

Towards Evaluating the Robustness of Neural Networks

Nicholas Carlini and David Wagner
University of California, Berkeley

Abstract

We consider how to measure the robustness of a neural network against adversarial examples. We introduce three new attack algorithms, tailored to three different distance metrics, to find adversarial examples: given an image x and a target class, we can find a new image x' that is similar to x but classified differently. We show that our attacks are significantly more powerful than previously published attacks: in particular, they find adversarial examples that are between 2 and 10 times closer.

Then, we study defensive distillation, a recently proposed approach which increases the robustness of neural networks. Our attacks succeed with probability 200 higher than previous attacks against defensive distillation and effectively break defensive distillation, showing that it provides little added security. We hope our attacks will be used as a benchmark in future defense attempts to create neural networks that resist adversarial examples.

1. Introduction

Deep neural networks have become increasingly effective at many difficult machine-learning tasks. In the image recognition domain, they are able to recognize images with near-human accuracy [10], [9]. They are also used for speech recognition [5], natural language processing [1], and playing games [17], [13].

However, researchers have discovered that existing neural networks appear to be vulnerable to attack. Szegedy *et al.* [20] first noticed the existence of *adversarial examples* on image classification: it is often possible to transform an image by a small amount and thereby change how the image is classified. Often, the total amount of change required can be very small. This work has inspired research on how to harden neural networks against these kinds of attacks.

This raises a natural question: How shall we evaluate schemes for hardening neural networks? How can we tell whether they are in fact secure? We take a first step towards answering these questions. The obvious method for evaluating a proposed defense is to try to attack it with the attacks described in the literature. However, as we will see, existing attacks are good enough to break unhardened neural networks, but not necessarily effective enough for evaluating new defenses.

We introduce three attack algorithms that are significantly more effective than previous attacks, optimized to produce adversarial examples minimizing the L_0 , L_2 , and

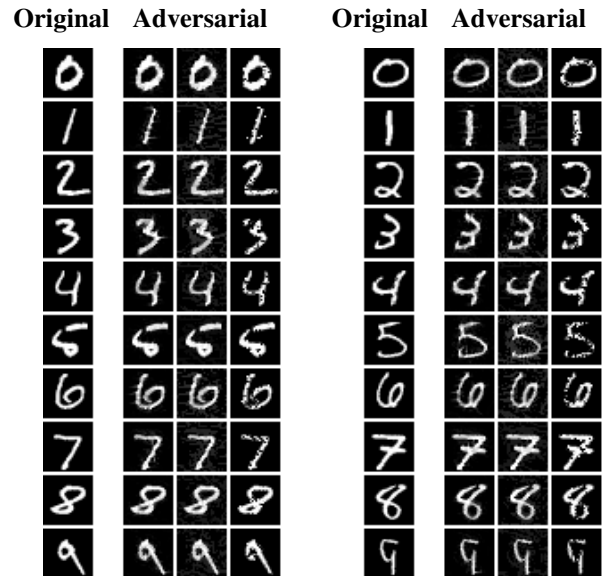


Figure 1. An illustration of our attacks. The leftmost column contains a handwritten digit from the MNIST data set. The next three columns show adversarial examples generated by our L_2 , L_∞ , and L_0 algorithms, respectively. The correct classification is self-evident. All misclassified instances share the same misclassified label of $l + 1 \pmod{10}$. Digits were chosen randomly from first 200 images in the test set.

L_∞ distance metrics respectively. Our algorithms are able to find small changes to an image to change it to be classified as any desired class. Our techniques find modifications that are 2–10 \times smaller than state-of-the-art attacks. In one extreme example, for the ImageNet classification [9] task, we can cause the Inception v3 [19] network to incorrectly classify images by changing only the lowest order bit of each pixel. Such changes are impossible to detect either visually or automatically.

Figure 1 shows examples of attacks generated using our techniques against a classifier for digit recognition. Attacks against CIFAR-10 and ImageNet classifiers have 10–100 \times less distortion.

Then, we turn to analyzing *defensive distillation* [15], a recent defense proposed for hardening neural networks against adversarial examples. The authors showed that defensive distillation defeats existing attack algorithms: defensive distillation reduces their success probability from 95% to 0.5%. Having built up a toolbox of powerful attacks, we apply them to defensive distillation. We discovered that defensive distillation is not secure: our attacks succeed in

finding adversarial examples for 100% of images. Distillation does add some value on top of unsecured networks, but does not significantly make adversarial examples more difficult to find. Thus, while defensive distillation stops previously published attacks, it cannot resist the more powerful attack techniques we introduce in this paper.

This case study illustrates the value of our attacks for evaluating proposed defenses. Also, the break of defensive distillation may be of independent interest, as it shows that defensive distillation adds little useful robustness; rather, it merely exploits blind spots in prior attacks. We then identify why defensive distillation stops existing attacks and why our attacks are able to succeed.

To enable others to more easily use our work to evaluate the robustness of other defenses, our adversarial image generation algorithms are available online at http://nicholas.carlini.com/code/nn_robust_attacks.

This paper makes the following contributions:

- We introduce three new attacks for the L_0 , L_2 , and L_∞ distance metrics. Our attacks are significantly more effective than previous approaches.
- We systematically evaluate the choice of objective function for finding adversarial examples, and show the choice can dramatically impact the efficacy of an attack.
- We apply these attacks to defensive distillation and discover that distillation provides little security benefit over un-distilled networks.

2. Background

2.1. Neural Networks and Notation

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 model parameters θ ; in our work the model is fixed, so for convenience we don't show the dependence on θ .

In this paper, we are focused on neural networks used as a (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 $y_1 + \dots + y_m = 1$. The output vector y is thus treated as a probability distribution, i.e., y_i is treated as the "probability" that 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*.

We use the notation from Papernot et al. [15]: we 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

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

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, sigmoid, ReLU [11], or ELU [2]. In this paper we focus primarily on networks that use a ReLU activation function, as it currently the most widely used activation function [19], [18], [12], [15]. We do not believe that the choice of non-linear activation function has any impact on the attacks we present.

A $h \times w$ -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 $[0, 1]$. 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.

2.2. Adversarial Examples

Szegedy *et al.* [20] first pointed out the existence of *adversarial examples*: given a valid input x , it is often possible to find a similar input x' such that $C^*(x) \neq C(x')$ yet x, x' are close according to some distance metric. In this paper we discuss three standard distance metrics for measuring similarity of images: L_0 distance, L_2 distance, and L_∞ distance. The L_p distance is $\|x - x'\|_p$, 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 which have been altered in an image.¹
- 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.
- 3) L_∞ distance measures the maximum change to any of the coordinates:

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

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.

1. In RGB images, there are three channels which 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.

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 from full-on to full-off will result in L_2 change of 1.0 and a L_∞ change of 1.0, not 255.)

While none of these distance measures is a perfect measure of perceptual similarity, empirically, when the distance between two images is sufficiently small under any of these metrics, the two images will be perceptually similar or even indistinguishable. Therefore, any defense that claims to be secure must, at minimum, demonstrate that it can resist attacks under each of these three distance metrics (and possibly others).

Next, we survey three attacks that have been proposed in the literature for generating adversarial examples, for the L_2 , L_∞ , and L_0 distance metrics, respectively.

2.3. L-BFGS

Szegedy *et al.* [20] 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:

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

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

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

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. Also, c is a constant. 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.

2.4. Fast Gradient Sign

The fast gradient sign [3] 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 and target label l , the fast gradient sign method sets

$$x' = x + \epsilon \cdot \text{sign}(\nabla \text{loss}_{F,l}(x)),$$

where ϵ is chosen to be sufficiently small so as to be undetectable. 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 maximize the loss function; then, it shifts all pixels simultaneously.

The fast gradient sign method treats the parameter ϵ as fixed and computes x' , which might or might not be an adversarial example. We propose the *gradient-sign ray method*, a variant that almost always finds an adversarial example: simply use line search on ϵ to find the minimum ϵ such that x' has a different classification from x .

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. As such, it should not be surprising if we can significantly improve on it.

2.5. Papernot's Attack

Papernot *et al.* introduced an attack for targeted misclassification [14]. Given x and a target class l , they try to find x' that is similar under the L_0 metric and such that $C(x') = l$. We give an intuitive summary of their attack, omitting implementation details that slightly increase accuracy. In Section 6.2 we provide more detail about their attack; for a complete description we encourage the reader to read their original paper [14].

Their method 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 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.

3. Our Approach

We now turn to our approach for constructing adversarial examples. To begin, we formally define the problem of finding an adversarial instance for an image x , as follows:

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

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 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_∞ .

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.

Formulation. The above formulation is difficult for existing algorithms to solve directly, as the constraint $C(x + \delta) \neq C^*(x)$ 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 $C(x + \delta) \neq C^*(x)$ whenever $f(x + \delta) \leq 0$, and otherwise $f(x + \delta) > 0$. There are many possible choices for f :

$$\begin{aligned} f_1(x') &= -\text{loss}_{F,s}(F(x')) + 1 \\ f_2(x') &= (F(x')_s - \max\{F(x')_i : i \neq s\})^+ \\ f_3(x') &= \text{relu}(F(x')_s - \max\{F(x')_i : i \neq s\}) - \log(2) \\ f_4(x') &= (F(x')_s - 0.5)^+ \\ f_5(x') &= -\log(2 - 2F(x')_s) \\ f_6(x') &= (Z(x')_s - \max\{Z(x')_i : i \neq s\})^+ \\ f_7(x') &= \text{relu}(Z(x')_s - \max\{Z(x')_i : i \neq s\}) - \log(2) \end{aligned}$$

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

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.

Then, instead of

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

we use an alternative formulation:

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

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

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

Choosing the constant c . When $c = 0$, gradient descent will not make any move away from the initial image. If we let $c \rightarrow \infty$, then the optimal solution to the constrained problem will also be the optimal solution to our revised formulation, for objective functions f_2 , f_4 , and f_6 .² 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. 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.

2. This is why we use the $(\cdot)^+$ operator in many of our objective functions.

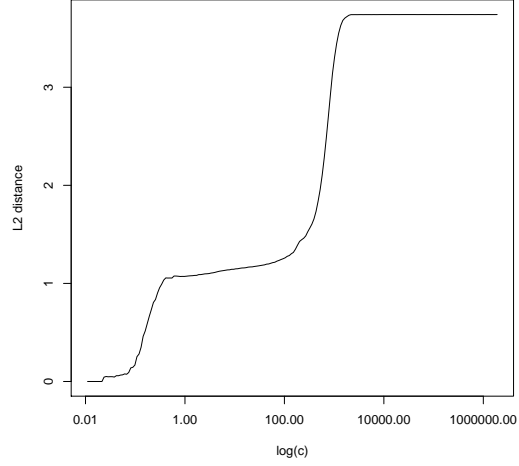


Figure 2. Sensitivity on the constant c . We plot the L_2 distance of the adversarial example computed by gradient descent as a function of c , for objective function f_2 . When $c < 1$, the attack rarely succeeds. The choice of c is stable from 1 to 100. After $c > 100$, the attack becomes less effective. When $c = 1000$, the constant is saturated and the distance remains constant.

We verify this by running our f_2 formulation for values of c spaced uniformly (on a log scale) from $c = 0.01$ to 1,000,000. We plot this line in Figure 2. As the figure shows, once c is sufficiently large, the exact value of c does not matter. Also, if we choose the smallest c such that $f(x^*) \leq 0$, we found that 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 .

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 actually update x_i on each iteration; instead, it clips the input to the minimization problem f . In other words, we replace $f(x + \delta)$ with $f(\min(\max(x + \delta, 0), 1))$, with the min and max taken component-wise.

	Change of Variable			Clipped Grad. Descent			Projected Grad. Descent		
	mean	stddev	prob	mean	stddev	prob	mean	stddev	prob
Cross entropy loss (f_1)	2.03	0.68	100%	1.71	1.56	100%	1.75	0.56	100%
Hinge loss on $F(\cdot)$ (f_2)	1.34	0.68	78%	1.18	0.47	79%	1.14	0.45	79%
Soft hinge on $F(\cdot)$ (f_3)	1.33	0.56	53%	1.23	0.51	54%	1.25	0.52	54%
Loss on $F(\cdot)$ (f_4)	1.58	0.66	73%	1.40	0.56	74%	1.43	0.56	74%
Loss on $\log(F(\cdot))$ (f_5)	1.11	0.40	100%	1.18	0.47	100%	1.22	0.40	100%
Hinge loss on $Z(\cdot)$ (f_6)	1.11	0.38	100%	1.16	0.40	100%	1.27	0.38	100%
Soft hinge on $Z(\cdot)$ (f_7)	1.13	0.38	100%	1.18	0.40	100%	1.14	0.39	100%

TABLE 1. 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.

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.

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 [8] optimizer almost exclusively, as we 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 nearly ten times more quickly than the others.

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.³

The results of this analysis are in Table 1. 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.

3. Adam converges to 95% of optimum within 1,000 iterations 92% of the time. We simply run it for 10,000 iterations at each step.

In fact, the worst performing objective function, cross entropy loss, is the approach that was most suggested in the literature previously [20], [16]. This is to be expected: the reason we include the hinge loss (of $(\cdot)^+$) in most of the objective functions is because once we have caused a misclassification, we do not want it to become possible to further minimize f by moving farther away from x . Using the cross entropy loss amplifies this more than any of the other objective functions, since the negative cross entropy loss is unbounded below. This means that when $F(x')_s$ is sufficiently small, a very large decrease in the negative cross entropy loss can be obtained by increasing the L_2 norm dramatically. This causes gradient descent to find very strong, but very distant, adversarial examples.

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 $\{0, 1, \dots, 255\}$. 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 i th pixel becomes $\lfloor 255(x_i + \delta_i) \rfloor$.

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.

Prior work has largely ignored the integrality constraints.⁴ 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.

4. One exception: Papernot's L_0 attack [14] handles this by only setting the output value to either 0 or 255.

4. Our Three Attacks

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 $t \neq C^*(x)$) and then search for w that solves

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

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 so that (a) we can perform targeted misclassification, and (b) 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.

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 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.

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 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.⁵ The intuition is that $g_i \cdot \delta_i$ tells us

5. Selecting the index i that minimizes δ_i is simpler, but it yields results with $1.5\times$ higher L_0 distortion.

how much reduction to $f(\cdot)$ we obtain from the i th 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 i th pixel, and we multiply this by how much the i th pixel has changed.

Papernot's attack *grows* a set — initially empty — of pixels which 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. Intuitively, our approach is superior because when shrinking the set, we are able to get more information about which pixels are not useful than guessing which pixels might become useful in the future.

Our algorithm is significantly more effective than Papernot's attack (see Section 5.1 for an evaluation). It is also efficient: we introduce optimizations that make it about as fast as our L_2 attack (with a single starting point). 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.

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

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

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

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

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.

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).

5. Attack Evaluation

Model Creation. We evaluate our attacks by training neural networks for MNIST, CIFAR-10, and ImageNet and then attacking them. We use standard architectures known to achieve high accuracy for training these models. We achieve 99.5% accuracy on MNIST, comparable to the state of the art. On CIFAR-10, we achieve 85% accuracy, compared to the state-of-the-art result of 95% [4], [18], [12]. Instead of training our own ImageNet model, we use the pre-trained Inception v3 network [19], which achieves 96% top-5 accuracy.

Because these models are not 100% accurate, it is more meaningful to consider targeted misclassification attacks than untargeted misclassification. For example, if a classifier is 80% accurate, it is trivial to find (untargeted) adversarial examples for the 20% of the inputs that are misclassified: simply return the original data point. To minimize this effect, we always consider *targeted* attacks: given an image x and a randomly chosen target label t (chosen uniformly at random from all labels other than the correct label), the attack goal is to find x' that will be classified as t .

MNIST. Our MNIST model has two 5x5 convolutional layers with max-pooling, followed by two fully connected layers. We regularize the model with L_2 regularization, and randomly shift the input by up to 3 pixels in each direction. We use the Adam [8] optimizer and train for 10 epochs. We achieve 99.5% accuracy on the test set.

CIFAR-10. Our model for CIFAR-10 has two 5x5 convolutional layers with max-pooling followed by three fully connected layers. We use batch normalization [7] and L2 regularization. We use the Adam optimizer and train for 50 epochs. We use data augmentation to increase accuracy of the classifier. 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. We achieve 85% accuracy on the test set.

5.1. Evaluation

Targeted Attack Evaluation. We compare our targeted attacks to the best results previously reported in prior publications, for each of the three distance metrics. In some cases, prior publications don’t report statistics on attack effectiveness, so we re-implemented their attacks. For each image in the test set, we pick a target t which is not the correct label uniformly at random. Results are found in Table 2.

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

	mean	ImageNet stddev	prob
Our L_0 attack	45.5	78.5	100%
Papernot <i>et al.</i>	109*	246*	95%*
Our L_∞ attack	0.004	0.000	100%
Fast Gradient Sign	0.008*	0.03*	100%*

TABLE 3. THE UNTARGETED VERSION OF OUR ATTACKS APPLIED TO IMAGENET. LOWER MEANS ARE BETTER. ALL ATTACKS ARE RE-IMPLEMENTED FOR THIS TASK.

examples with $2\times$ to $10\times$ lower distortion than the best previously published attacks, and succeed at least as often. Our L_∞ attacks are comparable in quality to prior work, but their success rate is much 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.

The multiple-starting-point gradient descent approach used for our L_2 attack decreased the distortion of the adversarial examples by 12% on average.

Untargeted Attack Evaluation. On the ImageNet dataset, Papernot’s attack and the fast gradient sign attack both fail to find adversarial examples. In order to provide a more complete analysis, we re-implement and re-run these attacks in an untargeted manner on ImageNet. For fast gradient sign, we use the untargeted version. For Papernot’s attack, we modify it slightly to work in the untargeted setting by changing the objective function. The results are given in Table 3.

All attacks achieve nearly 100% success rates at untargeted misclassification against ImageNet. Our attacks are significantly better: they achieve about a $2\times$ lower distortion than prior algorithms.

6. Defensive Distillation

Distillation was initially proposed as an approach to reduce a large model (the *teacher*) down to a smaller *distilled* model [6]. 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.

	MNIST			CIFAR			ImageNet		
	mean	stddev	prob	mean	stddev	prob	mean	stddev	prob
Our L_2 attack	1.51	1.30	100%	0.31	0.09	100%	0.37	0.05	100%
L-BFGS	3.24*	2.15*	100%*	3.13*	2.12*	83%*	0.80*	0.33*	82%*
Our L_0 attack	22.3	14.1	100%	4.15	1.92	100%	93.7	25.4	100%
Papernot <i>et al.</i>	35	?	97%	21.1*	17.7*	99%*			0%*
Our L_∞ attack	0.14	0.04	100%	0.013	0.005	100%	.005	0.004	100%
Fast Gradient Sign	0.12*	0.08*	62%*	0.019*	0.065*	24%*			0%*

TABLE 2. EVALUATION OF OUR ATTACKS, COMPARED TO THE PREVIOUS STATE OF THE ART ACROSS THREE DIFFERENT DATASETS. LOWER MEANS ARE BETTER. PREVIOUS ATTACKS DENOTED WITH AN ASTERISK ARE OUR RE-IMPLEMENTED VERSIONS OF THE ATTACK WHEN PRIOR PAPERS DID NOT REPORT THE RELEVANT RESULTS. FAST GRADIENT SIGN AND PAPERNOT’S ATTACK NEVER SUCCEEDED ON IMAGENET.

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 in its predictions.

As noted earlier, 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, 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.

6.1. Our Attacks

When we apply our attacks to defensively distilled networks, we find it provides only marginal value. We reimplement defensive distillation on MNIST as described [15] using the same model we used for our evaluation above.⁶

6. This is a slightly different architecture than as used in the original defensive distillation work.

Table 4 shows our attacks when applied to distillation. Compared to all of the previous attacks that fail to find adversarial examples, our attack succeeds with 100% success probability for each of the three distance metrics.

Distillation has added at least some security to the resulting network: the mean L_2 adversarial example is 20% larger, the mean L_0 adversarial example is 40% larger, and the mean L_∞ distance over twice as large. However, distillation does not add nearly as much security as argued initially. In an absolute sense, the distortion is so low that the adversarial examples are imperceptibly different from the original. As a result, the defensively distilled network is not secure.

6.2. 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 fails 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 simply 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 become ϵ at all locations except for the output class which has confidence $1 - 9\epsilon$ for some very small ϵ . 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 find a way to produce productive results 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.* [20], 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.

Papernot’s Attack fails for a completely different reason. This attack does not use the $F(\cdot)$ term in the network and instead only uses the values of $Z(\cdot)$. The derivatives here do not vanish because this is before the softmax. Rather, we found that the attack fails due to the implicit assumptions it makes. In order to greedily compute the most important pixel to change, it sets

$$\alpha_p = \frac{\partial Z(x)_t}{\partial x_p}$$

$$\beta_p = \left(\sum_{j=0}^9 \frac{\partial Z(x)_j}{\partial x_p} \right) - \alpha_p$$

so that α_p represents how much changing pixel p will change the target classification, and β_p represents how much changing p will change all other outputs. Then the algorithm picks

$$p^* = \arg \max_p (-\alpha_p \cdot \beta_p) \cdot (\alpha_p > 0) \cdot (\beta_p < 0)$$

so that $\alpha > 0$ (the target class is more likely), $\beta < 0$ (the other classes become less likely), and $-\alpha \cdot \beta$ is largest.

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 the α and β used above, because Papernot’s attack computes the gradient of the *input* to the softmax layer, α and β represent the size of the change at the input to the softmax layer. It is perhaps surprising that Papernot’s attack works on un-distilled networks: it treats all changes as being of equal importance, regardless of how much they change the softmax output. For example it will not change a pixel that increases α from 10 to 20 if it would also increase β from -100 to -80 , even if the latter is merely because some value would change from -40 to -20 . Thus, Papernot’s algorithm may fail to find an adversarial example even if one exists.

Recall from earlier 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 and causes

	Distilled Network		
	mean	stddev	prob
Our L_2 attack	1.84	1.65	100%
L-BFGS			0%*
Our L_0 attack	31.4	12.1	100%
Papernot <i>et al.</i>	110	?	0.5%
Our L_∞ attack	0.34	0.25	100%
Fast Gradient Sign			0%*

TABLE 4. THE EFFECTIVENESS OF OUR ATTACKS, AND PREVIOUSLY PUBLISHED ATTACKS, AGAINST DEFENSIVE DISTILLATION.

Papernot’s attack to fail spectacularly when applied to the distilled network.

Fast Gradient Sign fails at first for the same reason L-BFGS fails: the gradients are almost always zero. However, even if we use $\text{loss}_{F',l}(\cdot)$ as above to avoid the gradients being rounded to zero, the attack still does not succeed. One can imagine a naive variant of the fast gradient sign attack where instead of traveling in the direction of $d = \text{sign}(\nabla \text{loss}_{F,l}(x))$, we travel in a random direction r . It is then useful to ask the question: how much better is the direction d than a random direction r ? We find that on undistilled networks, d often performs 10 standard deviations above the expected performance of a random r ; however, when operating on a distilled network, d is only 1 standard deviation better than a random r .

7. Conclusion

The existence of adversarial examples limits the areas in which deep learning can be applied. It is an open problem to construct defenses that are robust to adversarial examples. In this paper, we propose powerful attacks that can be used to evaluate the efficacy of potential defenses. By systematically evaluating many possible attack approaches, we settle on one that can consistently find better adversarial examples than existing attacks. Applying our attacks against defensive distillation, we discover that defensive distillation is not secure.

We encourage those who create defenses to evaluate attacks on the best attacks available — not only the most efficient. We hope that our three attacks provide a baseline that can assist in evaluating future defenses.

References

- [1] ANDOR, D., ALBERTI, C., WEISS, D., SEVERYN, A., PRESTA, A., GANCHEV, K., PETROV, S., AND COLLINS, M. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042* (2016).
- [2] 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).
- [3] GOODFELLOW, I. J., SHLENS, J., AND SZEGEDY, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

- [4] GRAHAM, B. Fractional max-pooling. *arXiv preprint arXiv:1412.6071* (2014).
- [5] HINTON, G., DENG, L., YU, D., DAHL, G. E., MOHAMED, A.-R., JAITLY, N., SENIOR, A., VANHOUCKE, V., NGUYEN, P., SAINATH, T. N., ET AL. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.
- [6] HINTON, G., VINYALS, O., AND DEAN, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).
- [7] IOFFE, S., AND SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).
- [8] KINGMA, D., AND BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
- [9] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.
- [10] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (1998), 2278–2324.
- [11] MAAS, A. L., HANNUN, A. Y., AND NG, A. Y. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML* (2013), vol. 30.
- [12] MISHKIN, D., AND MATAS, J. All you need is a good init. *arXiv preprint arXiv:1511.06422* (2015).
- [13] 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).
- [14] 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.
- [15] 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).
- [16] SHAHAM, U., YAMADA, Y., AND NEGABAN, S. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432* (2015).
- [17] 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.
- [18] SPRINGENBERG, J. T., DOSOVITSKIY, A., BROX, T., AND RIEDMILLER, M. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806* (2014).
- [19] SZEGEDY, C., VANHOUCKE, V., IOFFE, S., SHLENS, J., AND WOJNA, Z. Rethinking the Inception architecture for computer vision. *arXiv preprint arXiv:1512.00567* (2015).
- [20] SZEGEDY, C., ZAREMBA, W., SUTSKEVER, I., BRUNA, J., ERHAN, D., GOODFELLOW, I., AND FERGUS, R. Intriguing properties of neural networks. *ICLR* (2013).