Adversarial Examples

A. M. Sadeghzadeh, Ph.D.

Sharif University of Technology
Computer Engineering Department (CE)
Data and Network Security Lab (DNSL)

April 11, 2023

Today's Agenda

Recap

Fast Gradient Sign Method (FGSM)

Let $\boldsymbol{\theta}$ be the parameters of a model, $\boldsymbol{x}$ the input to the model, y the label associated with $\boldsymbol{x}$ and $J(\boldsymbol{\theta}, \boldsymbol{x}, y)$ be the cost used to train the neural network.

We can linearize the cost function around the current value of $\boldsymbol{\theta}$, obtaining an optimal max-norm constrained perturbation of

$$\boldsymbol{\eta} = \epsilon \, sign(\nabla_x J(\boldsymbol{\theta}, \boldsymbol{x}, y))$$

We refer to this as the "**fast gradient sign method**" of generating adversarial examples.

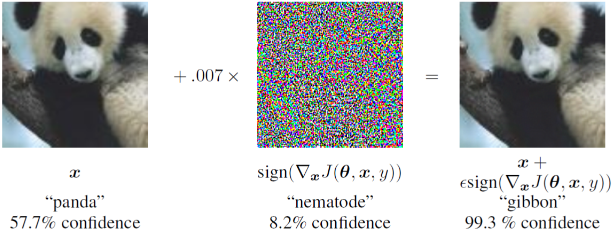


Figure 1: A demonstration of fast adversarial example generation applied to GoogLeNet (Szegedy et al., 2014a) on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image. Here our ϵ of .007 corresponds to the magnitude of the smallest bit of an 8 bit image encoding after GoogLeNet's conversion to real numbers.

Towards Evaluating the Robustness of Neural Networks

C&W Attack

2017 IEEE Symposium on Security and Privacy

Towards Evaluating the Robustness of Neural Networks

Nicholas Carlini David Wagner
University of California, Berkeley

Abstract

- The paper introduces **three new attacks** for the L_0, L_2, and L_∞ distance metrics. Proposed attacks are significantly more effective than previous approaches.

- As a case study, these attacks demonstrate that **defensive distillation** does not actually eliminate adversarial examples.
  - It constructs three new attacks (under three previously used distance metrics: L_0, L_2, and L_∞) that succeed in finding adversarial examples for 100% of images on defensively distilled networks.

- **Adaptive Adversary**
  - This case study illustrates the general need for better techniques to evaluate the robustness of neural networks.
  - The authors suggest that their attacks are a better baseline for evaluating candidate defenses.
  - Before placing any faith in a new possible defense, the authors suggest that designers at least check whether it can resist C&W attacks.

Threat Models

White-Box

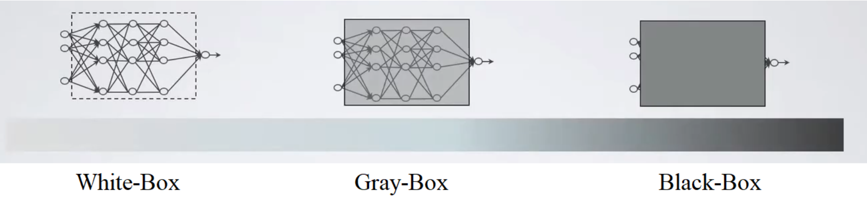
- The adversary has complete access to the algorithm, architecture, parameters, hyper-parameters, and input-output type of the target model.

Gray-Box

- The adversary has partial access to the algorithm, architecture, parameters, hyper-parameters, and input-output type of the target model.

Black-Box

- The adversary only has API access to the target model.



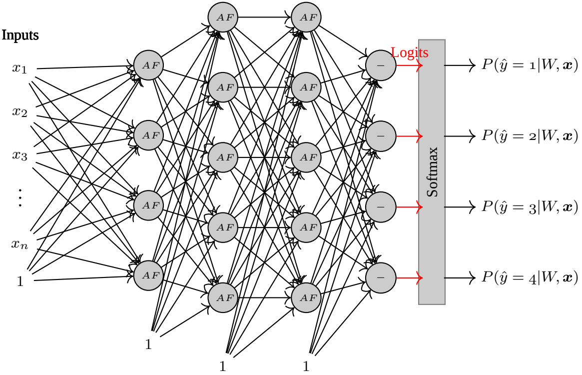
$(F$

All introduced attacks (including L-BFGS, FGSM, and C&W) have white-box threat model.

Notation

- A neural network is a function $F(x) = y$ that accepts an input $x \in \mathbb{R}^n$ and produces an output $y \in \mathbb{R}^m$. The classifier assigns the label $C(x) = \underset{i}{argmax}\ F(x)_i$ to the input x.

  Let $C^*(x)$ be the correct label of x.
- The inputs to the softmax function are called **logits** and denoted by $Z(x)$.



$$F(x) = Softmax(W^4(AF(W^3(AF(W^2(AF(W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4)$$

$$Z(x) = W^4(AF(W^3(AF(W^2(AF(W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4$$

Targeted and Untargeted Adversarial Examples

Untargetted attack

- The adversary wants to change the predication of the classifier to a wrong class.
  - Untargeted FGSM attack on clean data $(\boldsymbol{x}, y)$

$$\boldsymbol{x}_{adv} = \boldsymbol{x} + \epsilon.sign(\nabla_x J(W, \boldsymbol{x}, y))$$

Targeted and Untargeted Adversarial Examples

Untargetted attack

- The adversary wants to change the predication of the classifier to a wrong class.
  - Untargeted FGSM attack on clean data $(\boldsymbol{x}, y)$

$$\boldsymbol{x}_{adv} = \boldsymbol{x} + \epsilon.sign(\nabla_x J(W, \boldsymbol{x}, y))$$

Targeted attack

- The adversary wants to change the predication of the classifier to a given target class.
  - Targeted FGSM attack on clean data $(\boldsymbol{x}, y)$ for given target class t

$$\boldsymbol{x}_{adv} = \boldsymbol{x} - \epsilon.sign(\nabla_x J(W, \boldsymbol{x}, t))$$

Targeted and Untargeted Adversarial Examples

Untargetted attack

- The adversary wants to change the predication of the classifier to a wrong class.
  - Untargeted FGSM attack on clean data $(\boldsymbol{x}, y)$

$$\boldsymbol{x}_{adv} = \boldsymbol{x} + \epsilon.sign(\nabla_x J(W, \boldsymbol{x}, y))$$

Targeted attack

- The adversary wants to change the predication of the classifier to a given target class.
  - Targeted FGSM attack on clean data $(\boldsymbol{x}, y)$ for given target class t

$$\boldsymbol{x}_{adv} = \boldsymbol{x} - \epsilon.sign(\nabla_x J(W, \boldsymbol{x}, t))$$

C&W attacks focus on generating **targeted adversarial examples**.

Targeted Adversarial Examples

There are three different approaches to choose the target class

1. **Average Case**: select the target class uniformly at *random* among the labels that are not the correct label.
2. **Best Case**: perform the attack against all incorrect classes, and report the target class that was least difficult to attack (*the smallest size of perturbation*).
3. **Worst Case**: perform the attack against all incorrect classes, and report the target class that was most difficult to attack (*the largest size of perturbation*).

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.

L_P Norm

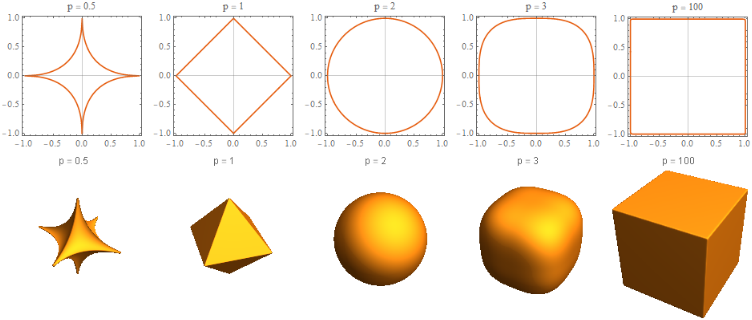
Let $p \geq 1$ be a real number, the P-norm (also called L_P-norm) of vector $\mathbf{x} = [x_1, x_2, \ldots, x_d]^T$ is

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

L_P Norm

Let $p \geq 1$ be a real number, the P-norm (also called L_P-norm) of vector $\mathbf{x} = [x_1, x_2, \ldots, x_d]^T$ is

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$$



The boundary of $\|\boldsymbol{x}\|_P = 1$
(Source).

Is L_P with $P < 1$ really a norm? The answer is no, because it violates the triangle inequality (See Convex Optimization by Stephen Boyd).

L_P Norm

The L_P distance is written $\|x - x'\|_P$, where $x, x' \in \mathbb{R}^n$ and the P-norm $\|.\|_P$ is defined as

$$\|x - x'\|_p = \left(\sum_{i=1}^{n} |x_i - x'_i|^p \right)^{1/p}$$

L_P Norm

The L_P distance is written $\|x - x'\|_P$, where $x, x' \in \mathbb{R}^n$ and the P-norm $\|.\|_P$ is defined as

$$\|x - x'\|_p = \left(\sum_{i=1}^{n} |x_i - x'_i|^p \right)^{1/p}$$

- ***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.

L_P Norm

The L_P distance is written $\|x - x'\|_P$, where $x, x' \in \mathbb{R}^n$ and the P-norm $\|.\|_P$ is defined as

$$\|x - x'\|_p = \left(\sum_{i=1}^{n} |x_i - x'_i|^p \right)^{1/p}$$

- $\boldsymbol{L_0}$ **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.

- $\boldsymbol{L_2}$ **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.

L_P Norm

The L_P distance is written $\|x - x'\|_P$, where $x, x' \in \mathbb{R}^n$ and the P-norm $\|.\|_P$ is defined as

$$\|x - x'\|_p = \left(\sum_{i=1}^{n} |x_i - x'_i|^p \right)^{1/p}$$

- $\boldsymbol{L_0}$ **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.
- $\boldsymbol{L_2}$ **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.
- $\boldsymbol{L_\infty}$ **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.

The approach

The formal definition of finding adversarial example for clean sample x is as follows

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

Where x and C is fixed, t is the target class, and the goal is to find δ that minimizes $\mathcal{D}(x, x+\delta)$.

The approach

The formal definition of finding adversarial example for clean sample x is as follows

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

Where x and C is fixed, t is the target class, and the goal is to find δ that minimizes $\mathcal{D}(x, x+\delta)$.

The above formulation is difficult for existing algorithms to solve directly, as the constraint $C(x+\delta) = t$ is highly non-linear. Therefore, the attack uses a different formulation that is better suited for optimization.

The approach

The formal definition of finding adversarial example for clean sample x is as follows

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

Where x and C is fixed, t is the target class, and the goal is to find δ that minimizes $\mathcal{D}(x, x+\delta)$.

The above formulation is difficult for existing algorithms to solve directly, as the constraint $C(x+\delta) = t$ is highly non-linear. Therefore, the attack uses a different formulation that is better suited for optimization.

We define an objective function f such that $C(x+\delta) = t$ if and only if $f(x+\delta) \leq 0$. Now, we have a new formulation for generating adversarial examples

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

The approach

The formal definition of finding adversarial example for clean sample x is as follows

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

Where x and C is fixed, t is the target class, and the goal is to find δ that minimizes $\mathcal{D}(x, x+\delta)$.

The above formulation is difficult for existing algorithms to solve directly, as the constraint $C(x+\delta) = t$ is highly non-linear. Therefore, the attack uses a different formulation that is better suited for optimization.

We define an objective function f such that $C(x+\delta) = t$ if and only if $f(x+\delta) \leq 0$. Now, we have a new formulation for generating adversarial examples

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

Using generalized Lagrange function, C&W attacks use the alternative formulation:

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

where $c > 0$ is a suitably chosen constant.

The approach

After instantiating the distance metric $\mathcal{D}$ with an L_P norm, the problem becomes

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

The approach

After instantiating the distance metric $\mathcal{D}$ with an L_P norm, the problem becomes

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

There are many possible choices for f:

$$f_2(x') = (\max_{i \neq t}(F(x')_i) - F(x')_t)^+$$

$$f_6(x') = (\max_{i \neq t}(Z(x')_i) - Z(x')_t)^+$$

$$f_3(x') = softplus(\max_{i \neq t}(F(x')_i) - F(x')_t) - \log(2)$$

$$f_7(x') = softplus(\max_{i \neq t}(Z(x')_i) - Z(x')_t) - \log(2)$$

$$f_1(x') = -loss_{F,t}(x') + 1$$

$$f_4(x') = (0.5 - F(x')_t)^+$$

$$f_5(x') = -\log(2F(x')_t - 2)$$

where $(e)^+ = max(e,0)$, $softplus(x) = log(1+exp(x))$, and $loss_{F,t}(x)$ is the cross entropy loss for x.

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.

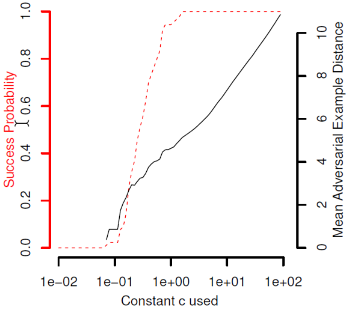


Fig. 2. Sensitivity on the constant c. We plot the L_2 distance of the adversarial example computed by gradient descent as a function of c, for objective function f_6. When $c < .1$, the attack rarely succeeds. After $c > 1$, the attack becomes less effective, but always succeeds.

Choosing the constant c - Binary Search

```
BINARY_SEARCH_STEPS = 9  # number of times to adjust the constant with binary search
INITIAL_CONST = 1e-3     # the initial constant c to pick as a first guess

# set the lower and upper bounds accordingly
lower_bound = np.zeros(batch_size)
CONST = np.ones(batch_size)*self.initial_const
upper_bound = np.ones(batch_size)*1e10

# adjust the constant as needed
for e in range(batch_size):
    if compare(bestscore[e], np.argmax(batchlab[e])) and bestscore[e] != -1:
        # success, divide const by two
        upper_bound[e] = min(upper_bound[e],CONST[e])
        if upper_bound[e] < 1e9:
            CONST[e] = (lower_bound[e] + upper_bound[e])/2
    else:
        # failure, either multiply by 10 if no solution found yet
        #          or do binary search with the known upper bound
        lower_bound[e] = max(lower_bound[e],CONST[e])
        if upper_bound[e] < 1e9:
            CONST[e] = (lower_bound[e] + upper_bound[e])/2
        else:
            CONST[e] *= 10
```

Box constraints

To ensure the modification yields a valid image, we have a constraint on δ: $x_i + \delta_i \in [0, 1]$ for all i. In the optimization literature, this is known as a **box constraint**.

Box constraints

To ensure the modification yields a valid image, we have a constraint on δ: $x_i + \delta_i \in [0, 1]$ for all i. In the optimization literature, this is known as a **box constraint**.

There are three different methods of approaching this problem.

1. **Projected gradient descent**
2. **Clipped gradient descent**
3. **Change of variables**

Box constraints

To ensure the modification yields a valid image, we have a constraint on δ: $x_i + \delta_i \in [0, 1]$ for all i. In the optimization literature, this is known as a **box constraint**.

There are three different methods of approaching this problem.

1. **Projected gradient descent**
   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**
3. **Change of variables**

Box constraints

To ensure the modification yields a valid image, we have a constraint on δ: $x_i + \delta_i \in [0, 1]$ for all i. In the optimization literature, this is known as a **box constraint**.

There are three different methods of approaching this problem.

1. **Projected gradient descent**
2. **Clipped gradient descent**

   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

     $$f(min(max(x + \delta, 0), 1))$$

     where 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**

Box constraints

To ensure the modification yields a valid image, we have a constraint on δ: $x_i + \delta_i \in [0, 1]$ for all i. In the optimization literature, this is known as a **box constraint**.

There are three different methods of approaching this problem.

1. **Projected gradient descent**
2. **Clipped gradient descent**
3. **Change of variables**

   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.

Evaluation

To choose the optimal c, we perform 20 iterations of binary search over c. For each selected value of c, we run 10000 iterations of gradient descent with the Adam optimizer.

	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 III

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

L_2 Attack

Given x, we choose a target class t (such that we have $t \neq C^*(x)$) and then search for w that solves

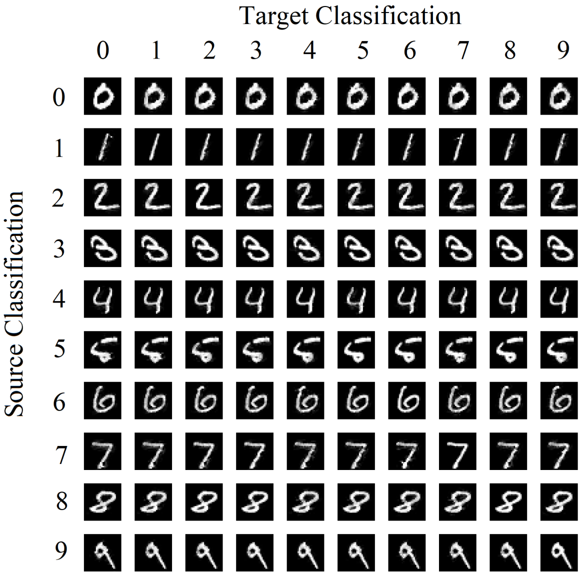
$$\text{minimize} \quad \|\frac{1}{2}(tanh(w)+1)-x\|_2^2 + c.f(\frac{1}{2}tanh(w))$$

with f defined as

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

The parameter κ encourages the solver to find an adversarial instance x' that will be classified as class t with high confidence. κ is 0 in the experiments.

L_2 Attack


Target Classification
0 1 2 3 4 5 6 7 8 9
Source Classification
0 1 2 3 4 5 6 7 8 9

L_0 Attack

The $\boldsymbol{L_0}$ **distance metric is non-differentiable** and therefore is ill-suited for standard gradient descent. Instead, An iterative algorithm is used in each iteration.

L_0 Attack

The $\boldsymbol{L_0}$ **distance metric is non-differentiable** and therefore is ill-suited for standard gradient descent. Instead, An iterative algorithm is used in each iteration.

- The algorithm **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.
  - It uses L_2 attack to identify which pixels are unimportant.
- **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.

L_0 Attack

The $\boldsymbol{L_0}$ **distance metric is non-differentiable** and therefore is ill-suited for standard gradient descent. Instead, An iterative algorithm is used in each iteration.

In more detail, on each iteration

- L_2 attack is conducted on the pixels in the **allowed set**.

L_0 Attack

The $\boldsymbol{L_0}$ **distance metric is non-differentiable** and therefore is ill-suited for standard gradient descent. Instead, An iterative algorithm is used in each iteration.

In more detail, on each iteration

- L_2 attack is conducted on the pixels in the **allowed set**.
- Let δ be the solution returned from L_2 attack on input image x, so that $x + \delta$ is an adversarial example.

L_0 Attack

The $\boldsymbol{L_0}$ **distance metric is non-differentiable** and therefore is ill-suited for standard gradient descent. Instead, An iterative algorithm is used in each iteration.

In more detail, on each iteration

- L_2 attack is conducted on the pixels in the **allowed set**.
- Let δ be the solution returned from L_2 attack on input image x, so that $x + \delta$ is an adversarial example.
- The **gradient of the objective function**, evaluated at the adversarial instance $g = \nabla_x f(x + \delta)$.
- The attack selects pixel $i = \underset{i}{argmin}\ g_i.\delta_i$ and fix i, i.e., **remove** i **from the allowed set**.

L_0 Attack

The $\boldsymbol{L_0}$ **distance metric is non-differentiable** and therefore is ill-suited for standard gradient descent. Instead, An iterative algorithm is used in each iteration.

In more detail, on each iteration

- L_2 attack is conducted on the pixels in the **allowed set**.
- Let δ be the solution returned from L_2 attack on input image x, so that $x + \delta$ is an adversarial example.
- The **gradient of the objective function**, evaluated at the adversarial instance $g = \nabla_x f(x + \delta)$.
- The attack selects pixel $i = \underset{i}{argmin}\ g_i.\delta_i$ and fix i, i.e., **remove** i **from the allowed set**.
  - The intuition is that $g_i.\delta_i$ tells us how much reduction to $f(.)$ we obtain from the ith pixel of the image, when moving from x to $x + \delta$ (Taylor expansion: $f(x) = f(x_0) + g^T\delta = f(x_0) + \sum_{i=1}^{n} g_i\delta_i$)
  - 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.
  - Selecting the index i that minimizes δ_i is simpler, but it yields results with 1.5× higher L_0 distortion.

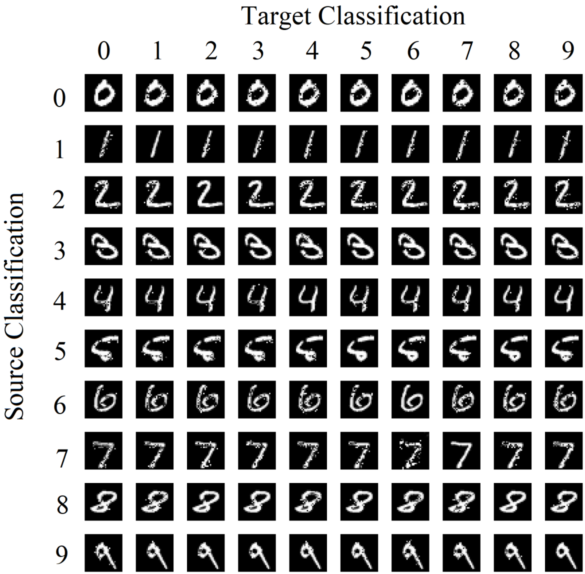
L_0 Attack

The $\boldsymbol{L_0}$ **distance metric is non-differentiable** and therefore is ill-suited for standard gradient descent. Instead, An iterative algorithm is used in each iteration.

In more detail, on each iteration

- L_2 attack is conducted on the pixels in the **allowed set**.
- Let δ be the solution returned from L_2 attack on input image x, so that $x + \delta$ is an adversarial example.
- The **gradient of the objective function**, evaluated at the adversarial instance $g = \nabla_x f(x + \delta)$.
- The attack selects pixel $i = \underset{i}{argmin}\ g_i.\delta_i$ and fix i, i.e., **remove** i **from the allowed set**.
  - The intuition is that $g_i.\delta_i$ tells us how much reduction to $f(.)$ we obtain from the ith pixel of the image, when moving from x to $x + \delta$ (Taylor expansion: $f(x) = f(x_0) + g^T\delta = f(x_0) + \sum_{i=1}^{n} g_i\delta_i$)
  - 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.
  - Selecting the index i that minimizes δ_i is simpler, but it yields results with 1.5× higher L_0 distortion.
- This process **repeats until the** L_2 **attack fails** to find an adversarial example.

L_0 Attack


Target Classification
0 1 2 3 4 5 6 7 8 9
Source Classification
0
1
2
3
4
5
6
7
8
9

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

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

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.

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

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

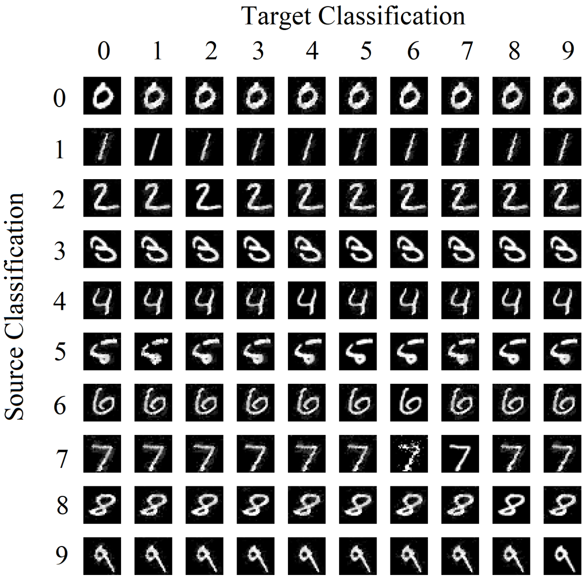
gradient descent produces very poor results: the $\|\delta\|_\infty$ term **only penalizes the largest (in absolute value) entry** in δ and has no impact on any of the other.

To solve this issue, the L_∞ term in **the loss function is replaced** by a penalty for any δ_i that exceed τ (initially 1, decreasing in each iteration). The new loss term **penalizes all large values** simultaneously. Following minimization is solved in each iteration

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

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

L_∞ Attack



Evaluation

	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 IV
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.

Evaluation

	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 V
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.