cover 1.1 million distinct samples.

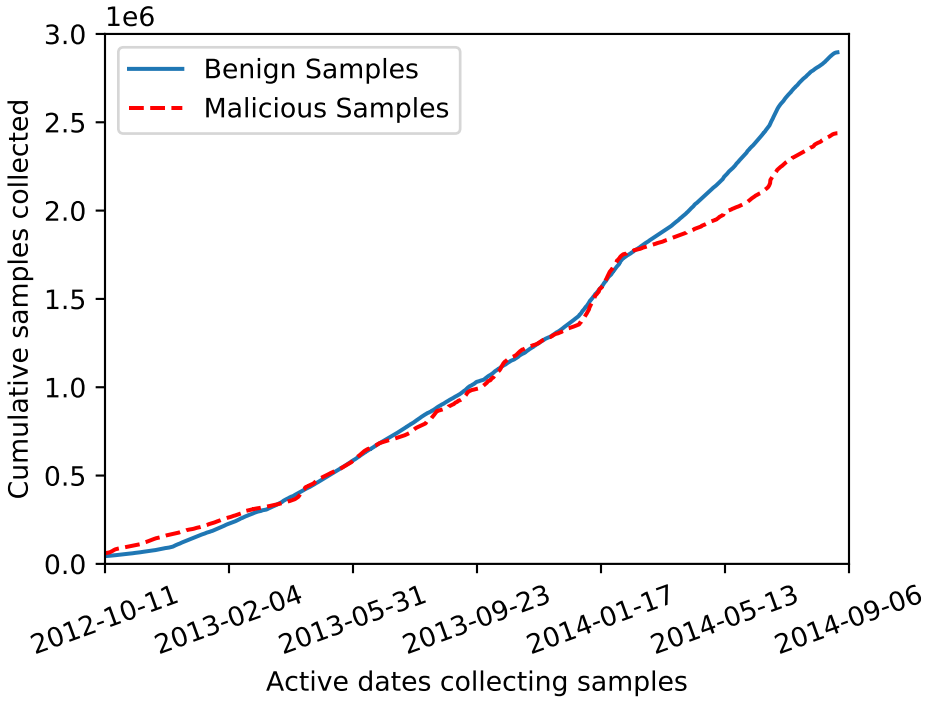

Figure 5.1: Plot of malware collection timeline; the ratio of malware to benign files is approximately evenly split, with cumulative growth rate nearly linear. Note the Y-axis is in millions.

5.3.2 Feature Set

For the remainder of this chapter, we focus exclusively on two different sets of dynamic features:

1. **Files and registry keys accessed** by the sample during execution. We track if the file was opened for reading, writing, or deleting, but do not track how much was read or written. We also drop all temporal locality (i.e., we do not track if one file was accessed before or after another file).
2. **System calls** made by the sample during execution. This consists of 44 different Windows system calls that VirusTotal believes have security relevance, along with whether or not they succeeded or failed.

5.3.3 Use Case

Because the features of our malware samples are dynamic, we must necessarily execute the potential malware samples before deciding if they are in fact malicious. We therefore target our classifier not as an antivirus tool that would execute on an end-user device, but rather as an classifier that can assist antivirus developers in identifying new malware or in determining which samples might warrant manual analysis from an expert.

5.3.4 Temporally Consistent Splitting

We must be very careful when creating training and testing data for malware classification. For most domains, this process is trivial: after a data collection period, the data is labeled and is then randomly partitioned into a training and a testing set arbitrarily. The accuracy on the test set will be representative of accuracy on new (previously-unseen) data.

In malware classification, this is not the case. Malware is often highly derivative, and evolves over time to evade prior detection approaches. For example, a malware author may create version 1 of some malware, and later modify it to make a version 2 so that this new version evades more antivirus detectors and is therefore generally harder to detect.

If we were to randomly partition the malware into training and testing sets, then it would be highly likely that for some malware samples, we would be *training* on version

2, but *testing* on version 1. This would make our classifier appear to be significantly more accurate than it otherwise could be in the real world.

Instead, the correct approach requires that we ensure that the oldest *training* sample is strictly younger than the earliest *testing* sample. This mirrors the actual deployment case of malware: at some time T, the classifier is trained using data up to time T but not after it. Then, we will ask how well the classifier is expected to behave well when looking at data after time T.

Previous work [56] demonstrated a **20 to 30 percentage point** decrease in accuracy when this proper temporal splitting is used over random cross validation, demonstrating its importance to accurately evaluate a malware classifier.

Unfortunately, almost no prior work evaluating neural networks on malware classification makes this distinction, making comparison with prior work difficult.

5.3.5 Ground Truth

Given that malware is inherently difficult to obtain ground truth for, we follow the approach of Kantchelian *et al.* [56] and make use of the fact that AV vendors typically only get better at detecting a specific file as malware as time goes on (and usually not worse). When a new sample is first seen, AV vendors initially may not be certain whether or not it is malicious. However, over time, humans are able to design better signatures that will detect samples not initially detected.

Therefore, we take the ground truth label for our malware to be the result of what the AV vendors label it as one month after it was first seen. In line with prior work, we consider a sample to be malicious if five or more of the classifiers label it as malware at this time [56].

This allows us to include all samples, and not just those which are confidently predicted the same way by a majority of AV vendors. (If we do as one prior study did, and remove all samples that are classified as malicious by between one and 30% of vendors, our true positive rate goes up by roughly 10 percentage points at most low false positive rates.)

5.3.6 Threat Model

An adversary would like to take a malicious sample, and modify it so that (a) the sample is now classified as benign, but (b) the essential (malicious) behavior of the sample remains the same.

In many other domains (such as image classification or speech recognition), an image or audio waveform is called an adversarial example if a human would give it a different label than a classifier. Because it is difficult to formalize "what a human would say", often times this is formalized by requiring that the distortion introduced is less than some threshold ϵ under some distance metric. For example, we may require that at most at most 20 pixels are changed, and rely on the fact that this (probably) does not change how a human would classify the image. This is clearly a sub-set of what actually is possible.

It is easier to formalize what is meant by an adversarial example on malware. As long as the file is classified as benign and still performs the essential malicious behavior, the adversary has succeeded. Exactly how much distortion has been introduced is not important. Even if we did try to measure distortion under some distance metric, even making one or two changes to a malware sample could completely change its behavior (e.g., by removing a necessary system call).

Therefore, we propose a threat model and distortion metric that is an underapproximation of what a true attacker could achieve, just as the formalization used for images (where we limit the distortion to be less than some threshold under ℓ_p-norm) is also an under-approximation. However, arguing security against that limited threat model is a useful first step towards the end goal of achieving robustness, under a more accurate threat model of what could actually be achieved.

This motivates our threat model of an **insertion adversary**: given a malware sample, an insertion adversary is only able to *inject new functionality* but can not remove or modify any existing functionality. Clearly no adversary is in the real world limited to this threat model: in practice a true adversary will be able to remove or re-write components of their malware sample in order to evade detection. However, we believe this threat model is interesting for two reasons:

1. **An insertion adversary can fool standard neural networks.** We show in Section 5.4.1 that even this limited adversary can easily fool malware classifiers that reach a high accuracy and reduce their accuracy to 0%, despite not being allowed to remove or modify any existing functionality.

2. **It is a necessary component of any complete classifier.** Any classifier which hopes to be robust in general must at least be robust against insertion attacks. As such, considering an insertion adversary is a useful first step in building a secure classifier.

While attacks that modify or remove existing functionality might not be easily automobile without breaking existing functionality, attacks that only require inserting new functionality are clearly trivial to implement. Given any piece of malware, it is

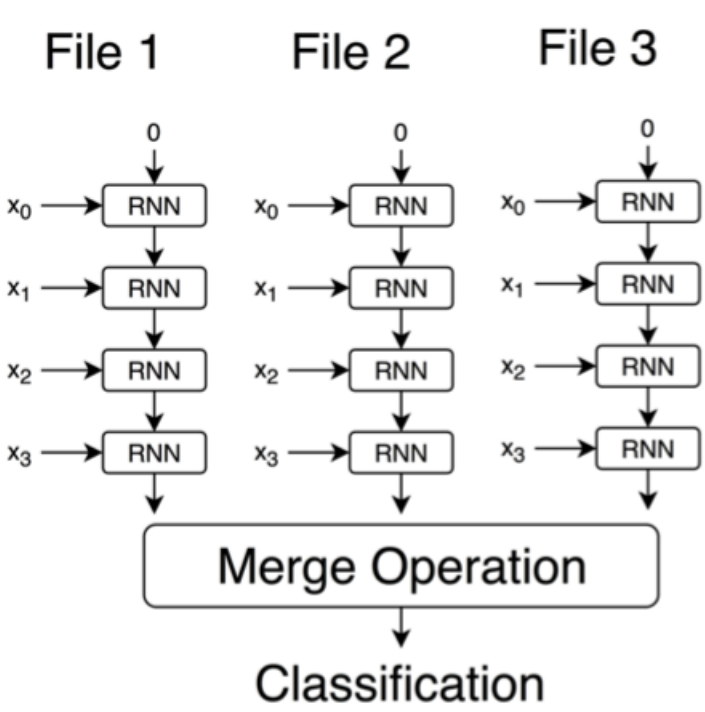

Figure 5.2: A diagram of our file path classifier. Each file path is read by a LSTM which labels each access as either malicious or benign. A merge operation then takes the predictions and outputs a prediction for the resulting sample.

easy to in an automated manner insert new behavior that has no side-effect other than to fool a classifier.

5.4 Case Study 1: File Access Classifier

We begin our study of robust classification with our first set of features: files accesses. This dataset consists of an unordered sequence of files and registry keys that are accessed by the sample; for the file accesses, we are given the mode (e.g., read/write/delete).

5.4.1 Initial Construction & Evaluation

We begin by constructing a neural network based classifier that receives as input these features and predicts whether or not the sample is malicious.

We treat each sample as a a collection of file or registry key access paths $\vec{x}$ where each x^i is represented directly as a raw sequence of bytes.

To classify a sample, we feed each file path x^i to a one-layer LSTM with 256 hidden units. We take the final state of the LSTM and apply a final linear layer to obtain a real number $g(x^i) = y^i$.

Now that we have multiple independent "maliciousness" predictions, we merge these predictions together by taking the mean. That is, the final prediction is to let

$$f(\vec{x}) = \text{sigmoid}\left(\frac{1}{N}\sum_{i=1}^{N} g(x^i)\right).$$

A diagram of this process is given in Figure 5.2.

Evaluation. We train this classifier on our dataset using 10 different temporal splits. We train using AdaGrad with a learning rate of 0.001 and a batch size of 32 for 10 epochs. This process takes 8 hours on a single commodity GPU.

As an implementation detail, to exercise the massively data-parallel nature of GPUs, we must require that all training examples access the same number of files and registry keys. To work around this issue, we select a large constant N (e.g., 400 in our case) and (a) for samples that access more than N files or registry keys, randomly sample N without replacement, and (b) for samples that access fewer than N files or registry keys, pad to N accesses with a special null path. Note that this step is only necessary for performance, and we have found our heuristic approach here does not significantly degrade accuracy compared to removing the padding operation (but is an order of magnitude faster to train). We do not need to change the inference process, as that is less computationally intensive.

We plot in Figure 5.3 the ROC curve of our classifier. We achieve a 55% true positive rate at 1% false positive rate given only these extremely limited set of features. This is in contrast to the 80% true positive rate of the anti-virus vendors surveyed in [56].

Robustness Analysis

We now turn to the question of evaluating the robustness of our trained classifier in adversarial settings. In particular, we ask if it is easy for an adversary to cause this classifier to misclassify a malware sample as benign. Initially, we assume the adversary has complete knowledge of the model parameters and all internal state, however we later relax this assumption.

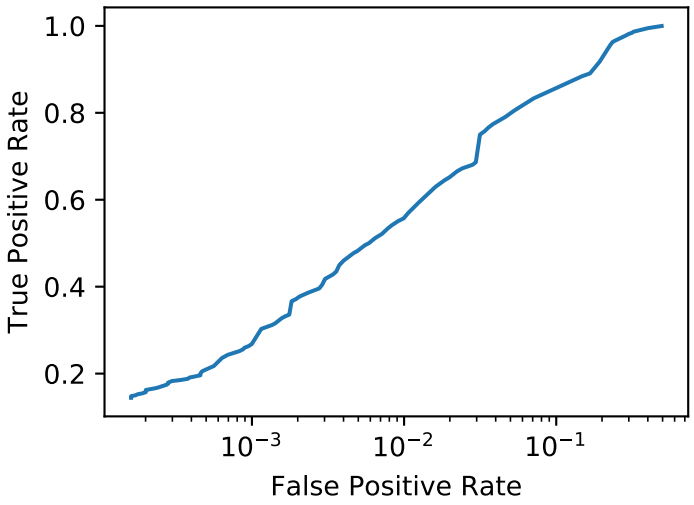

Figure 5.3: ROC curve of the file path and registry key access classifier; note the log-scaled x-axis.

Recall that we assume the highly limited setting of an adversary who can *only* insert new functionality, and can not remove existing functionality. Concretely, under this threat model for this domain, we allow the adversary to insert new file or registry key accesses but not to modify or remove existing ones. We also use as a feature whether any individual file access was opened for reading, writing, or deleting. We therefore restrict our insertion adversary to only being allowed to open files for reading: not writing or deleting. While it is true that we would be adding new functionality by deleting a file, it is not guaranteed that this operation can be performed in arbitrary settings.

Given this, there is only one reasonable attack method: insert as many benign accesses as possible until the sample becomes classified as benign.

Highly-Benign Accesses. To achieve this, we generate what we call highly-benign accesses: file or registry accesses that the classifier believes are indicative of benign samples.

For a given file path classifier $g : [0, 255]^* \to \mathbb{R}$, the *most benign* access is given by

$$x^* = \arg\min_{x \in [0,255]^*} g(x).$$

If we could perform this minimization process exactly, the optimal attack strategy would be to insert the access x^* in to the sample as many times as required.

However, identifying the most benign file access is a nontrivial optimization problem, as the optimization is performed over a discrete space high-dimensional space. We therefore approximate this problem and find "highly benign" accesses.

We approximate this minimization problem through greedy hill-climbing. Given an initial file or registry access x_0, we iteratively update

$$x_i = \underset{x:\mathcal{D}(x_{i-1},x_i)=1}{\arg\min} \; g(x_{i-1})$$

where we define $\mathcal{D}$ to be the edit distance between two strings. We terminate when either (a) the number of iterations exceeds some threshold or (b) the greedy hill climbing reaches a local minimum. This search procedure is clearly sub-optimal and greedy, but we find it is effective in generating highly benign files.

This leaves only the question of how we should initialize x_0 in our search. We initially try setting $x_0 =$ "", the empty string, however we find that greedy search quickly gets stuck in a local minimum. Instead, we take all files accessed during execution, and begin search from the top-100 most benign existing files or registry keys. This significantly improves the quality of our highly-benign access.

Figure 5.4 gives a distribution over how inserting highly benign accesses affects the true positive rate. An attacker who injects only three new file accesses can reduce the true positive rate to below 10%, and by inserting 10 new accesses it can be brought to 0%.

5.4.2 Improving Robustness with Existing Defenses

Before we design our own robust classifier, we ask: can state-of-the-art defense techniques be applied to malware classification to address the problem of evasion attacks? We evaluate two potential defenses that have seen success on other domains and find they are not effective for malware classification.

Randomization-based defenses. One simple method which has been proposed several times is to apply randomization [131] to the classifier, or to ensemble multiple classifiers together and report the accuracy on one of those (randomly selected) classifiers [83]. This prevents a malware developer from easily querying the classifier in order to construct highly benign accesses. These approaches have shown limited success on the image domain [4, 18] (where they were originally proposed) but we evaluate them here for completeness. We again find that these defenses are not effective due to the *transferability* [125] property of adversarial examples.

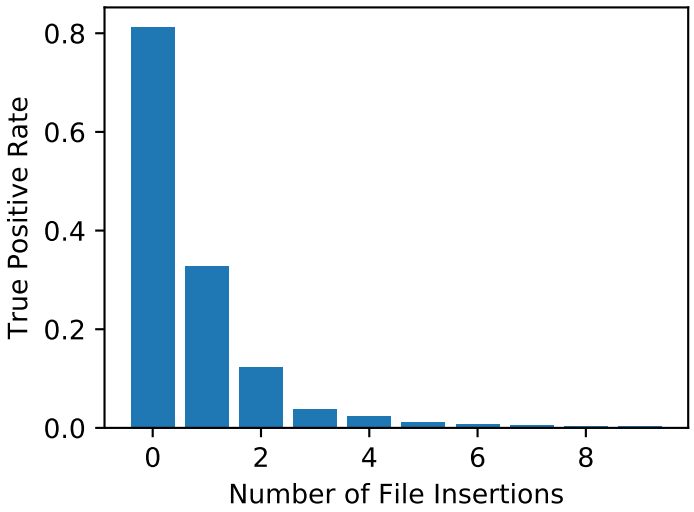

Figure 5.4: True positive rate of our malware classifier on the test set when each sample adds a small number of benign accesses. The true positive rate of the classifier falls below 10% with only three insertions.

This property, roughly stated, says that two classifiers (potentially using different architectures) trained independently (potentially on different datasets taken from the same underlying distribution) often learn the same decision boundaries. As such, samples that fool one classifier often *also fool the other* [100]. We find the transferability property holds true on malware, as well. We train two classifiers (with slightly different architectures) on two independent slices of our malware dataset (even numbered samples are provided to the first classifier as training data, odd numbered samples to the second). We then ask: are highly benign file paths on one classifier highly benign on the other?

We find the answer to this question is *yes*. While there is some divergence between the two classifiers decisions on individual files, when we evaluate the two classifiers on the access paths seen in the test set, we find a Pearson correlation coefficient of $p = 0.89$, indicating a highly-linear correlation. Moreover, if we add 5 new benign file accesses that are classified as highly benign by our first classifier, the true positive rate of our second classifier reaches below a 10% true positive rate.

Retraining-based defenses. The next style of defense we consider is *retraining-based* defenses. This style of defense is among the earliest studied [34]. The process of *adversarial training* [78] trains a classifier in repeated rounds. First, train an unsecured classifier $C_0(\cdot)$ on training data $\mathcal{X}_0$. Then, for each instance in the training

set, generate an adversarial example that fools the classifier, and add them to the training to obtain an augmented set $\mathcal{X}_1$; the classifier is then retrained on this dataset giving a classifier $C_1(\cdot)$. This process is repeated for N rounds until a final classifier $C_N(\cdot)$ is obtained.

This process is believed to be one of the most effective methods for training robust neural networks on the domain of images. However, we find this approach is not effective on the malware domain for several reasons:

- Generating highly-benign accesses is a slow process, requiring multiple rounds of optimization. Generating the adversarial training set alone for 1 epoch on highly-benign accesses would take over 100 GPU-hours.
- We consider an alternative more efficient way to generate adversarial examples. Instead of generating highly-benign accesses, we use the most-benign accesses that are already found in the training data. Unfortunately, retraining with these examples is not effective. We find the network quickly assigns these accesses a score of 0 (i.e., neither malicious nor benign) and pushes other accesses to be more benign, and the resulting network can still be attacked.

Even if retraining was significantly more effective and increased the number of file insertions that would be required from 3 to 3000, this would still not be sufficient: because adding new file accesses is essentially free for a malware developer, our evasion attack could be easily automated and the true positive rate of the classifier reduced to 0% with little effort.

5.4.3 Robust Classifier Design

Therefore, we instead design a new classifier to perform classification in a robust manner. The basic underlying idea of our monotonic classifier is simple. Whereas the prior classifier uses a RNN to output a maliciousness score, and takes the mean over all maliciousness scores, the robust classifier looks only at top maliciousness scores returned by the RNNs. It is intuitively simple to understand why this classifier is robust: because the classifier only uses the most malicious scores, inserting a new (less malicious) sample will not impact its classification.

In more detail, given the vector of file paths $\vec{x}$, we run a recurrent neural network $g(\cdot)$ over each sample to get a collection of maliciousness scores y^i, one for each file path x^i.

We then sort the elements of y^i to obtain a new ordered vector of elements z^i that satisfies $i \leq j \implies z^i \geq z^j$.

We then process $\vec{z}$ with a fully-connected neural network with non-negative weights and monotonic activation functions. Because fully-connected networks operate on fixed-size inputs, we take only the top K elements $\vec{z}^{1..K}$ as input to this network. In practice, we choose $K = 20$. We denote the fully-connected network $h(\vec{z})$. It is defined by initially assigning

$$h_0(\vec{z}) = \vec{z}$$

and then letting

$$h_i(\vec{z}) = \sigma(A_i h_{i-1}(\vec{z}) + b_i)$$

so that finally $h(\vec{z}) = h_N(\vec{z})$. Here, A_i are the weights (which we restrict to be non-negative) and b_i are the biases (which are unconstrained). We let σ be the ReLU activation function [93].

Theorem 1 *Given any sample represented as a vector of file access paths $\vec{x}$, inserting a new access x' can not decrease the maliciousness score of the final classifier.*

Proof (sketch). Given a collection of file access paths $\vec{x}$ so that $y^i = g(x^i)$ and z^i is a sorted vector of the largest K entries in y^i, the output of the neural network is given by $h(\vec{z})$. There are two cases:

Case 1: If the inserted access path x' had a maliciousness score $y' = g(x')$ that was not among the top-K, then it would not impact the output of $h(\cdot)$ because it would not be included an input $\vec{z}$ to the function.

Case 2: If the sample x' had a score y' that was among the top-K, then $y' > z^i$ for some i. Call this new vector $\vec{w}$ consisting of inserting y' into $\vec{z}$ at the correct position and removing the new-lowest element, so that

$$w^i = \begin{cases} z^i & \text{if } i < j \\ y' & \text{if } i = j \\ z^{i-1} & \text{if } i > j \end{cases}$$

In particular, we have that component-wise $w_i \geq z_i$ for all i.

We now must prove that $h(\vec{w}) \geq h(\vec{z})$. This is easy to see by induction on the number of layers in the network $h(\cdot)$, starting from the last layer.

If $h(\cdot)$ has one layer, we must show $\vec{w} \geq \vec{z}$ implies $h(\vec{w}) \geq h(\vec{z})$. We can show this by observing

$$h(\vec{w}) = \sigma(A \cdot \vec{w} + b) \geq \sigma(A \cdot \vec{z} + b) = h(\vec{z})$$

where here A is a row-vector and b is a scalar because the final output of the network $h(\cdot)$ is a single probability. Critically, this is only valid because A is non-negative.

Having proven the base case, we now prove the inductive case that $h_i(\vec{w}) \geq h_i(\vec{z})$ implies $h_{i+1}(\vec{w}) \geq h_{i+1}(\vec{z})$. This step is true for a nearly identical reason. We have

$$\begin{aligned}
h_i(\vec{w}) &\geq h_i(\vec{z}) && \implies \\
A_{i+1}h_i(\vec{w}) &\geq A_{i+1}h_i(\vec{z}) && \implies \\
A_{i+1}h_i(\vec{w}) + b_{i+1} &\geq A_{i+1}h_i(\vec{z}) + b_{i+1} && \implies \\
\sigma(A_{i+1}h_i(\vec{w}) + b_{i+1}) &\geq \sigma(A_{i+1}h_i(\vec{z}) + b_{i+1}) && \implies \\
h_{i+1}(\vec{w}) &\geq h_{i+1}(\vec{z})
\end{aligned}$$

And we have proven that this classifier is monotonic.

We are left with one final difficulty: this approach relies on processing the K most malicious files. What should we do if there are fewer than K file accesses? We observe that there is *only one* possible correct choice: to pad the remaining (non-existent) file accesses with the most-possible benign file access. Otherwise, a malware author could potentially reduce the maliciousness of a sample that made fewer than K accesses by inserting new, benign accesses.

In practice, we implement this by placing a cap on the output of the prediction function $h(\cdot)$ and enforce that it never output a value smaller than $-10,000$ and then whenever a malware sample performs fewer than K accesses we pad the remaining values with the value $-10,000$.

5.4.4 Training

Training this monotonic classifier is more difficult than training the non-monotonic classifier. Because the classifier only takes as input the top K values (we use 20) of the 400 file accesses, there is no gradient signal for 95% of file paths, making training difficult.

Worse still, for any random initialization, the file paths that are among the top-K are chosen randomly. So even though there is a gradient signal acting on these top-K values, there is no mechanism for selecting any one of the other file accesses to be among the top-K.

We resolve this difficulty by pre-training on the non-monotonic classifier: we initially train a non-monotonic classifier as described previously. We then remove the mean operation, and replace it with our monotonic merge operation. We fine-tune the weights of this classifier for another 5 epochs.

This process of training an easier-to-train classifier and using those weights for a

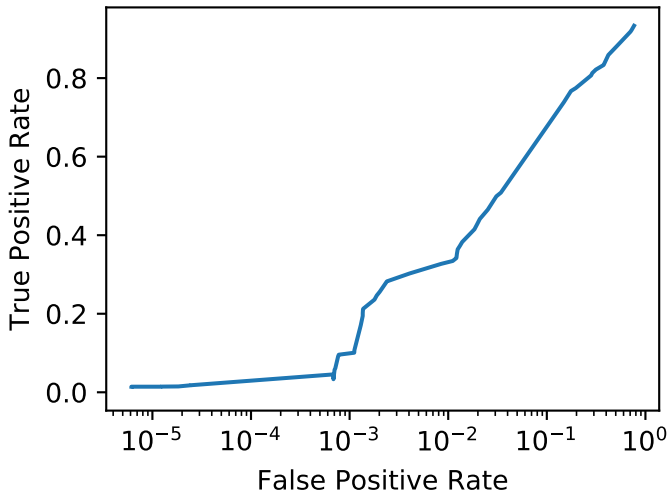

Figure 5.5: ROC curve of our robust, monotonic classifier. The true positive rate at 1% false positive rate is 33%.

different (harder-to-train) task is well studied in the domain of *transfer learning*, and we leverage its effectiveness here.

During the process of training, we observe that the resulting classifier assigns almost zero weight to the features numbered below 10, so that setting K greater than 10 does not improve accuracy. Upon investigating this, we discover the reason for this is that there are a sufficient number of malware samples that make only 10 accesses so that we are forced to pad with the most-benign access. This in turn causes any classifier which assigns any weight to this feature to label the malware sample as benign, and so the classifier dynamically learns how many features to examine.

5.4.5 Evaluation

We evaluate our classifier identically to the non-monotonic classifier, and find it achieves a 33% true positive rate at 1% false positive rate. While this is a significantly lower accuracy than the unsecured classifier, this classifier is *provably* robust against attack.

To verify the theory behind our approach is sound, we empirically verify through random sampling that inserting new file accesses never decreases the resulting maliciousness score of the neural network.

5.5 Case Study 2: System Call Classifier

We now switch feature sets and instead move to the problem of classifying a sample given access to the ordered sequence of system calls that it makes. The major difference between the system call classifier and the file access classifier is where we place our domain knowledge.

In the file access classifier, we apply neural networks to constructing an embedding. However, whereas file access paths can be seen as points from an extremely high dimensional dimensional space (255^N for a length N string), there are only a handful of possible system calls (44, in our case). This makes the embedding procedure—which accounted for nearly all of the effort in the file access classifier—trivial in the case of the system call classification.

Because of this, we focus our effort on what to do with the sequence of values. With the prior classifier, we either took the mean (in the non-monotonic case) or operated on predictions with a fully-connected neural network. In contrast, for the classifier we design in this section, we focus in detail on how to process a sequence of values in a monotonic manner.

5.5.1 Initial Construction & Evaluation

Given the sequential nature of the data, as a baseline, we apply a standard recurrent neural network to classify the sequence of system calls. We take the input as a the sequence of system calls along with whether or not the call succeeded, and feed this in to a Long Short-Term Memory (LSTM) neural network [46]. Specifically, we construct a one-hot encoded input with 88 possible values (for the 44 different system calls, each of which can either succeed or fail). We ignore all arguments to system these calls; we expect accuracy could be significantly improved if we included these arguments. To improve the efficiency of training, we pad each sequence to length 400 and train with mini-batch SGD (with a batch size of 32) for 10 epochs.

5.5.2 Evaluation.

Our model reaches an accuracy of 42% true positive rate at a 1% false positive rate. Figure 5.6 shows the ROC curve of the classifier.

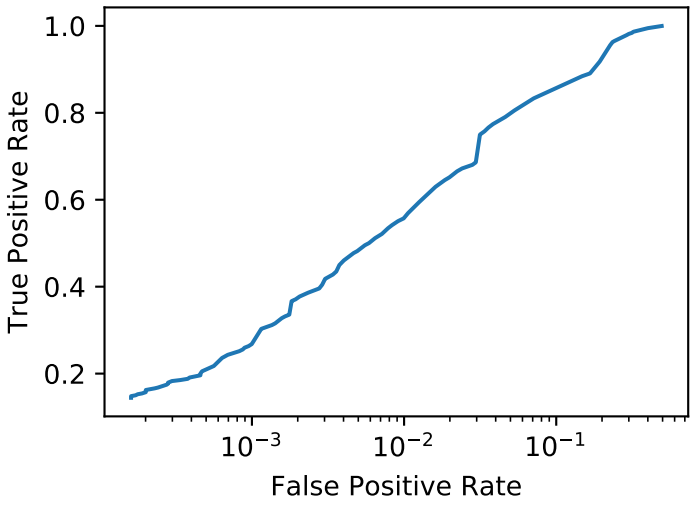

Figure 5.6: ROC curve of our non-robust system call classifier. The true positive rate at 1% false positive rate is 42%.

Robustness Analysis

While the previous classifier has respectable accuracy given the simplicity of the features available to it, we must now examine how it behaves in the presence of an adversary.

We develop a simple, greedy attack algorithm that applies *only* insertions to modify the sequence to a new sequence that is classified as benign. We observe that not all system calls can be inserted in a manner that preserves functionality. We prevent the adversary from inserting 10 of these[2] that may not be possible to insert in all situations without side effects. This leaves 34 system calls we allow the adversary to insert.

To determine the system calls to insert, we take the initial sequence $\vec{x} = x_0 x_1 \dots x_n$ and then from this construct new sequences $\vec{x}^{i,j}$ by inserting the new system call j at position i. Formally:

$$\vec{x}^{i,j}_k = \begin{cases} x_k & \text{if } k < i \\ j & \text{if } k = i \\ x_{k-1} & \text{if } k > i \end{cases}$$

[2]We do not allow the adversary to insert the ExitProcess, DeleteService, TerminateProcess, RegDeleteKeyA, ExitWindowsEx, DeleteFileW, RegDeleteKeyW, DeleteService, RegDeleteKeyA, or ReplaceFileW. Our robust classifier will still be robust even if these could be inserted.

Then, we choose the best (i, j) tuple as

$$(\hat{i}, \hat{j}) = \arg\min_{i,j} f(\vec{x}^{i,j})$$

and then greedily update

$$\vec{x} \leftarrow \vec{x}^{\hat{i},\hat{j}}$$

and repeat as long as $f(\vec{x}^{i,j}) < f(\vec{x})$.

This procedure results in a new sequence of system calls that is modified from the original by only inserting new system calls.

Attack Evaluation. The above attack succeeds with 100% success with an average of 23 system calls inserted. Note that this attack is not optimal — because system calls are inserted in a greedy manner, there may exist adversarial sequences with fewer insertions. However, even with this weak attack, the classifier we have constructed is completely broken.

5.5.3 Robust Classifier Design

Just as we are able to build a file path classifier with provable robustness to inserting new file accesses, we can also build a system call classifier with robustness to adversarial examples under a limited threat model. We begin with some definitions to make precise statements.

Definition 1 *An input x is a subsequence of the input x' (notationally, $x \prec x'$) if there is an increasing function $g : \mathbb{N} \to \mathbb{N}$ so that $x_i = x'_{g(i)}$ for all $i < |x|$*

Observe that under this definition of subsequence, an adversary who is limited to performing insertion operations can only transform a sequence of system calls x to a new sequence of system calls x' such that $x \prec x'$.

Definition 2 *We are given a classifier $f(\vec{x}) = y$ taking a sequence of system calls $\vec{x}$ and returning a probability y that the sequence is from a malicious sample. This classifier $f(\cdot)$ is* monotonic with respect to insertion *if for all inputs $x \prec x'$ it satisfies $f(x) \leq f(x')$.*

It is easy to see that a classifier is monotonic with respect to insertion if and only if it is perfectly robust against to insertion attacks. Therefore, we reduce our problem of constructing a classifier robust to insertion attacks to designing a monotonic classifier (with respect to insertion).

Neural Network Design

As background, a recurrent neural network $f(\cdot)$ operates on a sequence of inputs $\vec{x}$. It maintains a state vector s beginning with

$$s_0 = \vec{0}$$

and updating

$$s_i = g(s_{i-1}, x_i)$$

for each element x_i in $\vec{x}$ and for some update function $g(\cdot)$. Finally, the last state is processed and returned

$$f(x) = h(s_n)$$

for some final projection layer h.

In a *standard* recurrent neural network, we use the update rule

$$g(s, x) = \sigma(As + Bx + c)$$

for learned weights A, B, c and some nonlinearity σ. The projection final function is often defined as a simple linear combination of the output values.

$$h(s_n) = d \cdot s_n + e$$

Variants of RNNs (such as LSTMs or GRUs) differ in how they implement the update function $g(\cdot)$, but all take this general structure.

Given that all recurrent neural networks use this same overall architecture, we make use of it too. We are therefore tasked with designing the functions $g(\cdot)$ and $h(\cdot)$.

Recall that being monotonic requires that

$$x \prec x' \implies f(x) \leq f(x').$$

Let s_n be the final hidden state when the function g operates recursively on input x and let s'_n be the final hidden state on x'.

Being monotonic therefore requires

$$x \prec x' \implies h(s_n) \leq h(s'_n).$$

To simplify our monotonicity property, we further require $h(\cdot)$ itself is a monotonically increasing function, our goal is

$$x \prec x' \implies s_n \leq s'_n.$$

We achieve this by setting

$$s_i = s_i + \max(g(s_{i-1}, x_i), 0).$$

Concretely, we define

$$g(s_{i-1}, x_i) = As_{i-1} + Bx_i + c.$$

This update rule ensures that for all time steps $t < u$ we have $s_t^i \leq s_u^i$. This is a necessary, but not sufficient, condition for being monotonic. To address this, we additionally require that $A_{ij} \geq 0$, i.e., every element of A must be non-negative. This finally is sufficient to ensure monotonicity.

Theorem 2 *This classifier is monotonic with respect to insertion: for any two inputs $x \prec x'$, we have that $f(x) \leq f(x')$.*

Proof. Assume we are given two inputs x and x' where $x \prec x'$, and further there is only *one* element different between x and x'. We will prove that $f(x) \leq f(x')$. Inductively this guarantees the property for all $x \prec x'$.

Let s_i be the hidden state of the monotonic RNN after processing the first i elements of x, and s'_i be the hidden state of x'.

Let j be the index of the position where x and x' differ (i.e., x'_j is the inserted item so that $x_{1..j-1} = x'_{1..j-1}$ and $x_{j..n} = x'_{j+1..n+1}$). Because the prefixes are identical we have that

$$s_{j-1} = s'_{j-1}.$$

We now unroll one iteration of the RNN

$$s_j = s_{j-1} + \max(A \cdot s_{j-1} + B \cdot x_j + c, 0) \geq s_{j-1} = s'_{j-1}.$$

and it is easy to see that the new initial state is only larger. Because the states from here are monotonically increasing, the property holds true.

5.5.4 Toy Problem Experiments

To validate the potential efficacy of our classifier, and to confirm we have not overly-constrained its expressivity, we perform some toy experiments for simple classification tasks.

Task 1: Item Membership. As a first simple test, we construct a length-100 sequence s of random integers between 1 and 100. We then choose one number r (randomly) between 1 and 100 and train a classifier to answer "Does item r occur in the sequence s?"

We then train three classifiers on this task: a standard Recurrent Neural Network (RNN), a LSTM, and our monotonic classifier. We train using a batch size of 32 with RMSProp and perform grid search for the optimal learning rate between 0.1 and 0.0001. Initially, we observe that while the RNN and LSTM converge to 100% accuracy, our monotonic classifier remains stuck at random guessing, achieving 63% accuracy.

When we investigate why the classifier does not converge, we find it is due not to vanishing gradients, but exploding gradients. To give the intuition behind why this happens, observe that the update rule is essentially

$$s_{i+1} = s_i + As_i$$

(if the matrix B and bais c were removed) which has a closed-form solution of

$$s_i = (A + I)^i s_0$$

and so if the matrix $(A + I)$ has any eigenvalues greater than one in absolute value, s_i will increase exponentially.

Traditional recurrent neural networks avoid this by either initializing the weight matrix to be a Gaussian with small standard deviation centered around zero [31], or by constructing the matrix so that all eigenvalues are equal to 1 or -1 (i.e., so that the weight matrix is orthogonal) [112]. The first solution is not possible in our case because the entries A_{ij} must be non-negative. The second solution is also not possible, because the only way to make $(A + I)$ orthogonal so that $A_{ij} \geq 0$ would be to set A identically to 0.

Theorem 3 *If $A + I$ is orthogonal and $A_{ij} \geq 0$ for all i, j then $A_{ij} = 0$ for all i, j.*

Proof. It is easy to see this by contradiction. Assume that $A_{i'j'} > 0$ for some i', j', and that $A_{ij} \geq 0$ for all others.

Case 1: If $i' = j'$ then let $B = (I + A)(I + A)^T$. By definition of matrix multiplication

$$B_{i'j'} = \sum_k (I + A)_{i'k} \cdot (I + A)_{kj'}$$

because each entry of A is non-negative we can conclude

$$\sum_k (I+A)_{i'k} \cdot (I+A)_{kj'} \geq (I+A)_{i'j'} \cdot (I+A)_{i'j'}$$

and further

$$(I+A)_{i'j'} \cdot (I+A)_{i'j'} = (1+A_{i'j'}) \cdot (1+A_{i'j'}) > 1$$

because $A_{i'j'} > 0$. However this implies at least one entry along the diagonal of B is not equal to 1. If $(I+A)$ were orthogonal, then we must have that $(I+A)(I+A)^T = I$ (by definition), but clearly $B \neq I$ because at least one entry of B along the diagonal is greater than 1.

Case 2: If instead $i' \neq j'$ a similar argument holds. Let $B = (I+A)(I+A)^T$. Again let

$$B_{i'j'} = \sum_k (I+A)_{i'k} \cdot (I+A)_{kj'} \geq (I+A)_{i'j'} \cdot (I+A)_{i'j'}.$$

This time we observe

$$(I+A)_{i'j'} \cdot (I+A)_{i'j'} = (0+A_{i'j'}) \cdot (0+A_{i'j'}) > 0$$

again because $A_{i'j'} > 0$. Our contradiction this time observes that $B_{i'j'} \neq I_{i'j'}$. Therefore, in order to maintain orthogonality, we would require that $A \equiv 0$ to maintain an orthogonal matrix.

This has dramatic implications for our construction: it suggests that any nontrivial assignment of the weights A will result in a neural network whose hidden states increase exponentially quickly.

Improved training approach. In order to train a monotonic classifier on longer sequences, we must alter our training approach. Instead of training directly on the complete sequence, we begin by training our classifier on a smaller problem with shorter sequences. Then, once the classifier has converged to an initial solution and no longer encounters exponentially increasing states (or, more correctly, still has exponentially increasing states, but with a small enough constant that it does not inhibit useful gradient signal), we slowly increase the sequence length and begin to train it on longer sequences.

This approach is effective; we show the results of this experiment on this task in Figure 5.7.

Task 2: Subsequence Membership. Next, we perform a slightly more difficult task: subsequence membership. We fix a sequence q. Then, we train a classifier to

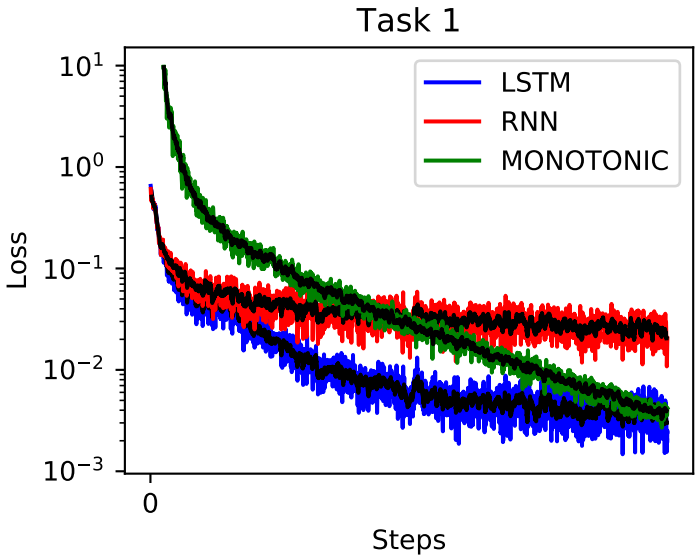

Figure 5.7: Training curves of three architectures on the simple task of determining single-item membership.

answer the question: given a sequence s, is $q \prec s$? To train the monotonic RNN, we begin with a short sequence s of length 20 and slowly increase its length until we reach length 100. The training curve is shown in Figure 5.8. The monotonic RNN reaches a lower loss than the RNN after $30,000$ iterations but does not reach the loss of the LSTM until much later (over $100,000$ iterations).

Task 3: Consecutive Subsequence Membership. Finally, we attempt a task that should be impossible to solve correctly: a consecutive version of the prior task. Instead of requiring that q be a subsequence of s, we now test if q is a *consecutive subsequence* of s. That is, there exists some i so that $s_{i,i+|q|} = q$.

A standard RNN and LSTM can solve this problem perfectly. However, expectedly, our monotonic classifier can not: because consecutive subsequence classification is non-monotonic with respect to insertion, the classifier can not encode the problem.

However, we did notice the accuracy of the classifier was better than random chance. When we investigate why this is the case we observe that this classifier just learned to perform a subsequence membership test. Because any (random) sequence s will probably not have q as a subsequence, returning "true" when q is any subsequence and "false" otherwise will reach better accuracy than random guessing.

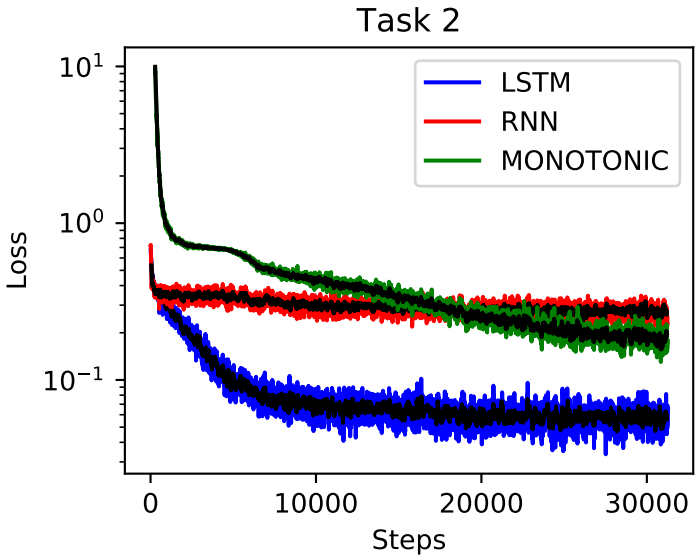

Figure 5.8: Training curves of three architectures on the task of determining subsequence membership.

5.5.5 Malware Evaluation

We now evaluate this classifier on the domain of malware classification. We find that the difficulties of training a monotonic classifier on the toy tasks only increase in difficulty for the malware task.

In the above simple tasks, we began by training our classifier on small sequences and then gradually scaled it to larger sequences. This is not easy to do for malware: there are no obvious "small sequences" to train on.

We experimented with a few approaches that were not effective which we briefly examine here:

- **Initially train on the samples that happen to be short.** While this approach is conceptually simple, we found it was ineffective for at least two reasons. First, there are not many short samples to begin with (and, as we found, there are not enough to train an accurate classifier). However, more importantly, we also discovered that training a non-monotonic LSTM on the short sequences does not generalize to the long sequences: they are sufficiently different that the classifier does not learn to classify the longer sequences correctly.
- **Randomly sub-sample from the system call sequences.** We can make system call sequence shorter by randomly deleting some of the system calls

that it makes, to make a subsequence of arbitrary length. When we attempt to train a classifier on this dataset, we do not succeed for a different reason: the malicious behavior is usually only a small fraction of the total behavior of a malware sample. By randomly sub-sampling the sequence system calls, we often remove the only malicious behavior that could be detected.

- **Use the non-monotonic LSTM to pick better sub-samples.** Instead of randomly sub-sampling the system call sequences, we could alternatively attempt to use the LSTM from the non-monotonic classifier to predict which substrings are the malicious sub-strings, and then not remove those. Unfortunately, while this is feasible, the monotonic classifier is unable to learn from the sequences: the LSTM picks up on substring membership, but the monotonic classifier can only operate on subsequences. Therefore, attempting to train the monotonic classifier on what worked for the LSTM was also ineffective.

How should we go about classifying the malware samples, then? Initially, we take a simple approach to validate that the classifier is at least able to represent the malware classification task. We validate this by manually setting the internal state of the neural network.

To choose the internal hidden state, we initially (through classical methods) identify sub-sequences that correspond to malware. For each file, we enumerate all subsequences of length up to 5 that occur most often in malware samples but do not occur often in benign samples. Then, we build into the hidden state of the classifier the weights which will make it a subsequence classifier. This approach is effective, and accurately classifies the malware samples.

While effective, this approach is not completely satisfying. The reason that neural networks are interesting is that they are able to discover features automatically. If we are forced to initialize their state to perform subsequence membership by hand, it is no longer satisfying. However, it does show that the architecture is expressive enough to solve the task; the challenge is to find an effective training procedure.

In order to perform effective training, instead of directly initializing the internal weights of the neural network based on the subsequences of malicious behavior, we construct a new training set consisting of examples of the sub-sequences we want the classifier to learn, and then train on these specifically crafted examples. This allows the classifier to reach high accuracy on the specific sub-sequences we would like it to learn, and hence classify the malware families with high accuracy.

Evaluation. Given this training approach, we now evaluate the accuracy of our classifier. The major limiting factor in our evaluation is that this monotonic classifier

is only effective at identifying malware when there are subsequences that separate it from benign samples. Because not all malware contains such subsequences, we limit our evaluation to specific malware families.

Performing a "fair" evaluation for evaluating the effectiveness of classifying a single family of malware is difficult. If we attempt to separate one family of malware from benign files, our classifier reaches extremely high accuracy: by limiting ourself to only one family of malware, it is much easier to construct a fingerprint of this one family than a fingerprint for "something malicious".

On the other hand, if we attempt to separate one family of malware from other malicious samples, we have dataset problems. Each AntiVirus vendor has its own set of family labels, and not all AntiVirus vendor agrees on when a sample is a member of a given family. When separating one family of malware from the benign data, this does not significantly impact results. However, when separating one family of malware from *other* malware, it is important that the other malware is not mislabeled.

We therefore study the former problem—separating malware from one family from the benign data—because it is well formed, even if it is easier than the original classification task.

We train this classifier on the complete set of benign files and the malware from the top 10 families by size (excluding generic catch-all families such "trojan" or various "unidentified" families). We report the ROC curve for this classifier in Figure 5.9. Observe that the accuracy is significantly higher on this task, largely because of the reduced complexity of the dataset.

However, we believe that this experiment demonstrates the potential efficacy of a monotonic classifier design.

Robustness Analysis. We finally empirically verify that the approach is effective and is robust to evasion attack through insertions. Again, we randomly insert new system calls and verify that indeed no insertion ever causes the classifier to decrease its maliciousness score.

5.6 Conclusion & Future Work

We argue that neural networks, which appear fundamentally vulnerable to test-time evasion attacks, can be applied to security-critical domains through carefully designing architectures that are provably secure under some restricted threat model.

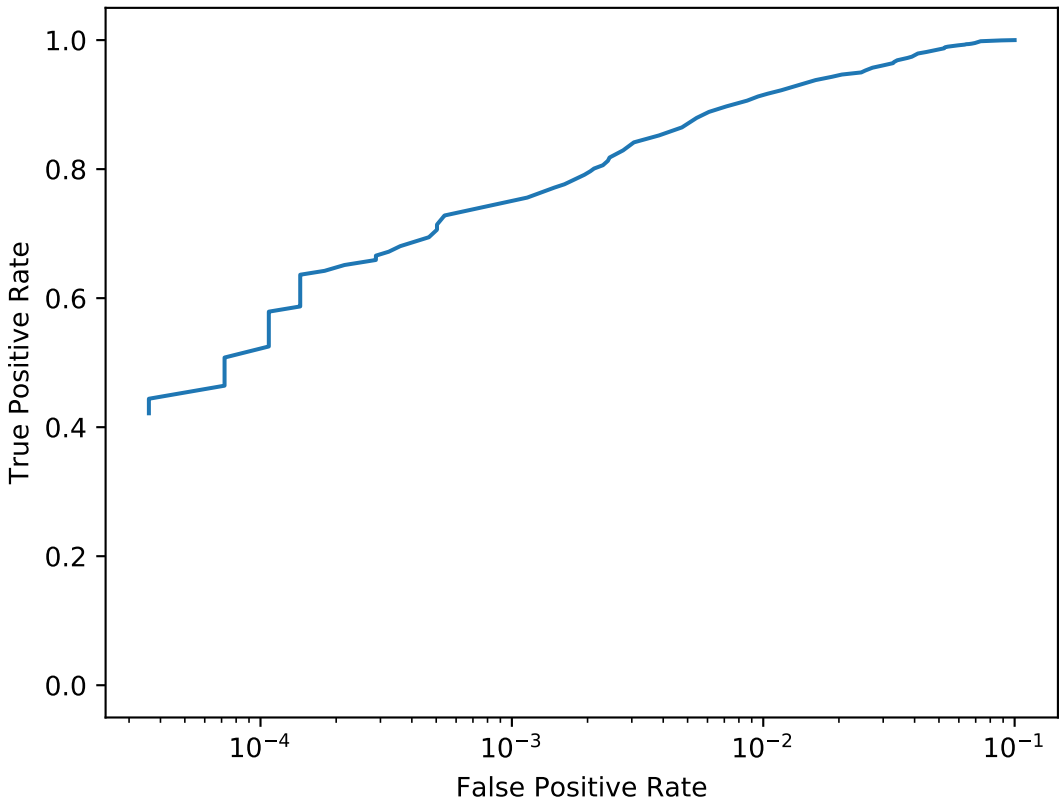

Figure 5.9: ROC curve of the monotonic classifier on a single malware family (averaged over 10 different families).

Through considering the domain of malware classification, we find that while traditional neural networks can achieve high accuracy on difficult datasets, the attacks we develop can automatically construct malware samples that behave identically but remain undetected. We mount these attacks under the threat model of an *insertion adversary*, who can only insert new functionality but can not remove or change any existing functionality. Then, we demonstrate that two new architectures allow us to develop robust classifiers that are not vulnerable to these insertion attacks.

We argue that in general, when necessary, neural networks can (and should) be designed for robustness. We have found that it is feasible to achieve perfect, provable robustness under restricted threat models. These significant security benefits come at a cost, however. Training our architecturally robust classifiers is more computationally expensive, more sensitive to initialization, and yields lower accuracy. We firmly believe that in security-critical domains, this is a worthwhile tradeoff. We encourage future work to follow in this direction designing classifiers that are provably robust even if they achieve lower accuracy in settings where security is important.

5.7 Conclusion & Future Work

We argue that neural networks, which appear fundamentally vulnerable to test-time evasion attacks, can be applied to security-critical domains through carefully designing architectures that are provably secure under some restricted threat model.

Through considering the domain of malware classification, we find that while traditional neural networks can achieve high accuracy on difficult datasets, the attacks we develop can automatically construct malware samples that behave identically but remain undetected. We mount these attacks under the threat model of an *insertion adversary*, who can only insert new functionality but can not remove or change any existing functionality. Then, we demonstrate that two new architectures allow us to develop robust classifiers that are not vulnerable to these insertion attacks.

We argue that in general, when necessary, neural networks can (and should) be designed for robustness. We have found that it is feasible to achieve perfect, provable robustness under restricted threat models. These significant security benefits come at a cost, however. Training our architecturally robust classifiers is more computationally expensive, more sensitive to initialization, and yields lower accuracy. We firmly believe that in security-critical domains, this is a worthwhile tradeoff. We encourage future work to follow in this direction designing classifiers that are provably robust even if they achieve lower accuracy in settings where security is important.

Chapter 6

Conclusion

The existence of adversarial examples limits the areas in which deep learning can be applied. It is therefore of utmost importance that we can effectively determine whether or not a given model is vulnerable to evasion attacks. In this dissertation, we propose applying powerful optimization-based methods to generate these adversarial examples. By systematically evaluating many possible attack approaches, we settle on one that can consistently find better adversarial examples than all existing approaches. We use this evaluation as the basis of our L_0, L_2, and L_∞ attacks.

Indeed, these same types of attack methods which are effective at generating adversarial examples on image classifiers are also effective at generating adversarial examples on speech recognition systems, further validating the generality of gradient-based methods for generating adversarial examples.

Unlike standard machine-learning tasks, where achieving a higher accuracy on a single benchmark is in itself a useful and interesting result, this is not sufficient for secure machine learning. We must consider how an attacker might react to any proposed defense, and evaluate whether the defense remains secure against an attacker who knows how the defense works. In evaluating 14 different defenses, we find that existing defenses lack thorough security evaluations. Our evaluations of these defenses expand on what is believed to be possible with constructing adversarial examples: we have shown that, so far, there are no known intrinsic properties that differentiate adversarial examples from regular inputs. We believe that constructing defenses to adversarial examples is an important challenge that must be overcome before these networks are used in potentially security-critical domains, and hope our attacks can bring us closer towards this goal.

Despite all of of these flaws, we argue that neural networks can be applied to security-critical domains through carefully designing architectures that are provably secure

under some restricted threat model.

Through considering the domain of malware classification, we find that while traditional neural networks can achieve high accuracy on difficult datasets, the attacks we develop can automatically construct malware samples that behave identically but are remain undetected. We mount these attacks under the threat model of an *insertion adversary*, who can only insert new functionality but can not remove or change any existing functionality. Then, we demonstrate that two new architectures allow us to develop robust classifiers that are not vulnerable to these insertion attacks.

We argue that in general, when necessary, neural networks can (and should) be designed for robustness. We have found that it is feasible to achieve perfect, provable robustness under restricted threat models. These significant security benefits come at a cost, however. Training our architecturally robust classifiers is more computationally expensive, more sensitive to initialization, and yields inferior accuracy. We firmly believe that in security-critical domains, this is a worthwhile tradeoff.

We hope this dissertation will provide a useful starting point for both evaluating and constructing neural networks robust in the presence of an adversary.

Bibliography

[1] Arlot, S., Celisse, A., et al. A survey of cross-validation procedures for model selection. *Statistics surveys 4* (2010), 40–79.

[2] Arnab, A., Miksik, O., and Torr, P. H. On the robustness of semantic segmentation models to adversarial attacks. *arXiv preprint arXiv:1711.09856* (2017).

[3] Athalye, A., Carlini, N., and Wagner, D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420* (2018).

[4] Athalye, A., Carlini, N., and Wagner, D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420* (2018).

[5] Athalye, A., Engstrom, L., Ilyas, A., and Kwok, K. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397* (2017).

[6] Barreno, M., Nelson, B., Joseph, A. D., and Tygar, J. The security of machine learning. *Machine Learning 81*, 2 (2010), 121–148.

[7] Barreno, M., Nelson, B., Sears, R., Joseph, A. D., and Tygar, J. D. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security* (2006), ACM, pp. 16–25.

[8] Behzadan, V., and Munir, A. Vulnerability of deep reinforcement learning to policy induction attacks. *arXiv preprint arXiv:1701.04143* (2017).

[9] Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[10] Bhagoji, A. N., Cullina, D., and Mittal, P. Dimensionality reduction as a defense against evasion attacks on machine learning classifiers. *arXiv preprint arXiv:1704:02654* (2017).

[11] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. Evasion attacks against machine learning at test time. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2013), Springer, pp. 387–402.

[12] Biggio, B., Nelson, B., and Laskov, P. Support vector machines under adversarial label noise. *ACML 20* (2011), 97–112.

[13] Biggio, B., Nelson, B., and Laskov, P. Poisoning attacks against support vector machines. *29th International Conference on Machine Learning* (2012).

[14] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[15] Borgwardt, K. M., Gretton, A., Rasch, M. J., Kriegel, H.-P., Schölkopf, B., and Smola, A. J. Integrating structured biological data by kernel maximum mean discrepancy. *Bioinformatics 22*, 14 (2006), e49–e57.

[16] Bourzac, K. Bringing big neural networks to self-driving cars, smartphones, and drones. `http://spectrum.ieee.org/computing/embedded-systems/bringing-big-neural-networks-to-selfdriving-cars-smartphones-and-drones`, 2016.

[17] Carlini, N., Mishra, P., Vaidya, T., Zhang, Y., Sherr, M., Shields, C., Wagner, D., and Zhou, W. Hidden voice commands. In *25th USENIX Security Symposium (USENIX Security 16), Austin, TX* (2016).

[18] Carlini, N., and Wagner, D. Magnet and "efficient defenses against adversarial attacks" are not robust to adversarial examples. *arXiv preprint arXiv:1711.08478* (2017).

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

[20] Cesare, S., and Xiang, Y. Classification of malware using structured control flow. In *Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing-Volume 107* (2010), Australian Computer Society, Inc., pp. 61–70.

[21] Christodorescu, M., and Jha, S. Static analysis of executables to detect malicious patterns. Tech. rep., DTIC Document, 2006.

[22] Cisse, M., Adi, Y., Neverova, N., and Keshet, J. Houdini: Fooling deep structured prediction models. *arXiv preprint arXiv:1707.05373* (2017).

[23] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint arXiv:1511.07289* (2015).

[24] Dahl, G. E., Stokes, J. W., Deng, L., and Yu, D. Large-scale malware classification using random projections and neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (2013), IEEE, pp. 3422–3426.

[25] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (2009), IEEE, pp. 248–255.

[26] Drucker, H., Wu, D., and Vapnik, V. N. Support vector machines for spam categorization. *IEEE Transactions on Neural networks 10*, 5 (1999), 1048–1054.

[27] Feinman, R., Curtin, R. R., Shintre, S., and Gardner, A. B. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410* (2017).

[28] Firoiu, V., Whitney, W. F., and Tenenbaum, J. B. Beating the world's best at super smash bros. with deep reinforcement learning. *arXiv preprint arXiv:1702.06230* (2017).

[29] Gandotra, E., Bansal, D., and Sofat, S. Malware analysis and classification: A survey. *Journal of Information Security 2014* (2014).

[30] Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., and Vázquez, E. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security 28*, 1 (2009), 18–28.

[31] Glorot, X., and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010), pp. 249–256.

[32] Gong, Y., and Poellabauer, C. Crafting adversarial examples for speech paralinguistics applications. *arXiv preprint arXiv:1711.03280* (2017).

[33] Gong, Z., Wang, W., and Ku, W.-S. Adversarial and clean data are not twins. *arXiv preprint arXiv:1704.04960* (2017).

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

[35] Graham, B. Fractional max-pooling. *arXiv preprint arXiv:1412.6071* (2014).

[36] Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning* (2006), ACM, pp. 369–376.

[37] Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. A kernel two-sample test. *Journal of Machine Learning Research 13*, Mar (2012), 723–773.

[38] Grosse, K., Manoharan, P., Papernot, N., Backes, M., and McDaniel, P. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280* (2017).

[39] Grosse, K., Papernot, N., Manoharan, P., Backes, M., and McDaniel, P. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435* (2016).

[40] Hannun, A. Sequence modeling with ctc. *Distill* (2017). https://distill.pub/2017/ctc.

[41] Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567* (2014).

[42] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.

[43] Hendrik Metzen, J., Genewein, T., Fischer, V., and Bischoff, B. On detecting adversarial perturbations. In *International Conference on Learning Representations* (2017). arXiv preprint arXiv:1702.04267.

[44] Hendrycks, D., and Gimpel, K. Early methods for detecting adversarial images. In *International Conference on Learning Representations (Workshop Track)* (2017).

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

[46] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[47] Hu, W., Hu, W., and Maybank, S. Adaboost-based algorithm for network intrusion detection. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 38*, 2 (2008), 577–583.

[48] Hu, W., and Tan, Y. Generating adversarial malware examples for black-box attacks based on gan. *arXiv preprint arXiv:1702.05983* (2017).

[49] Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284* (2017).

[50] Idika, N., and Mathur, A. P. A survey of malware detection techniques. *Purdue University 48* (2007).

[51] Ilyas, A., Engstrom, L., Athalye, A., and Lin, J. Query-efficient black-box adversarial examples. *arXiv preprint arXiv:1712.07113* (2017).

[52] Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[53] Jia, R., and Liang, P. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328* (2017).

[54] Jyothsna, V., Prasad, V. R., and Prasad, K. M. A review of anomaly based intrusion detection systems. *International Journal of Computer Applications 28*, 7 (2011), 26–35.

[55] Kabanga, E. K., and Kim, C. H. Malware images classification using convolutional neural network. *Journal of Computer and Communications 6*, 01 (2017), 153.

[56] Kantchelian, A., Tschantz, M. C., Afroz, S., Miller, B., Shankar, V., Bachwani, R., Joseph, A. D., and Tygar, J. D. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security* (2015), ACM, pp. 45–56.

[57] Kearns, M., and Li, M. Learning in the presence of malicious errors. *SIAM Journal on Computing 22*, 4 (1993), 807–837.

[58] Kephart, J. O., Sorkin, G. B., Arnold, W. C., Chess, D. M., Tesauro, G. J., White, S. R., and Watson, T. Biologically inspired defenses against computer viruses. In *IJCAI (1)* (1995), pp. 985–996.

[59] Kereliuk, C., Sturm, B. L., and Larsen, J. Deep learning and music adversaries. *IEEE Transactions on Multimedia 17*, 11 (2015), 2059–2071.

[60] Kinable, J., and Kostakis, O. Malware classification based on call graph clustering. *Journal in computer virology 7*, 4 (2011), 233–245.

[61] Kingma, D., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[62] Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems* (2017), pp. 972–981.

[63] Kolosnjaji, B., Zarras, A., Webster, G., and Eckert, C. Deep learning for classification of malware system call sequences. In *Australasian Joint Conference on Artificial Intelligence* (2016), Springer, pp. 137–149.

[64] Kos, J., Fischer, I., and Song, D. Adversarial examples for generative models. *arXiv preprint arXiv:1702.06832* (2017).

[65] Krizhevsky, A., and Hinton, G. Learning multiple layers of features from tiny images.

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

[67] LeCun, Y., Bengio, Y., et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks 3361*, 10 (1995), 1995.

[68] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature 521*, 7553 (2015), 436.

[69] LeCun, Y., Cortes, C., and Burges, C. J. The mnist database of handwritten digits, 1998.

[70] Li, X., and Li, F. Adversarial examples detection in deep networks with convolutional filter statistics. *arXiv preprint arXiv:1612.07767* (2016).

[71] Lippmann, R. P., and Cunningham, R. K. Improving intrusion detection performance using keyword selection and neural networks. *Computer Networks 34*, 4 (2000), 597–603.

[72] Liu, Y., Chen, X., Liu, C., and Song, D. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770* (2016).

[73] Lowd, D., and Meek, C. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining* (2005), ACM, pp. 641–647.

[74] Lowd, D., and Meek, C. Good word attacks on statistical spam filters. In *CEAS* (2005).

[75] Ma, W., Duan, P., Liu, S., Gu, G., and Liu, J.-C. Shadow attacks: automatically evading system-call-behavior based malware detection. *Journal in Computer Virology 8*, 1 (2012), 1–13.

[76] Maas, A. L., Hannun, A. Y., and Ng, A. Y. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML* (2013), vol. 30.

[77] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).

[78] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *International Conference on Learning Representations* (2018). accepted as poster.

[79] Makandar, A., and Patrot, A. Malware analysis and classification using artificial neural network. In *Trends in Automation, Communications and Computing Technology (I-TACT-15), 2015 International Conference on* (2015), IEEE, pp. 1–6.

[80] McLaughlin, N., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., Safaei, Y., Trickel, E., Zhao, Z., Doupe, A., et al. Deep android malware detection. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy* (2017), ACM, pp. 301–308.

[81] Mei, S., and Zhu, X. Using machine teaching to identify optimal training-set attacks on machine learners. In *AAAI* (2015), pp. 2871–2877.

[82] Melicher, W., Ur, B., Segreti, S. M., Komanduri, S., Bauer, L., Christin, N., and Cranor, L. F. Fast, lean and accurate: Modeling password guessability using neural networks. In *Proceedings of USENIX Security* (2016).

[83] Meng, D., and Chen, H. MagNet: a two-pronged defense against adversarial examples. In *ACM Conference on Computer and Communications Security (CCS)* (2017). arXiv preprint arXiv:1705.09064.

[84] Mishkin, D., and Matas, J. All you need is a good init. *arXiv preprint arXiv:1511.06422* (2015).

[85] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[86] Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., and Frossard, P. Universal adversarial perturbations. *arXiv preprint arXiv:1610.08401* (2016).

[87] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. Deepfool: a simple and accurate method to fool deep neural networks. *arXiv preprint arXiv:1511.04599* (2015).

[88] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. 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.

[89] Moser, A., Kruegel, C., and Kirda, E. Limits of static analysis for malware detection. In *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual* (2007), IEEE, pp. 421–430.

[90] Mozilla. Project deepspeech. `https://github.com/mozilla/DeepSpeech`, 2017.

[91] Munroe, R. Tasks. `https://xkcd.com/1425/`, 2014.

[92] Murad, K., Shirazi, S. N.-u.-H., Zikria, Y. B., and Ikram, N. Evading virus detection using code obfuscation. In *International Conference on Future Generation Information Technology* (2010), Springer, pp. 394–401.

[93] Nair, V., and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)* (2010), pp. 807–814.

[94] Nelson, B., Barreno, M., Chi, F. J., Joseph, A. D., Rubinstein, B. I., Saini, U., Sutton, C. A., Tygar, J. D., and Xia, K. Exploiting machine learning to subvert your spam filter. *LEET 8* (2008), 1–9.

[95] Nelson, B., and Joseph, A. D. Bounding an attackâĂŹs complexity for a simple learning model. In *Proc. of the First Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML), Saint-Malo, France* (2006).

[96] Nguyen, A., Yosinski, J., and Clune, J. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 427–436.

[97] Odén, A., and Wedel, H. Arguments for fisher's permutation test. *The Annals of Statistics* (1975), 518–520.

[98] Papernot, N., Carlini, N., Goodfellow, I., Feinman, R., Faghri, F., Matyasko, A., Hambardzumyan, K., Juang, Y.-L., Kurakin, A., Sheatsley, R., et al. cleverhans v2. 0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768* (2016).

[99] Papernot, N., and McDaniel, P. On the effectiveness of defensive distillation. *arXiv preprint arXiv:1607.05113* (2016).

[100] Papernot, N., McDaniel, P., and Goodfellow, I. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277* (2016).

[101] Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. Practical black-box attacks against deep learning systems using adversarial examples. *arXiv preprint arXiv:1602.02697* (2016).

[102] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)* (2016), IEEE, pp. 372–387.

[103] Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. Distillation as a defense to adversarial perturbations against deep neural networks. *IEEE Symposium on Security and Privacy* (2016).

[104] Park, Y., Reeves, D., Mulukutla, V., and Sundaravel, B. Fast malware classification by automated behavioral graph matching. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research* (2010), ACM, p. 45.

[105] Pascanu, R., Stokes, J. W., Sanossian, H., Marinescu, M., and Thomas, A. Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2015), IEEE, pp. 1916–1920.

[106] Perdisci, R., Dagon, D., Lee, W., Fogla, P., and Sharif, M. Misleading worm signature generators using deliberate noise injection. In *Security and Privacy, 2006 IEEE Symposium on* (2006), IEEE, pp. 15–pp.

[107] Petrov, S. Announcing syntaxnet: The worldâĂŹs most accurate parser goes open source. *Google Research Blog, May 12* (2016), 2016.

[108] Rieck, K., Holz, T., Willems, C., Düssel, P., and Laskov, P. Learning and classification of malware behavior. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2008), Springer, pp. 108–125.

[109] Rozsa, A., Rudd, E. M., and Boult, T. E. Adversarial diversity and hard positive generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (2016), pp. 25–32.

[110] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*, 3 (2015), 211–252.

[111] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*, 3 (2015), 211–252.

[112] Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120* (2013).

[113] Saxe, J., and Berlin, K. Deep neural network based malware detection using two dimensional binary program features. In *Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on* (2015), IEEE, pp. 11–20.

[114] Shaham, U., Yamada, Y., and Negahban, S. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432* (2015).

[115] Sharif, M., Bhagavatula, S., Bauer, L., and Reiter, M. K. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 1528–1540.

[116] Sharma, Y., and Chen, P.-Y. Attacking the madry defense model with L_1-based adversarial examples. *arXiv preprint arXiv:1710.10733* (2017).

[117] Shen, S., Jin, G., Gao, K., and Zhang, Y. APE-GAN: Adversarial Perturbation Elimination with GAN. *arXiv preprint arXiv:1707.05474* (2017).

[118] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of Go with deep neural networks and tree search. *Nature 529*, 7587 (2016), 484–489.

[119] Smith, S. W., et al. The scientist and engineer's guide to digital signal processing.

[120] Sommer, R., and Paxson, V. Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on* (2010), IEEE, pp. 305–316.

[121] Song, L., and Mittal, P. Inaudible voice commands. *arXiv preprint arXiv:1708.07238* (2017).

[122] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806* (2014).

[123] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, 1 (2014), 1929–1958.

[124] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the Inception architecture for computer vision. *arXiv preprint arXiv:1512.00567* (2015).

[125] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. *ICLR* (2013).

[126] Tesauro, G. J., Kephart, J. O., and Sorkin, G. B. Neural networks for computer virus recognition. *IEEE expert 11*, 4 (1996), 5–6.

[127] Valiant, L. G. A theory of the learnable. *Communications of the ACM 27*, 11 (1984), 1134–1142.

[128] Wang, Q., Guo, W., Zhang, K., Ororbia II, A. G., Xing, X., Liu, X., and Giles, C. L. Adversary resistant deep neural networks with an application to malware detection. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017), ACM, pp. 1145–1153.

[129] Warde-Farley, D., and Goodfellow, I. Adversarial perturbations of deep neural networks. *Advanced Structured Prediction, T. Hazan, G. Papandreou, and D. Tarlow, Eds* (2016).

[130] Wittel, G. L., and Wu, S. F. On attacking statistical spam filters. In *CEAS* (2004).

[131] Xie, C., Wang, J., Zhang, Z., Ren, Z., and Yuille, A. Mitigating adversarial effects through randomization. *International Conference on Learning Representations* (2018). accepted as poster.

[132] Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., Yu, D., and Zweig, G. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256* (2016).

[133] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A. C., Salakhutdinov, R., Zemel, R. S., and Bengio, Y. Show, attend and tell: Neural image caption generation with visual attention. In *ICML* (2015), vol. 14, pp. 77–81.

[134] Xu, W., Qi, Y., and Evans, D. Automatically evading classifiers. In *Proceedings of the 2016 Network and Distributed Systems Symposium* (2016).

[135] Zantedeschi, V., Nicolae, M.-I., and Rawat, A. Efficient defenses against adversarial attacks. *arXiv preprint arXiv:1707.06728* (2017).

[136] Zhang, G., Yan, C., Ji, X., Zhang, T., Zhang, T., and Xu, W. Dolphinatack: Inaudible voice commands. *CCS* (2017).

[137] Zhang, J., and Zulkernine, M. Anomaly based network intrusion detection with unsupervised outlier detection. In *Communications, 2006. ICC'06. IEEE International Conference on* (2006), vol. 5, IEEE, pp. 2388–2393.

[138] Zhang, J., Zulkernine, M., and Haque, A. Random-forests-based network intrusion detection systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 38*, 5 (2008), 649–659.