Trustworthy AI

Spring 2024

Yuan Tian

#3: Adversarial Machine Learning and Defenses-continued

Recall: Our Optimization Problem

Two steps:

Step 1: Define an objective function $\boldsymbol{obj_t}$ such that:

$$\text{if } \boldsymbol{obj_t(x+\eta) \leq 0} \text{ then } \boldsymbol{f\,(x+\eta) = t}$$

Step 2: Solve the following optimization problem:

find	η
minimize	$\|\boldsymbol{\eta}\|_\infty + c \cdot obj_t(\boldsymbol{x+\eta})$
such that	$\boldsymbol{x+\eta} \in [\mathbf{0},\mathbf{1}]^n$

Hard Box Constraint

Dealing with Constraints

find $\boldsymbol{\eta}$

minimize $\|\boldsymbol{\eta}\|_p + c \cdot obj_t(\boldsymbol{x} + \boldsymbol{\eta})$

such that $\boldsymbol{x} + \boldsymbol{\eta} \in [\mathbf{0}, \mathbf{1}]^n$

Given $\boldsymbol{x}$ is constant, this is the same as enforcing $\boldsymbol{\eta_i} \in [-\boldsymbol{x_i}, 1 - \boldsymbol{x_i}]$ for every $\boldsymbol{\eta_i}$. We can then use either of these two methods:

Projected gradient descent (PGD)

"Fit" all coordinates to be within the box

$$\boldsymbol{project}((\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_n)) = \left(clip_1(\boldsymbol{\eta}_1), \ldots, clip_n(\boldsymbol{\eta}_n)\right)$$

$$clip_i(\boldsymbol{\eta_i}) = \begin{cases} -\boldsymbol{x_i} & \text{if } \boldsymbol{\eta_i} < -\boldsymbol{x_i} \\ \boldsymbol{\eta_i}, & \text{if } \boldsymbol{\eta_i} \in [-\boldsymbol{x_i}, 1 - \boldsymbol{x_i}] \\ 1 - \boldsymbol{x_i}, & \text{if } \boldsymbol{\eta_i} > 1 - \boldsymbol{x_i} \end{cases}$$

LBFGS-B optimizer:

Used by Carlini & Wagner

pass each $\boldsymbol{\eta_i} \in [-\boldsymbol{x_i}, 1 - \boldsymbol{x_i}]$ separately to the optimizer.

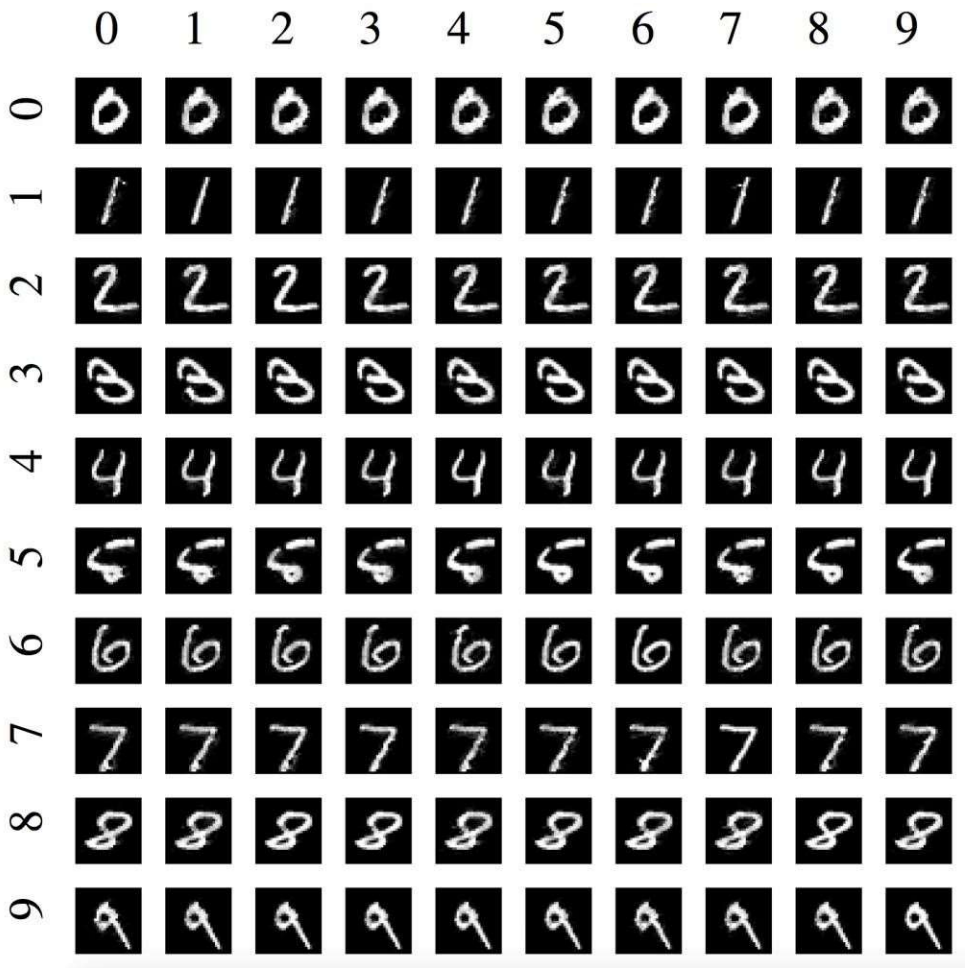
"**-B**" stands for box constraints

Note: if we also want $\|\boldsymbol{\eta}\|_\infty < e$ then we can also add the box constraints $\boldsymbol{\eta_i} \in [-e, e]$

With this approach we get

Target label

Initial label



What we see is that on the MNIST (digit recognition) data set it is not difficult to get a realistic looking image that fools the neural network classifier…

Another attack…often used during training

- So far, we looked at FGSM as well as an attack to minimize the distance to the original input (e.g., image, audio)
- Now, we illustrate another attack, a variant of FGSM applied iteratively **with projection**.
- The attack uses Projected Gradient Descent (PGD) and is referred to as a PGD attack.
- This is a commonly used attack for adversarial training: training the network to be robust.

Illustrating the PGD attack

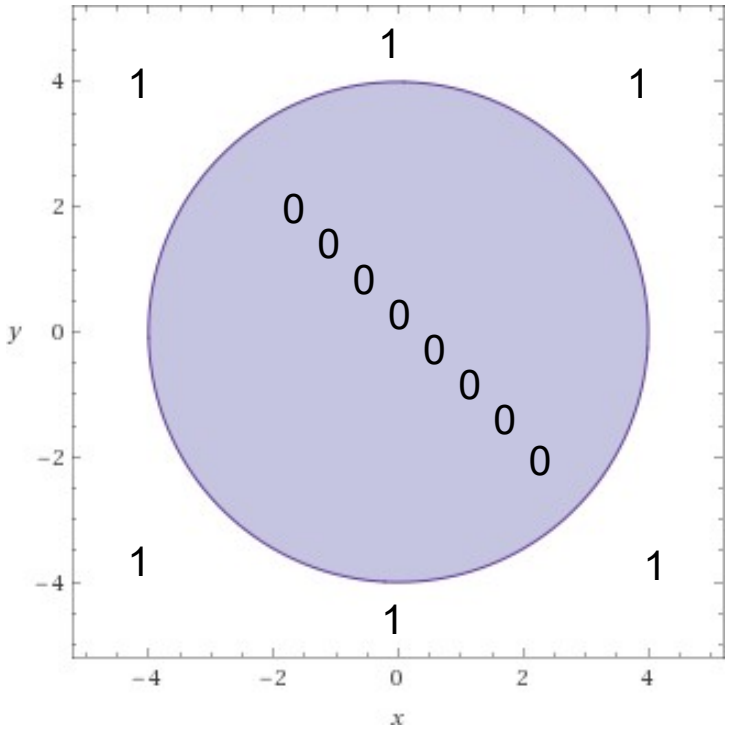
Given a **dataset** of points (x, y) where label is:

0 if $x^2+y^2<16$

1 otherwise

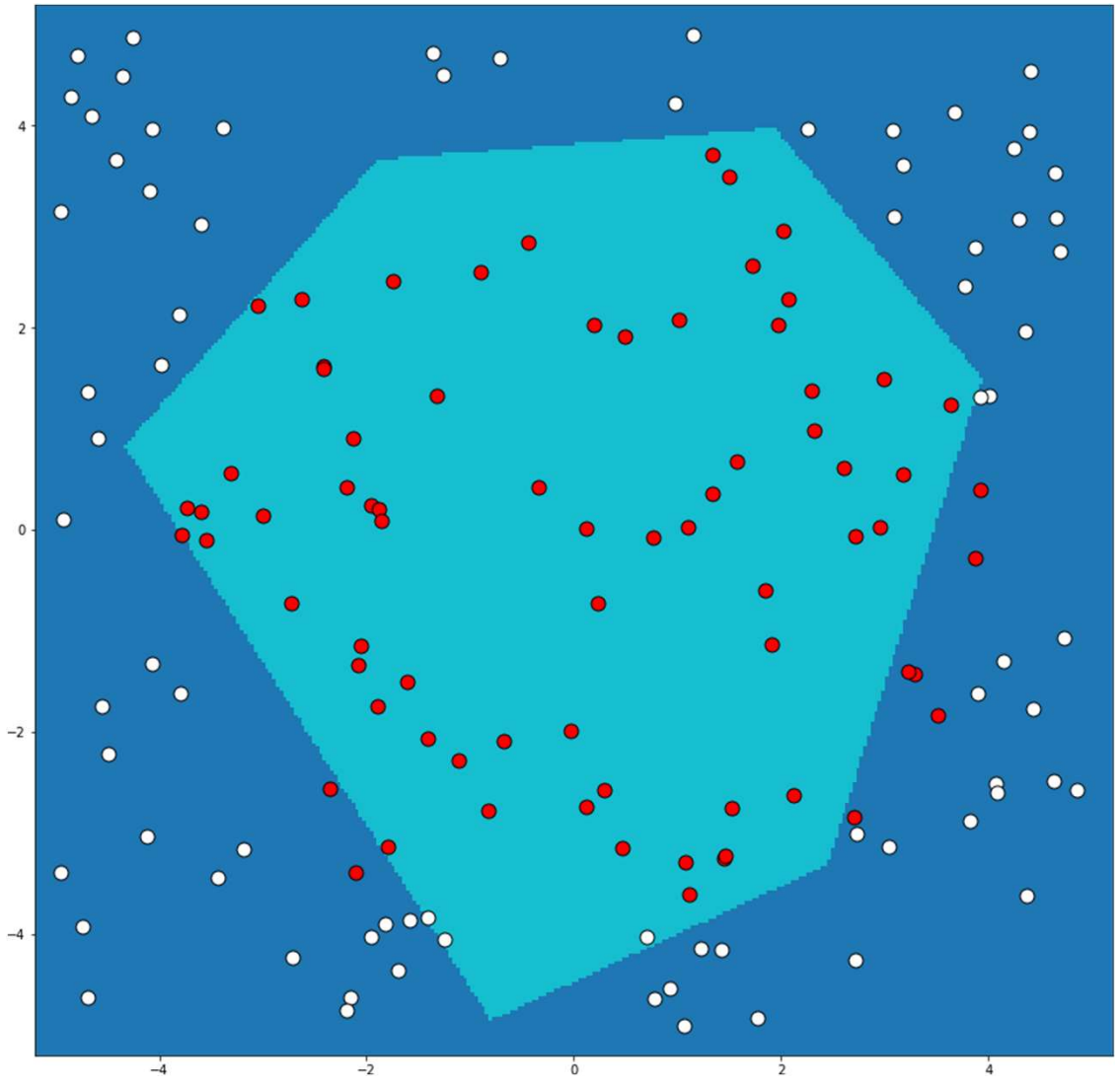
train a neural network to classify the points correctly

Illustrating the PGD attack



https://www.wolframalpha.com/input/?i=plot+x%5E2+%2B+y%5E2+%3C+16

After training we get the classifier:



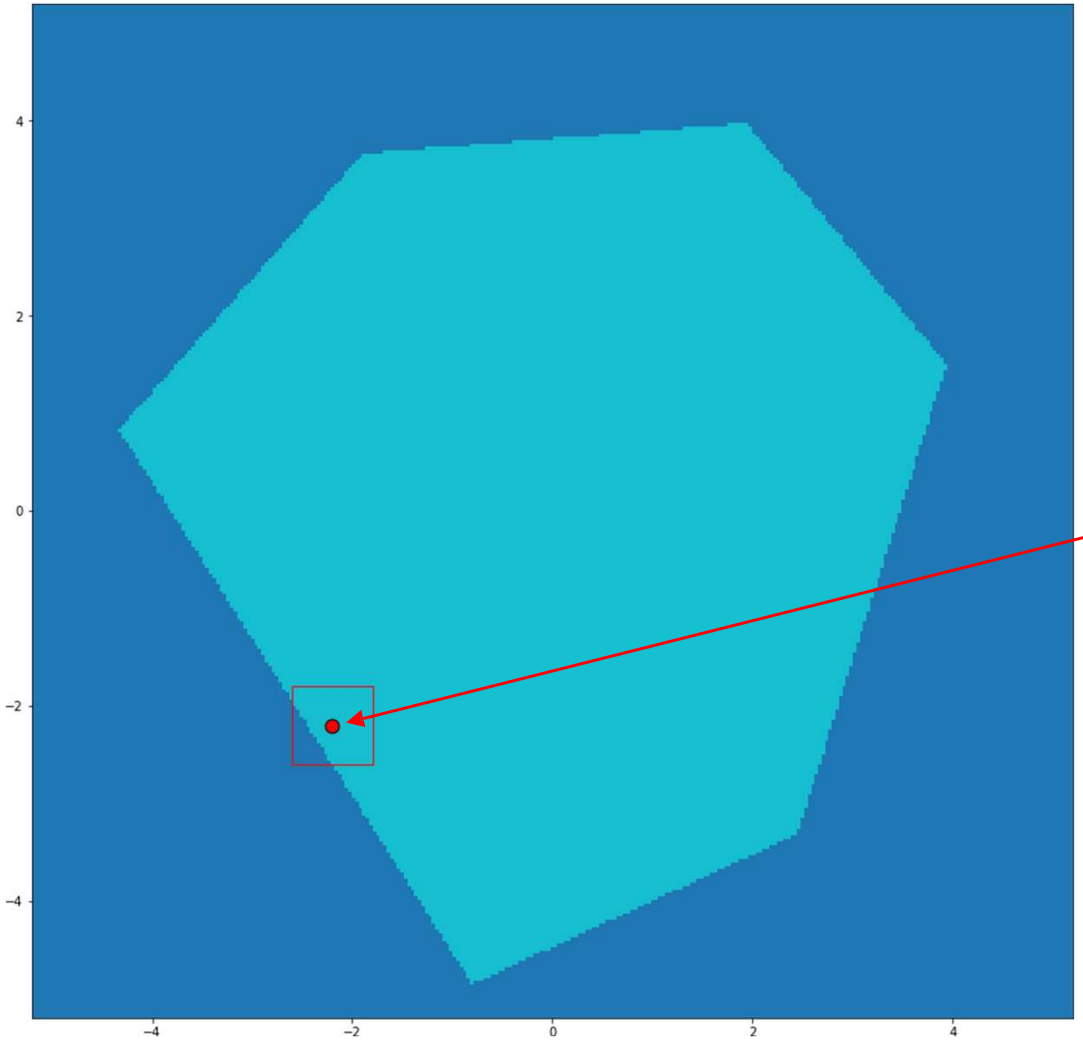
Dark blue – neural network predicts 1 (property does not hold)

Light blue – neural network predicts 0 (property holds)

Red dots – those where property actually holds

White dots – those where property actually does not hold

Lets pick a point…



Goal:

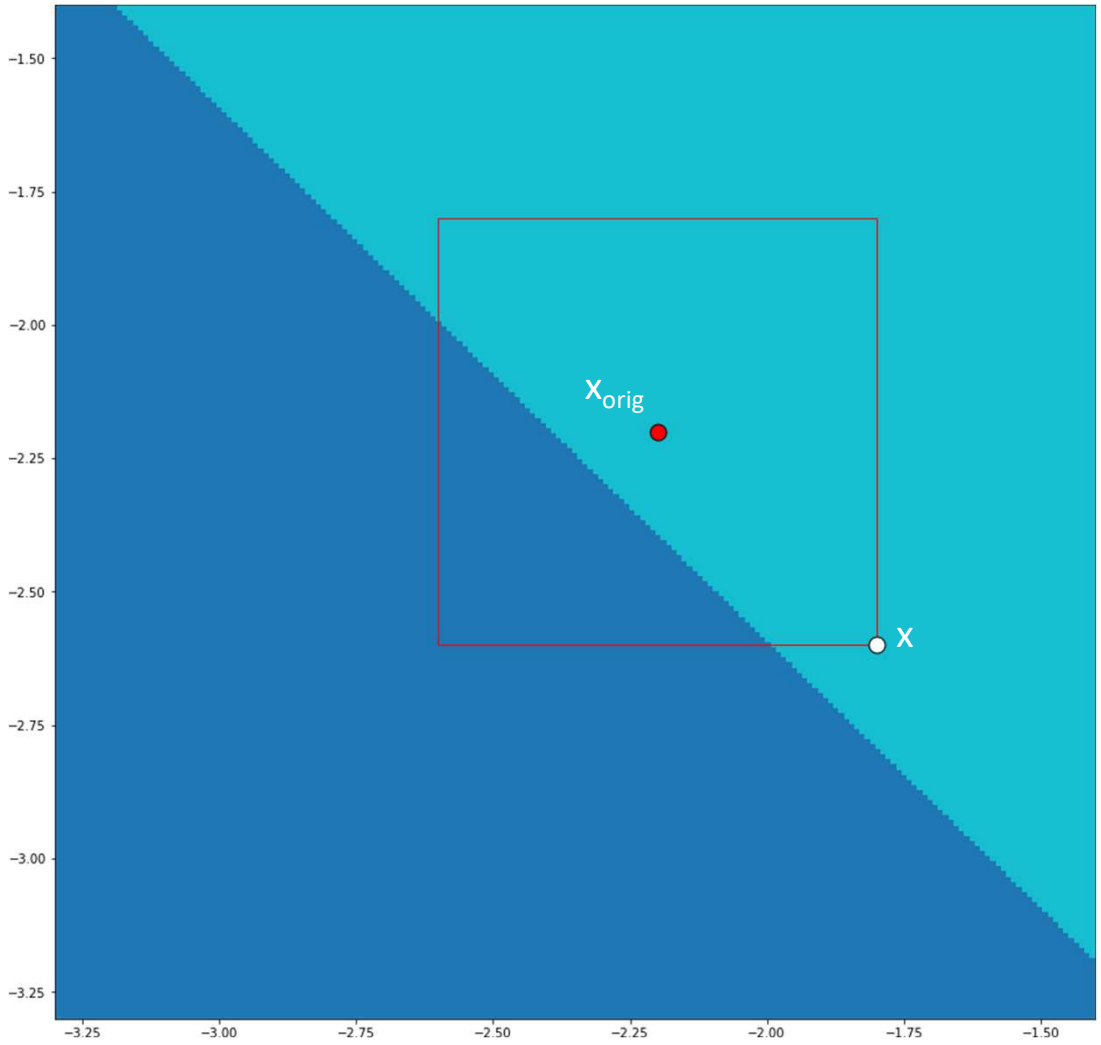
Find adversarial input in

L_{inf} ball around:

x_{orig} = (-2.2, -2.2)
(red point)

with ε=0.4

Lets Zoom in a bit…

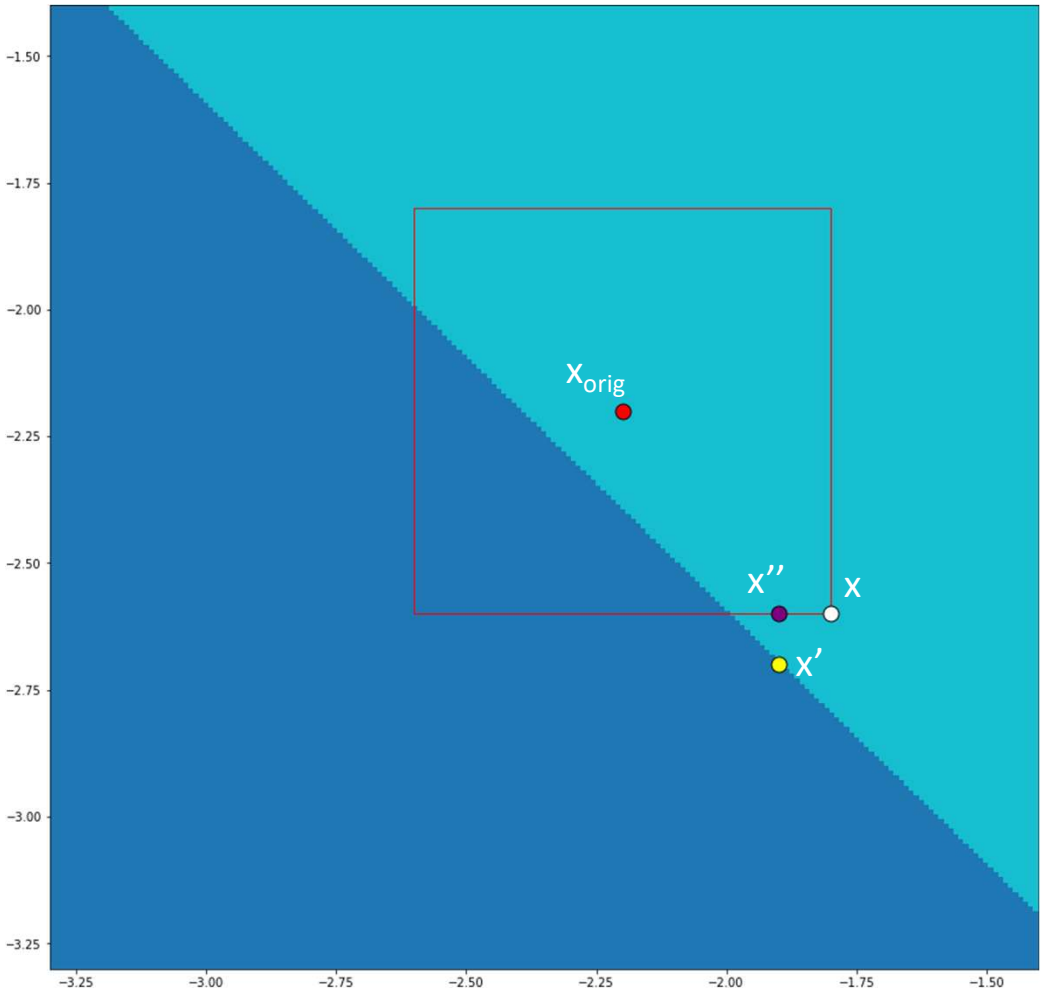


Initialize PGD with:

x = (-1.8, -2.6)

Note: this is just for the example to illustrate projection. In practice, one picks a point at random in the box

PGD Iteration 1



NN(x) = [0.5973, 0.4027]

Loss(x) = 0.5153

∇_x Loss(x) = [-0.852, -1.373]

Change Δ

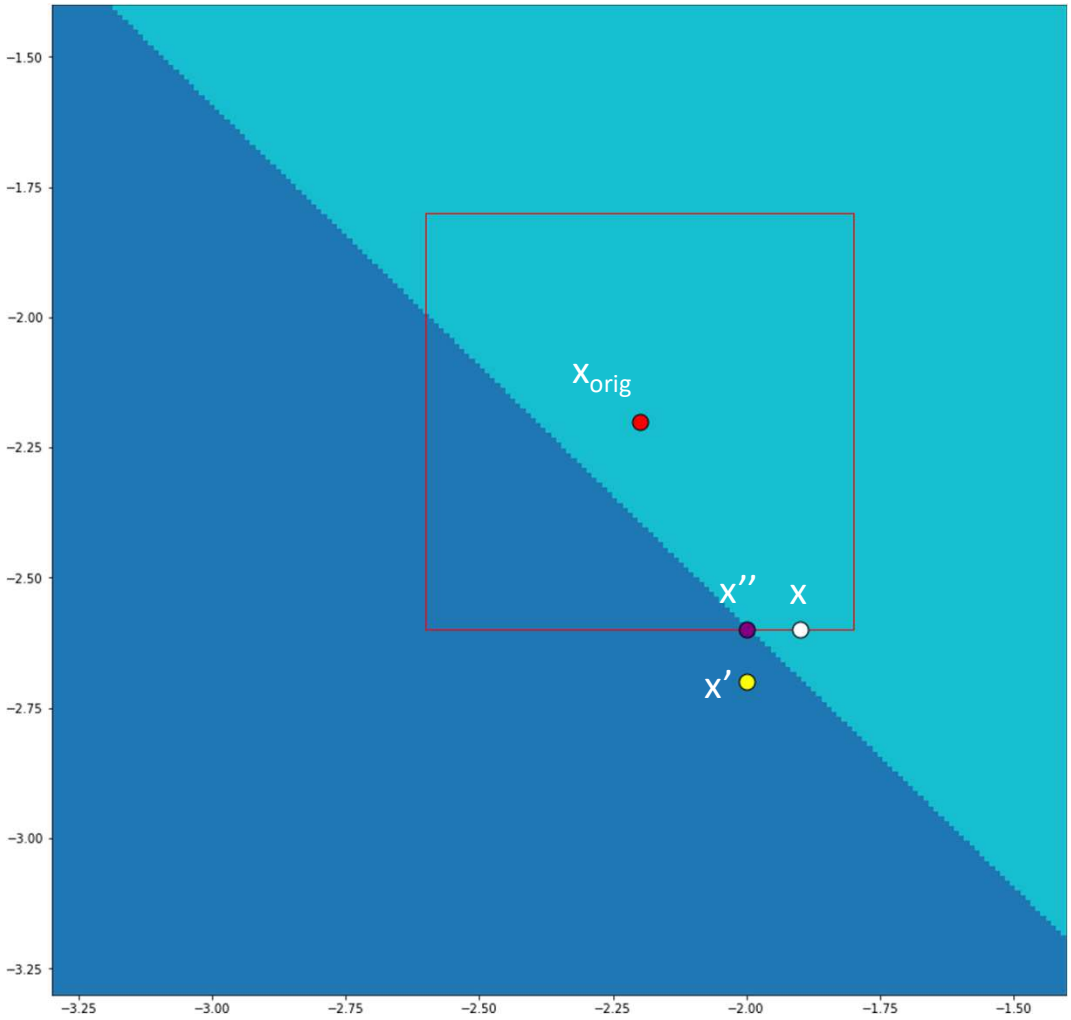
$x' = x + 0.1 * \text{sign}(\nabla_x \text{Loss}(x))$
= [-1.9, -2.7] (yellow point)

Up-to-here, its just standard untargeted FGSM attack but with **smaller step-size** of 0.1 than ε which is 0.4.

But now we also **project**:

$x'' = \textbf{project}(x', x_{orig}, \varepsilon)$
= [-1.9, -2.6] (purple point)

PGD Iteration 2



x'' from before now named x:

NN(x) = [0.5455, 0.4545]
(so point x = (-1.9, -2.6) is
not yet a counter example

Loss(x) = 0.6060

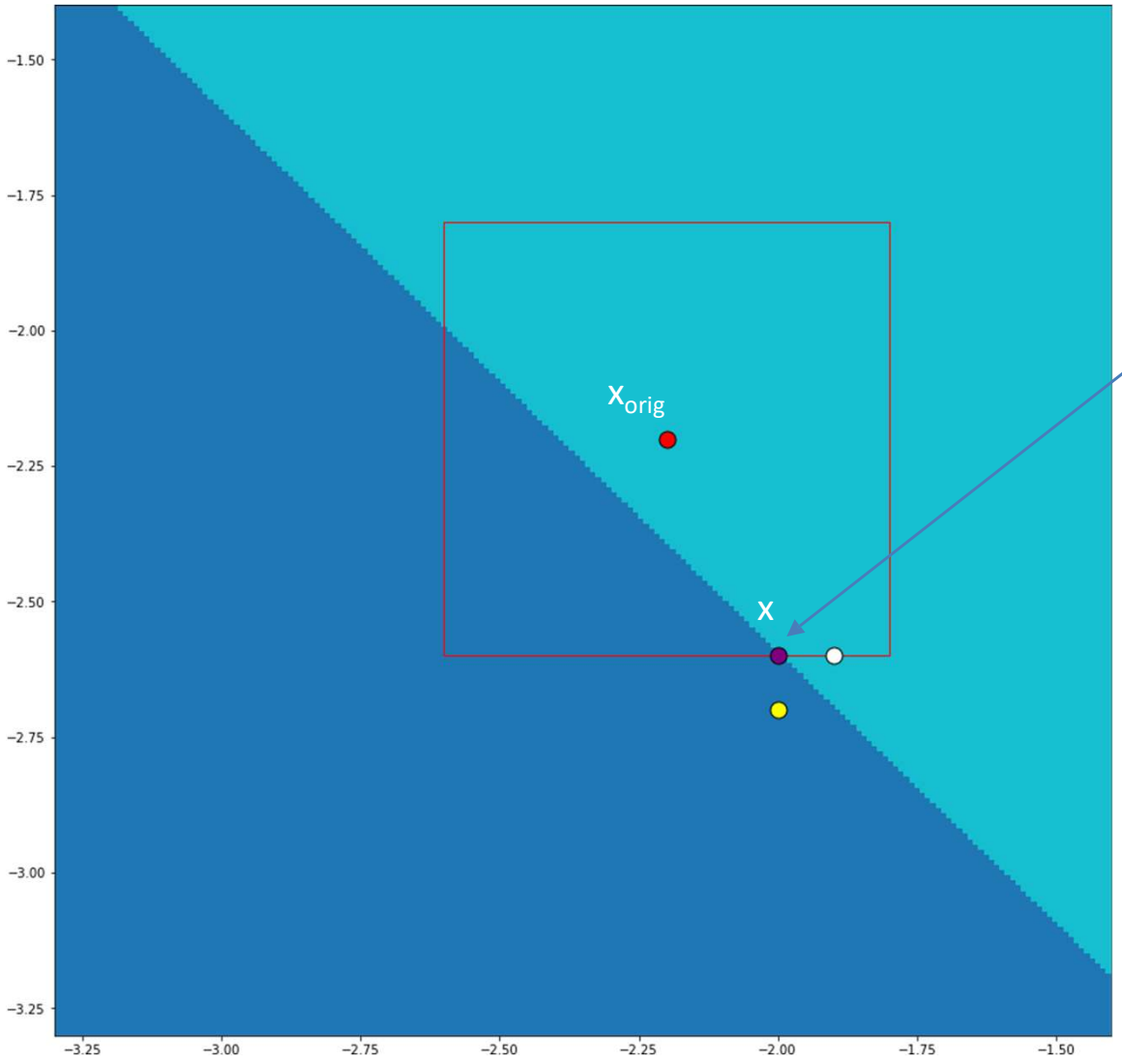
∇_x Loss(x) =[-0.9621, -1.5493]

Change Δ

$x' = x + 0.1 * \text{sign}(\nabla_x \text{Loss}(x))$
$= [-2, -2.7]$

$x'' = \textbf{project}(x', x_{orig}, \varepsilon)$
$= [-2, -2.6]$

PGD Iteration 3



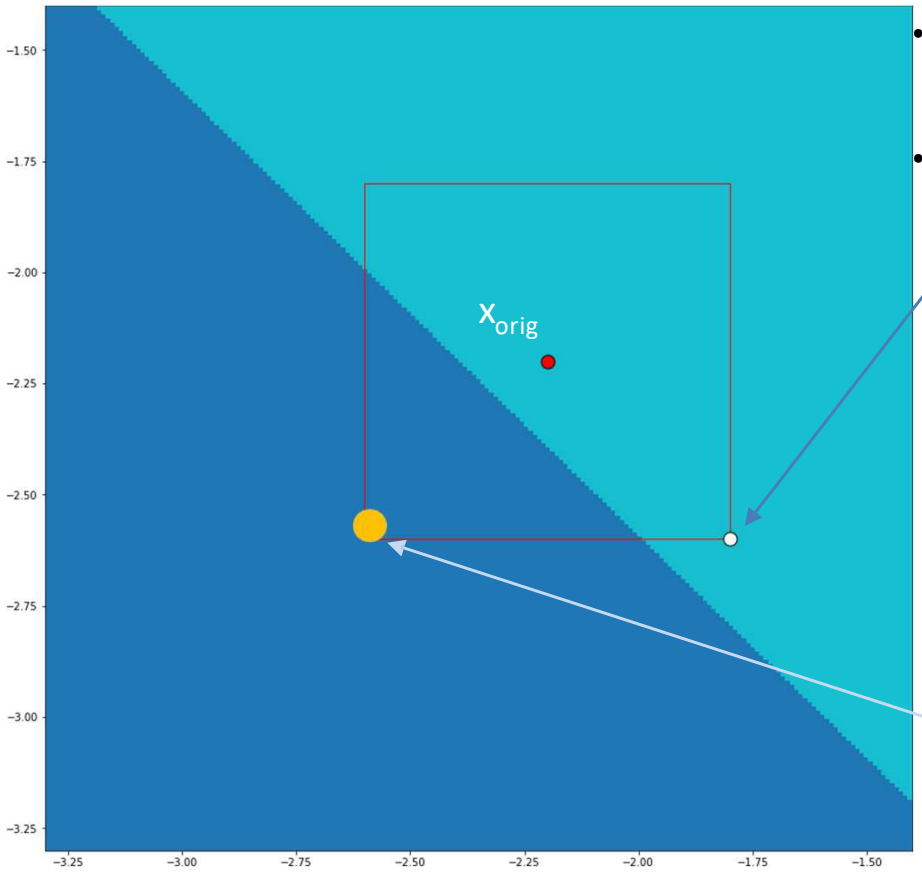
NN(x) = [0.4927, 0.5073]

found adversarial example

x = [-2, -2.6]

Neural network predicts 1, although $(-2)^2 + (-2.6)^2 < 16$ so it should have been classified as 0

Some notes on PGD



- The goal of the PGD attack is to **find a point in the region which maximizes the loss** (it may still classify to the same label as x_{orig})
- For our example, we started at the corner. Typically one starts the search with a **random point inside the box**.
- One stops PGD after **a pre-defined number of iterations** (e.g., 10).
- In our example, we always stepped outside the box to illustrate projection, and then projected to the box. It is possible to never step outside the box and thus **projection will have no effect**.
- It is possible the final produced example **is inside the box**, and not on the boundary. However, when we project, if outside the box, we will end up on the boundary.
- In this example, loss is **likely to be highest** somewhere around the big orange point (typically far from the decision boundary). Of course, when we are searching, we **don't know the actual decision boundary**.
- One can implement PGD in **two ways**:
  - **a)** by projecting current **point** x' to the ε-box around x_{orig} as well as [0,1] for each dimension, or
  - **b)** by projecting the **change** Δ to [-ε, +ε] as well as to the constraints needed so each element in the resulting point is between [0,1] (see slide 3 in this lecture)
- Step size (in our example 0.1) is **typically smaller than ε** (in FGSM it is ε).

- Projection is linear-time in the dimension for L_∞ and L_2 norms.
- An **open problem:** finding efficient projections for various convex regions that are **more expressive than boxes** (e.g., convex polyhedral restrictions).

What are the key differences of these attacks?

Summary of adversarial attacks

Attack Type	Region	Optimization	Outcome
FGSM (targeted, untargeted)	Change $\boldsymbol{\eta}$ fixed to [-ε, +ε].	Take exactly one ε-sized step	Produced example will be on boundary of region.
PGD (typically untargeted, but can be targeted)	Can be instantiated with any region one can project to.	Take many steps. Uses projection to stay inside region. For special case of l_∞, step size smaller than ε.	Result will be inside region. Tries to maximize loss.
C&W [Images] (presented as targeted)	No real restriction, except image has to be in [0,1] (like all other methods). This restricts the region for the change $\boldsymbol{\eta}$: $\boldsymbol{\eta}$ has to be bounded s.t. original image + $\boldsymbol{\eta}$ stays in [0,1].	Aims to produce a change $\boldsymbol{\eta}$ with small l_∞. Takes many steps, using LBFGS-B to ensure $\boldsymbol{\eta}$ stays in bounds.	Result will be inside [0,1], with a hopefully small l_∞ distance from original image.

Can we Avoid Adversarial Examples?

Many works have tried to, but follow-up works showed that all fail

The main **successful defenses** in practice now incorporate adversarial examples during training

Some pretty good experimental defenses exist

Adversarial Accuracy vs. Test Accuracy

Adversarial accuracy refers to a metric on the test set where for each data point we check if the network classifies the point correctly **and** the network is robust in a region around that point.

Example [l_∞ ball]: Let ϵ =0.3 , and let the test set T contain 100 examples. For each example $d_i \in T$, lets check if in the l_∞ region of size ≤ 0.3 around d_i, we find an (adversarial) example with a different classification than d_i. For that purpose we typically use a **PGD attack**. Now suppose, 95 of the 100 examples classify correctly and for 15 of these 95, we find an adversarial example. Then, our adversarial accuracy will be $\frac{8\#}{\$\#\#} = 80\%$ and our test accuracy will be $\frac{\%\&}{\$\#\#} = 95\%$.

Adversarial accuracy and Test accuracy can be at odds: it is possible to raise the adversarial accuracy which tends to lower test accuracy. This trade off is being **actively investigated**.

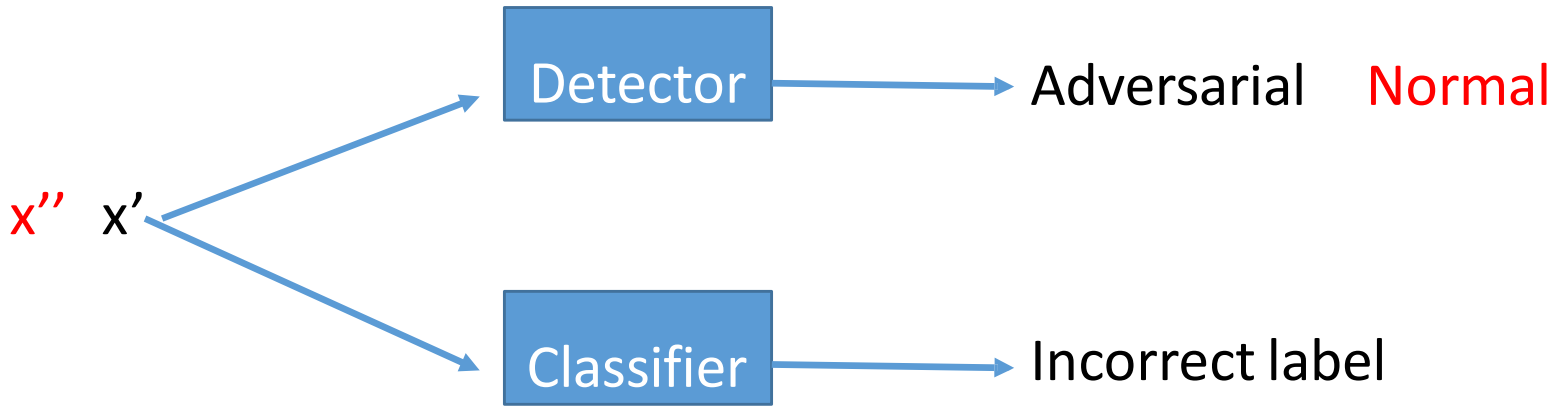
Defending against adversarial examples

- General philosophy for security solutions
  - Prevention
  - Detection
  - Response
- Prevention
  - robust classifiers
- Detection
  - detecting adversarial examples
- Response
  - manual labeling?
  - collecting more data?
  - Gradient masking?

Detecting adversarial examples

- Binary classification
  - Normal example vs. adversarial example
- Add one more label “adversarial”
  - E.g., 0, 1, 2, …, 9, adversarial
- Extracting features and building detectors

Challenges of detecting adversarial examples



Attackers are adaptive

Response

- Manual labeling
- Collecting more data
  - Other sensor data
- Gradient masking

Gradient Masking

Many trivial ways to hid the gradient.

- Changing the model to return most likely class and not the probability.
- infinitesimal changes in the input will not change the output at all.

But are we making our model more robust?

We are just giving the adversary fewer clues to figure out the holes in the model.

Limitation:

- The defender might increase the attacker's cost by training models with higher input dimensionality or modeling complexity leading to increased number of queries to train the substitute.

Prevention – robust classifiers

- Empirically robust classifier
  - A particular attack cannot find adversarial example within a L_p norm ball
  - (p, s)-robust against an attack for x, if the attack does not find adversarial perturbation whose L_p norm is no larger than s.

- Certifiably robust classifier
  - No adversarial examples exist within a L_p norm ball.
  - (p, s)-certifiably robust for x, if no adversarial perturbation whose L_p norm is no larger than s exists.

Defense as Optimization Problem

Outer minimization problem

find $\boldsymbol{\theta}$ that minimizes the high loss, aiming to train a robust classifier

find θ

minimize $\rho(\theta)$

where $\rho(\theta) = \mathbf{E}_{(\mathrm{x},\mathrm{y})\sim D}\left[\max_{x' \in S(x)} L(\theta, x', y)\right]$

Inner maximization problem

find $\boldsymbol{x'}$ that achieves high loss

D is the underlying distribution

$\mathbf{E}_{(\mathrm{x},\mathrm{y})\sim D}$ is typically estimated with the empirical risk

$S(x)$ denotes the perturbation region around point x, that is, we want all points in $S(x)$ to classify the same as x . We can pick $S(x)$ to be:

Madry et.al, 2017

$$S(x) = \{ \boldsymbol{x}' \mid \|\boldsymbol{x} - \boldsymbol{x}'\|_p < \boldsymbol{\epsilon} \}$$

PGD Defense in Practice

Step 1: select a mini-batch B of examples from dataset D.

Step 2: compute B_{max} by applying PGD attack (actually computes an approximation) as follows to every point $(x, y) \in B$:

$$x_{max} = \underset{x' \in S(x)}{\mathbf{argmax}} \; L(\theta, x', y)$$

Note: x_{max} need not be adversarial example; it just aims to maximize L

Step 3: solve outer problem:

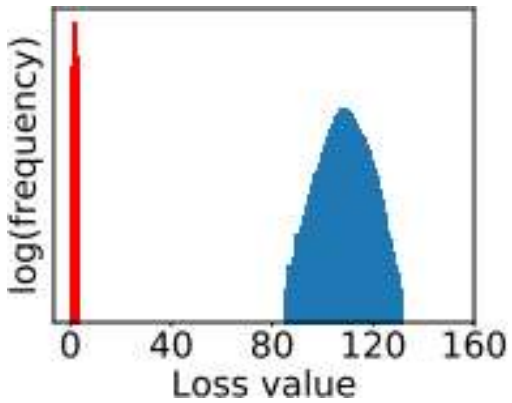
$$\theta' = \theta - \frac{\$}{|B_{max}|} \sum_{(x_{m\,a}\;, y) \in B_{m\,a}} \nabla_{\$} L(\theta, x_{max}, y)$$

Step 4: goto Step 1. Various stopping criteria, including reaching a certain number of epochs.

*The conversion of the original min-max problem to the 4 steps above is based on Danskin's theorem

Why do we think we can find a good approximate solution to the inner maximization problem?

Experiments show that many local maxima inside $S(x)$ have well-concentrated loss values. This is inline with why we believe neural network training is possible (many local minima with similar values).



This graph is for a **single example**: goal is to maximize the cross-entropy loss measured for 100,000 random starting points in $S(x)$.

The red graph indicates the value of the loss L for an adversarially trained network.

The blue graph is for the loss L of a non-adversarially trained network.

Points to Consider when Defending

Model capacity matters: larger networks are more defendable and less easy to be attacked with transferrable examples. Training smaller nets with PGD has negative effects on accuracy.

Training with **adversarial examples from PGD attacks (many steps and project)** tends to perform better than training with adversarial examples from FGSM attacks (one step, no projection).

Even on larger networks, defenses can negatively affect accuracy (e.g. CIFAR). More research is needed here. By this we mean that after the network is trained, we test its accuracy on the test set. And there, it is more robust yet more points classify incorrectly.

"No free lunch in adversarial robustness", Tsipras et. al. 2018
Proves that if we want robust model, decrease in standard accuracy is inevitable!

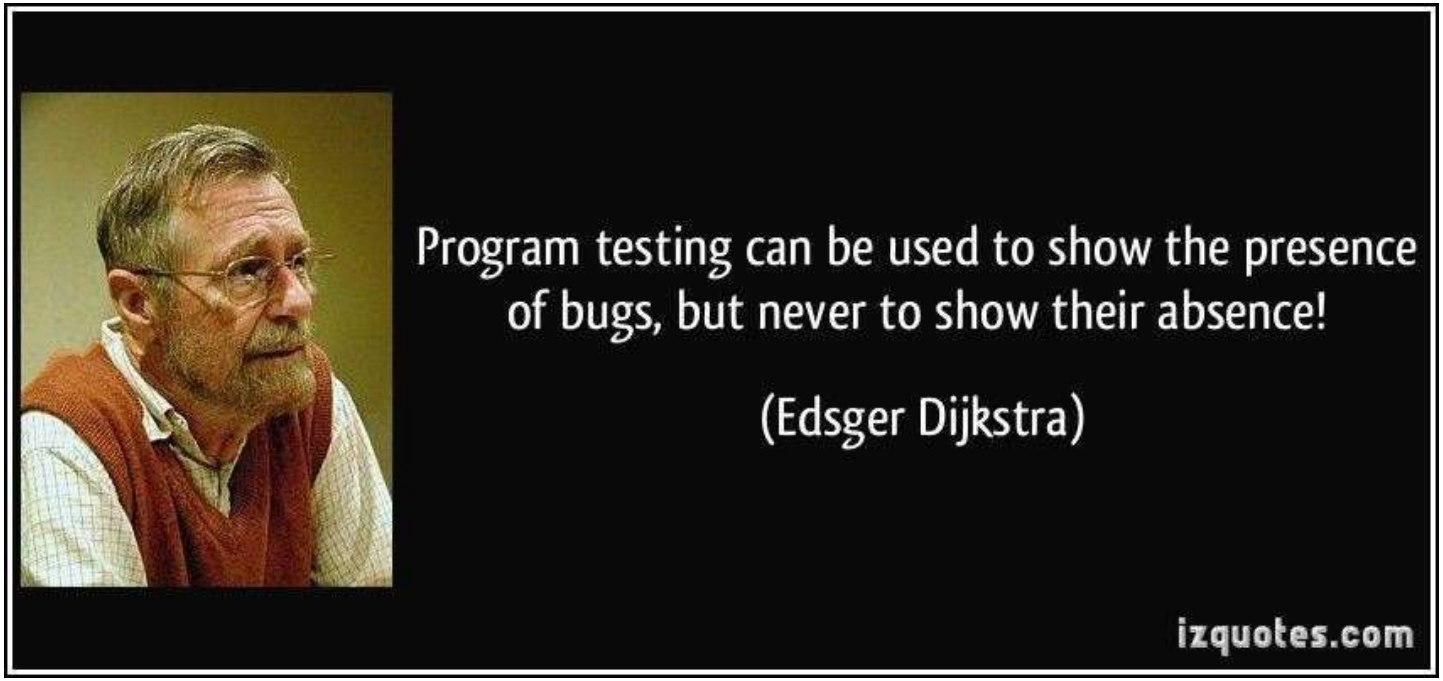
"Adversarially Robust Generalization Requires More Data ", Schmidt et. al. 2018
Provides lower bound on number of samples needed to achieve adversarial robustness

"Theoretically Principled Trade-off between Robustness and Accuracy", Zhang et.al, 2019
Improves slightly on the PGD defense; also combines with standard (e.g., cross-entropy) loss.

Issues of adversarial training

- No certifiable guarantee
- May not be empirically robust against unseen attacks
  - Use multiple attacks during training
- May not be robust to perturbation larger than s used in training

The fundamental problem



The attacks and defenses so far are similar to testing: they may work well in practice sometimes, but provide no formal guarantees.

More generally we want

Automated verifier to prove properties of realistic networks

Useful in:

- Certifying large cyber-physical systems that use NN
- Proving robustness of NN
- Learning interpretable specs of the NN
- Comparing NNs

Lecture on 04/22 will provide more discussions on this topic

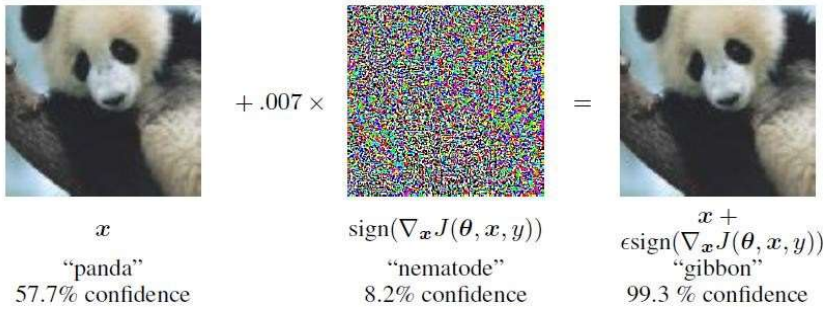
Lecture Summary

- We looked at a way to (experimentally) defend the network by training with adversarial examples, specifically the PGD defense. This results in a min-max nested optimization problem.

- Adversarial training can lower standard accuracy. Remains a question of research interest, how to avoid this from happening.

Lecture Summary

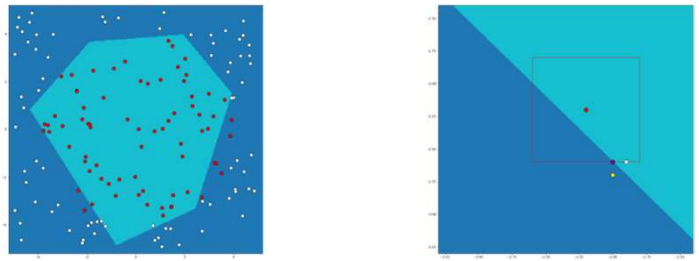
Deep Learning is susceptible to adversarial examples



Generating Adversarial examples (an optimization problem)

- FGSM
- C&W (minimize perturbation)
- PGD

An example of the PGD attack



- We looked at a way to (experimentally) defend the network by training with adversarial examples, specifically the PGD defense. This results in a min-max nested optimization problem.